A Theory of Formalisms for Representing Knowledge

Heng Zhang[1*], **Guifei Jiang**[2], **Donghui Quan**[1]

[1]Zhejiang Lab, Hangzhou, Zhejiang 311121, China
[2]College of Software, Nankai University, Tianjin 300350, China
h.zhang@zhejianglab.org, g.jiang@nankai.edu.cn, donghui.quan@zhejianglab.org

Abstract

There has been a longstanding dispute over which formalism is the best for representing knowledge in AI. The well-known "declarative vs. procedural controversy" is concerned with the choice of utilizing declarations or procedures as the primary mode of knowledge representation. The ongoing debate between symbolic AI and connectionist AI also revolves around the question of whether knowledge should be represented implicitly (e.g., as parametric knowledge in deep learning and large language models) or explicitly (e.g., as logical theories in traditional knowledge representation and reasoning). To address these issues, we propose a general framework to capture various knowledge representation formalisms in which we are interested. Within the framework, we find a family of universal knowledge representation formalisms, and prove that all universal formalisms are recursively isomorphic. Moreover, we show that all pairwise inter-translatable formalisms that admit the padding property are also recursively isomorphic. These imply that, up to an offline compilation, all universal (or natural and equally expressive) representation formalisms are in fact the same, which thus provides a partial answer to the aforementioned dispute.

Extended version — https://arxiv.org/abs/2412.11855

Introduction

Knowledge is extensively acknowledged as a cornerstone of intelligence (McCarthy and Hayes 1981; Delgrande et al. 2024), playing a crucial role in intelligent systems. How to effectively represent, acquire, utilize and evolve knowledge is undoubtedly one of the most critical parts of realizing artificial general intelligence (AGI). Knowledge representation serves as the foundation and starting point for all these tasks. In traditional knowledge representation and reasoning (KR), representations of knowledge are often regarded as "explicit, symbolic, declarative representations of information" (Delgrande et al. 2024). However, in this work, we will consider more general forms of knowledge representation.

Over the past seven decades, researchers have devoted substantial efforts to developing various knowledge representation formalisms. An incomplete list includes: monotonic logical systems such as Prolog (van Emden and Kowalski 1976) and description logics (Baader et al. 2017); non-monotonic logics such as circumscription (McCarthy 1980) and default logic (Reiter 1980); graph-based representations such as Bayesian networks (Pearl 1985) and semantic network (Sowa 1991), and parametrized models such as recurrent neural networks (Rumelhart, Hinton, and Williams 1986) and transformers (Vaswani et al. 2017). It is noteworthy that machine learning and knowledge representation are inherently intertwined; all learning algorithms are actually based on some formalisms for representing knowledge.

The quest for the best formalism of knowledge representation has sparked a longstanding dispute. A prime example is the "declarative vs. procedural" controversy, which centers on choosing between declarative statements and procedures as the primary means of representing knowledge. Similarly, the ongoing debate between symbolic AI and connectionist AI revolves around the question of whether knowledge should be represented explicitly or implicitly. In general, knowledge representation formalisms in machine learning, such as convolutional neural networks in deep learning and transformers in large language models, are implicit, while all logical formalisms in traditional KR are explicit.

In this work, we will undertake a systematic exploration of disputed issues, particularly the search for the optimal knowledge representation formalism. A general framework is needed to systematically evaluate varied formalisms. While extensive philosophical deliberations on fundamentals of knowledge representation exist, including the physical symbol system hypothesis (Newell and Simon 1976) and the knowledge representation hypothesis (Smith 1982), they primarily remain within the realm of theoretical consensus-building. Departing from this path, our goal is to obtain rigorous conclusions from a broad and inclusive framework through meticulous mathematical demonstrations, thereby deepening the understanding of the nature of representation.

The main contributions of this work are threefold. Firstly, we propose a general framework to capture all the knowledge representation formalisms in which we are interested, and propose a novel definitions for universal formalisms. Secondly, we find a family of universal formalisms; based on them, we then prove that all possible universal formalisms are recursively isomorphic. Thirdly, we show that, under a natural condition, all subrecursive formalisms that can be

*Corresponding author.

translated into each other are recursively isomorphic. These results show us that, if an offline compilation is allowed, almost all the natural representation formalisms with the same expressive power can be regarded as the same, which thus provides us a partial answer to the aforementioned dispute.

Conventions and Notations

Suppose A, B and C are sets. By $p : A \rightharpoonup B$ we denote that p is a *partial function* (or *mapping*) from A to B. We call p a partial function *on* A if $A = B$. Let $dom(p)$ and $ran(p)$ denote the *domain* and *range* of p, respectively. In addition, we call p a *function* from A to B, written $p : A \to B$, if p is *total*. Given $X \subseteq A$, by $p|_X$ we denote the *restriction* of p to X. A partial function q is called an *extension* of p, or equivalently, q *extends* p, if p is a restriction of q. Given functions $p : A \to B$ and $q : B \to C$, by $p \circ q$ we denote the *composition* of p and q. All other notions of functions such as injectiveness, surjectiveness and bijectiveness are standard.

The reader is assumed familiar with logic. A *signature* is a set that consists of *predicate* and *function symbols*, each associated with a nonnegative integer, called its *arity*. *Constants* are nullary function symbols. Let σ be a signature. By σ*-atoms* and σ*-sentences* (or *sentences of* σ) we denote atoms and sentences, respectively, built from σ and standard logical connectives and quantifiers as usual. Let FO and SO denote the classes of first-order and second-order sentences, respectively. A *structure of* σ (or simply, σ*-structure*) $\mathcal{A}$ is armed with a nonempty *domain* A, maps each predicate symbol $P \in \sigma$ to a relation $P^{\mathcal{A}}$ on A, and maps each function symbol $f \in \sigma$ to a function $f^{\mathcal{A}}$ on A, both are of the same arity. A *UNA-structure* of σ is a σ-structure $\mathcal{A}$ that satisfies the unique name assumption, i.e., $c^{\mathcal{A}} \neq d^{\mathcal{A}}$ for every pair of distinct constants c, d in σ. Given $\upsilon \subseteq \sigma$, let $\mathcal{A}|_\upsilon$ denote the *restriction* of $\mathcal{A}$ to υ. We call $\mathcal{A}$ a σ*-expansion* of $\mathcal{B}$ if $\mathcal{B} = \mathcal{A}|_\upsilon$ for some $\upsilon \subseteq \sigma$. Let ϕ be a σ-sentence. We write $\mathcal{A} \models \phi$ if $\mathcal{A}$ is a *model* of ϕ. Given a class $\mathbb{C}$ of σ-structures, we write $\mathbb{C} \models \phi$ if $\mathcal{A} \models \phi$ for all $\mathcal{A} \in \mathbb{C}$. Given a set Σ of sentences and a sentence ψ, we write $\Sigma \vDash \phi$ and $\psi \vDash \phi$ if ϕ is a *logical implication* of Σ and ψ, respectively.

Every *Turing machine* M is armed with a two-way infinite tape, a reading head, a finite set Q of states and a fixed symbol set $\{0, 1, B\}$. There is exactly one *starting state* and at least one *halting state* in Q. Every halting state is either an **yes** state or a **no** state, but cannot be both. Both the input and output are strings in $\{0,1\}^*$, stored in the tape, starting from the position of the reading head and ending with B. Let $L \subseteq \{0,1\}^*$. We say M *accepts* L if, for every $\pi \in \{0,1\}^*$, M *accepts* π (i.e., M on input π halts at an **yes** state) if $\pi \in L$, and never halts otherwise; and M *decides* L if, for every $\pi \in \{0,1\}^*$, M accepts π if $\pi \in L$, and rejects π otherwise. We say L is *recursively enumerable* (respectively, *recursive*) if it is accepted (respectively, decided) by some Turing machine. Moreover, Turing machines can also be used to compute functions. We say a partial function p from $\{0,1\}^*$ to $\{0,1\}^*$ is *computed* by M if, given $\pi \in \{0,1\}^*$ as input, M halts with the output ω iff p is defined on π and $p(\pi) = \omega$. We say p is *partial recursive* if it is computed by some Turing machine, and p is *recursive* if it is partial recursive and $dom(p)$ is recursive.

To simplify the presentation, we will fix $[\![\cdot]\!]$ as an injective mapping that maps every finite object to a string in $\{0,1\}^*$. For example, given a Turing machine M, by $[\![M]\!]$ we denote the encoding of M in $\{0,1\}^*$. Moreover, we require that both $[\![\cdot]\!]$ and its inverse can be effectively obtained.

Framework

The major goal of this work is to carry out a careful comparison between different formalisms for representing knowledge. To this end, we have to propose a general framework that captures all the formalisms in which we are interested.

To build the desired framework, one immediate thought might be to define a family of abstract logical formalisms, similar to abstract logical systems proposed for establishing Lindström's theorem (Lindström 1969). But we do not pursue this approach in this work. The main reasons are as follows. Firstly, the framework established in this way is not general enough. It is important to note that logic is not the only method for representing knowledge. Secondly, from a user's perspective, the internal logical semantics of a representation formalism are actually not important. Users are primarily concerned with the outputs generated from given inputs. This aligns with a behaviorist perspective.

Regarding the primary computational task, we will focus on knowledge reasoning. While there are certainly other important tasks, such as knowledge acquisition (learning) and maintenance, knowledge reasoning typically runs online, with its efficiency directly determining the performance of the underlying system. In contrast, knowledge acquisition and maintenance can in general be performed offline.

We aim to go beyond the traditional reasoning problem to tackle a more general computational problem, known as *query answering* (QA). The problem of QA has been extensively studied in databases, see, e.g., (Fagin et al. 2005) and was later introduced into KR to implement data-intensive knowledge reasoning, see, e.g., (Calvanese et al. 2007).

The problem of QA in KR is defined as follows:

> Given a database D, a knowledge base K and a query ϕ, determine whether ϕ is inferable from D and K.

Intuitively, D stores the *observed facts*, ϕ describes the *question* that the user want to ask, and K represents the *knowledge* needed to answer the questions. It should be noted that if D is empty, QA degenerates into the *traditional reasoning problem*; if ϕ is restricted to a proposition symbol, QA simplifies to both the *query evaluation problem* in databases and the *classification problem* in machine learning.

Following the behaviorist perspective, a notion of *abstract knowledge base* can then be defined as *the class of database-query pairs* (D, ϕ) *such that* ϕ *is inferable from* D *and the underlying knowledge base*. To define this formally, we need to establish what constitutes valid databases and queries.

Databases and Queries. We assume Δ to be a countably infinite set, consisting of all the constants used in databases and queries. Following the tradition in databases, both the closed-world assumption (CWA) and the open-world assumption (OWA) can be made (Abiteboul, Hull, and Vianu 1995). Therefore, each predicate symbol is either an OWA-predicate symbol or a CWA-predicate symbol, but not both.

A *database signature* is a signature $\sigma \supseteq \Delta$ that involves no function symbols of arities greater than 0. A *query signature* is a signature $\upsilon \supseteq \Delta$, containing no CWA-predicate symbol. Given any database signature σ, let $\mathit{Fact}(\sigma)$ denote the set of all σ-atoms that involve no variables and equality.

Definition 1. Let σ be a database signature. A *σ-database* is a partial function $D : \mathit{Fact}(\sigma) \rightharpoonup \{1, 0, -1\}$ such that

1. (finiteness of observation) there are only a finite number of atoms $\alpha \in \mathit{dom}(D)$ such that $D(\alpha) \geq 0$;
2. (completeness of CWA-predicates) D is defined on every atom that involves a CWA-predicate symbol in σ.

Intuitively, in the above definition, by $D(\alpha) = 1$ (respectively, $D(\alpha) = 0$) we mean that α was observed to be true (respectively, false), and by $D(\alpha) = -1$ we mean that α has not been observed yet, but its truth is already determined by the current observation and a fixed set of rules under CWA.

Every *observed fact* of D is an atom $\alpha \in \mathit{dom}(D)$ such that $D(\alpha) \geq 0$. Let $\mathit{DC}(D)$ denote the set of constants each of which appears in at least one observed fact of D. Moreover, D is said to be *positive* if there is no atom $\alpha \in \mathit{dom}(D)$ such that $D(\alpha) = 0$, i.e., no negative fact is allowed in D.

We are interested in the following classes of databases:

1. $\mathscr{D}^{\sigma}_{\mathrm{All}}$: the class of arbitrary σ-databases;
2. $\mathscr{D}^{\sigma}_{\mathrm{Pos}}$: the class of positive σ-databases.

Let $\mathcal{A}$ be a structure of some signature $\upsilon \supseteq \sigma$. We say that $\mathcal{A}$ is a *model* of D, written $\mathcal{A} \models D$, if we have both

1. $\mathcal{A} \models \alpha$ for all $\alpha \in \mathit{dom}(D)$ with $D(\alpha) = 1$, and
2. $\mathcal{A} \not\models \alpha$ for all $\alpha \in \mathit{dom}(D)$ with $D(\alpha) = 0$.

Next, we define what constitutes a query language:

Definition 2. Given a query signature σ, a *query language* of σ is a recursive class $\mathscr{Q}$ of FO-sentences of σ such that

1. $\mathscr{Q}$ is closed under conjunctions, that is, if $\phi, \psi \in \mathscr{Q}$, then $\phi \wedge \psi \in \mathscr{Q}$;
2. $\mathscr{Q}$ is closed under constant renaming, that is, if $\tau : \Delta \to \Delta$ is injective and $\phi \in \mathscr{Q}$, then $\tau(\phi) \in \mathscr{Q}$;
3. $\mathscr{Q}$ contains at least one non-tautological sentence.

The notation $\tau(\phi)$ above denotes the sentence obtained from ϕ by replacing every occurrence of each $c \in \Delta$ with $\tau(c)$.

Example 1. Both Boolean conjunctive queries (CQs) and unions of conjunctive queries (UCQs, i.e., existential positive FO-sentences) are query languages according the above definition, see, e.g., (Abiteboul, Hull, and Vianu 1995).

We believe that employing first-order fragments as query languages is a reasonable assumption for the following reasons. According to Lindström's second theorem, first-order logic is the most expressive semi-decidable logic that admits the Löwenheim-Skolem property (Lindström 1969). It is also worth to mention that most of the results presented in this paper can be generalized to other semi-decidable logics.

Knowledge Bases. To simplify the presentation, in the rest of this paper, we *fix σ_D as a database signature, σ_Q a query signature, $\mathscr{D} \in \{\mathscr{D}^{\sigma_D}_{\mathrm{All}}, \mathscr{D}^{\sigma_D}_{\mathrm{Pos}}\}$, and $\mathscr{Q}$ a query language of σ_Q*. Now, let us present a definition for abstract knowledge bases, following the spirit of abstract OMQA-ontology in (Zhang, Zhang, and You 2016; Zhang and Jiang 2022).

Definition 3. A *knowledge base (KB)* over $(\mathscr{D}, \mathscr{Q})$ is a subclass K of $\mathscr{D} \times \mathscr{Q}$ satisfying all the following properties:

1. (Correctness of tautological queries) If $\phi \in \mathscr{Q}$ is a tautology and $D \in \mathscr{D}$, then $(D, \phi) \in K$;
2. (Closure under query implications) If $(D, \phi) \in K$ and $\psi \in \mathscr{Q}$ and $\phi \models \psi$, then $(D, \psi) \in K$;
3. (Closure under query conjunctions) If $(D, \phi) \in K$ and $(D, \psi) \in K$, then $(D, \phi \wedge \psi) \in K$;
4. (Closure under database extensions) If $(D, \phi) \in K$ and $D_0 \in \mathscr{D}$ extends D, then $(D_0, \phi) \in K$;
5. (Closure under constant renaming) If $(D, \phi) \in K$ and $\tau : \Delta \to \Delta$ is injective, then $(\tau(D), \tau(\phi)) \in K$.

The notation $\tau(\cdot)$ above is the same as that in Definition 2, and it is naturally generalized to databases.

In the above, almost all properties are natural and easy to understand. We only give explanations for Properties 4 and 5. Intuitively, Property 4 states that reasoning about open-world information should be monotone, i.e., adding new observed OWA-facts will not change previous answers. Note that, by the definition of database, all CWA-predicates are information complete so that no CWA-fact can be added to a database (to build an extension), which means that Property 4 cannot be applied to any CWA-predicate.

Property 5 rests on the assumption that knowledge should encapsulate general properties applicable to all objects in the underlying domain, rather than including propositions about specific objects. Consequently, the names of objects should not influence the results of QA.

In Definition 3, only Boolean queries are used, but this is not limiting. By allowing constants in queries, we enable an efficient conversion from arbitrary QA to Boolean QA.

In machine learning, bounded-error algorithms are commonly used. Unfortunately, finding a meaningful method to evaluate error rates for representation-depend tasks like reasoning is extremely difficult, if not impossible (Lynch 1974). This is why our framework does not account for this aspect.

Moreover, an important question arises as to whether the above properties (1-5) indeed capture the class of knowledge bases represented in any formalism in which we are interested. First consider the necessity. In a straightforward way, one can verify it case by case. The following is an example.

Example 2. Suppose σ_D contains no CWA-predicate symbols. Let Σ be a set of sentences (in a monotone logic such as FO or SO) of a signature $\sigma \supseteq (\sigma_D \cup \sigma_Q) \setminus \Delta$. Let

$$K_\Sigma := \{(D, \phi) \in \mathscr{D} \times \mathscr{Q} : D \cup \Sigma \models \phi\}.$$

It is easy to verify that K_Σ is a KB over $(\mathscr{D}, \mathscr{Q})$.

However, as aforementioned, there are a very large number of knowledge representation formalisms to be verified. To avoid this, we address the question from a semantical perspective. Let D be a database recording the current observation in a certain domain. Based on the observation D, the knowledge base will produce a certain belief. The latter can be denoted by a class of worlds (structures) in which the belief holds. We thus have the following definition:

Definition 4. Let $\sigma \supseteq \sigma_D$ be a signature. A *belief mapping* of (σ_D, σ) is a function $\mathbb{M}$ that maps every σ_D-database to a class of σ-structures such that

1. if $\mathcal{A} \in \mathbb{M}(D)$ then $\mathcal{A} \models D$;
2. if $\tau : \Delta \to \Delta$ is injective, and ϕ an FO-sentence of σ, then $\mathbb{M}(D) \models \phi$ iff $\mathbb{M}(\tau(D)) \models \tau(\phi)$;
3. if D_0 is an extension of D and ϕ an FO-sentence of σ such that $\mathbb{M}(D) \models \phi$, then $\mathbb{M}(D_0) \models \phi$

for all σ_D-databases D and D_0.

In the above, the first condition states that the belief produced from the observation must be consistent with the observation; the second asserts that, up to a constant renaming, from the same observation, $\mathbb{M}$ produces the same belief; and the third denotes that adding new observed OWA-facts will not change answers obtained by query answering with $\mathbb{M}$.

Next, we show how circumscription (McCarthy 1980) defines a belief mapping. Some notations are needed.

The language of circumscription is the same as first-order logic, armed with the minimal model semantics. Let $\upsilon \supseteq \sigma_D$ be a signature. Let υ_c be the set of all CWA-predicate symbols in σ_D, and $\upsilon_o := \upsilon \setminus \upsilon_c$. Let $\mathcal{A}$ and $\mathcal{B}$ be υ-structures. We write $\mathcal{A} \subseteq_{\upsilon_c} \mathcal{B}$ if $\mathcal{A}$ and $\mathcal{B}$ share the same domain, and

1. for all $P \in \upsilon_c$, we have $P^{\mathcal{A}} \subseteq P^{\mathcal{B}}$, and
2. $\mathcal{A}|_{\upsilon_o} = \mathcal{B}|_{\upsilon_o}$, i.e., OWA-parts of $\mathcal{A}$ and $\mathcal{B}$ are the same.

Furthermore, let Σ be a set of FO-sentences of υ, and let D be a σ_D-database. We use $Mod^u_m(D, \Sigma, \upsilon_c)$ to denote the class of all UNA-structures of υ that are $\subseteq_{\upsilon_c}$*-minimal models* (i.e., minimal under the order $\subseteq_{\upsilon_c}$) of both D and Σ.

Example 3. Let $\mathbb{M}$ denote the mapping that maps each σ_D-database D to $Mod^u_m(D, \Sigma, \upsilon_c)$. It is easy to see that $\mathbb{M}$ is a belief mapping as it satisfies Conditions 1-3 of Definition 4.

With a belief mapping $\mathbb{M}$, the KB can then be defined:

$$kb(\mathbb{M}, \mathscr{D}, \mathscr{Q}) := \{(D, \phi) \in \mathscr{D} \times \mathscr{Q} : \mathbb{M}(D) \models \phi\}.$$

The following proposition tells us that, despite its excessive inclusiveness, every belief mapping defines a knowledge base satisfying all the properties of Definition 3.

Proposition 1. *Let σ be a signature such that $\sigma_D \cup \sigma_Q \subseteq \sigma$, and $\mathbb{M}$ be a belief mapping of (σ_D, σ). Then $kb(\mathbb{M}, \mathscr{D}, \mathscr{Q})$ is a KB over $(\mathscr{D}, \mathscr{Q})$.*

Now, let us show the sufficiency of Properties 1-5 of Definition 3 to capture the notion of knowledge bases. It suffices to find a logical representation for each KB in Definition 3. By logical representations, we use theories of McCarthy's circumscription under the unique name assumption.

Proposition 2. *Let K be a KB over $(\mathscr{D}, \mathscr{Q})$, and σ be the set consisting of all CWA-predicate symbols in σ_D. Then there are a set Σ of FO-sentences such that, for all $D \in \mathscr{D}$ and $\phi \in \mathscr{Q}$, $Mod^u_m(D, \Sigma, \sigma) \models \phi$ iff $(D, \phi) \in K$.*

Sketched Proof. The main idea involves constructing a rule for each pair $(D, \phi) \in K$ such that its body describes D and its head records ϕ. If the facts in D have been observed, the rule will be triggered to support QA on ϕ. Let Σ be the set of all such rules. We can prove that Σ is the desired set. □

Formalisms. Based on the definition of abstract knowledge bases, we are now able to present a general definition of formalisms for representing knowledge in AI systems.

Definition 5. A *quasi knowledge representation formalism (qKRF)* over $(\mathscr{D}, \mathscr{Q})$ is defined as a mapping Γ such that

1. *dom*(Γ) is recursive subset of $\{0, 1\}^*$, and each string in *dom*(Γ) is called a *theory* of Γ;
2. Γ maps each theory π of Γ to a KB over $(\mathscr{D}, \mathscr{Q})$.

Moreover, a qKRF Γ is a *knowledge representation formalism (KRF)* if it admits an additional property as follows:

3. It is recursively enumerable to check, given $\pi \in$ *dom*(Γ), $D \in \mathscr{D}$ and $\phi \in \mathscr{Q}$, whether $(D, \phi) \in \Gamma(\pi)$ or not.

In the above definition, the three properties establish key requirements for every knowledge representation formalism: Property 1 stipulates that the formalism must possess a language with an effective method for determining whether a given expression is legal; Property 2 defines the semantics of the formalism by associating each legal expression (or theory) in the language with an abstract knowledge base; and Property 3 ensures the implementability of the formalism by requiring that there exists a Turing machine capable of solving the query answering problem for this formalism.

Example 4. Suppose σ_D involves no CWA-predicate symbols. Let $\mathscr{L} \in \{\mathrm{FO}, \mathrm{SO}\}$ and $\sigma \supseteq (\sigma_D \cup \sigma_Q) \setminus \Delta$ be a signature. Let $\Gamma_{\mathscr{L}}$ be a mapping that maps each finite set Σ of σ-sentences in $\mathscr{L}$ to a KB K_Σ defined as follows:

$$K_\Sigma := \{(D, \phi) \in \mathscr{D} \times \mathscr{Q} : D \cup \Sigma \vDash \phi\}.$$

It is easy to verify that both Γ_{FO} and Γ_{SO} are qKRFs over $(\mathscr{D}, \mathscr{Q})$, and the former is a KRF, but the latter is not.

Universal KRFs

In the last section, we have proposed a very general definition for knowledge representation formalisms. Both in theory and in practice, we would like the underlying knowledge representation formalism to be as expressive as possible. In this section, we will study what universal (q)KRFs are, and prove some interesting properties that such (q)KRFs enjoy.

There are at least two natural ways to define the class of universal (q)KRFs. The first is from a perspective on expressive power. We first present a notion defined in this way.

Definition 6. A qKRF Γ over $(\mathscr{D}, \mathscr{Q})$ is said to be *expressively complete* if *ran*(Γ) consists of all the recursively enumerable KBs over $(\mathscr{D}, \mathscr{Q})$.

According to the above definition, given an expressively complete qKRF Γ, for each knowledge base K represented in Γ, there exists a Turing machine to implement query answering on K. However, this does not guarantee that Γ itself qualifies as a KRF. To be a KRF, Γ must have a single Turing machine capable of implementing query answering for all knowledge bases it represents. This leads to a natural question: Is there an expressively complete KRF? We will answer this question in the remainder of this section.

Another natural approach to defining universal KRFs involves reducibility between KRFs, defined as follows:

Definition 7. Let Γ and Γ_0 be KRFs over $(\mathscr{D}, \mathscr{Q})$. Then Γ is *reducible to* Γ_0 if there is a recursive function $p :$ *dom*$(\Gamma) \to$ *dom*(Γ_0) such that $\Gamma = \Gamma_0 \circ p$. In addition, we call p a *reduction* from Γ to Γ_0.

With this notion, universal KRFs can be defined as below:

Definition 8. Let Γ be a KRF over $(\mathscr{D}, \mathscr{Q})$. Then Γ is said to be *universal* if every KRF over $(\mathscr{D}, \mathscr{Q})$ is reducible to Γ.

In other words, a universal KRF is a formalism such that query answering with any KB represented in a KRF can be implemented in the underlying formalism by an effective translation. Note that translating is a rather general approach to implement knowledge reasoning systems, and implementing knowledge reasoning systems by procedural (or other kinds of) programs is in fact a type of translations. These thus demonstrate why the above definition is natural for universal formalisms of knowledge representation.

In order for Definition 8 to make sense, we have to answer the following question: *Is there indeed a universal KRF according to this definition?* At first glance, the answer to this question appears to be obviously affirmative. For example, as we know, every recursively enumerable KB can be accepted by a Turing machine. So, a naive idea is by defining a mapping Γ as follows: If M is Turing machine recognizing some KB K, then let $\Gamma(\llbracket M \rrbracket) := K$. Unfortunately, such a mapping is not possible to be a KRF because $dom(\Gamma)$ is not recursive. Notice that the latter is an immediate corollary of Rice's theorem, see, e.g., (Rogers 1987).

To construct the desired KRF, our general idea is by carefully identifying a recursive subset L of $dom(\Gamma)$ such that the restriction of Γ to L is a KRF. To implement it, we propose an effective transformation to convert every Turing machine M to a Turing machine M^* that accepts a KB. The class of arbitrary Turing machines is clearly recursive. Since the transformation is effective, the class of Turing machines M^*, where M is an arbitrary Turing machine, is thus recursive, too. This then implements the general idea.

Next, we show how to construct the transformation. Some notations are needed. Given a Turing machine M, let

$$\mathbb{K}(M) := \{(D, \phi) \in \mathscr{D} \times \mathscr{Q} : M \text{ accepts } \llbracket D, \phi \rrbracket\}.$$

Given a subclass K of $\mathscr{D} \times \mathscr{Q}$, let $cl(K)$ denote the minimum superclass of K which admits Properties 1-5 of Definition 3. *We need to assure the desired transformation* $(\cdot)^*$ *satisfying the property:* $\mathbb{K}(M^*) = cl(\mathbb{K}(M))$.

The desired machine M^* is constructed from M by implementing Procedure 1. Now, let us explain how M^* works. Let $D \in \mathscr{D}$ and $\phi \in \mathscr{Q}$. Roughly speaking, the computation of M^* on the input $\llbracket D, \phi \rrbracket$ can be divided into six parts:

1. Simulate M on $\llbracket D, \phi \rrbracket$; accept if M accepts.
2. Check whether ϕ is a tautology; accept if true.
3. Check whether there is a sentence $\psi \in \mathscr{Q}$ such that $\psi \models \phi$ and that M^* accepts $\llbracket D, \psi \rrbracket$; accept if true.
4. Check whether there exist a pair of sentences $\psi, \chi \in \mathscr{Q}$ such that $\phi = \psi \wedge \chi$ and that M^* accepts both $\llbracket D, \psi \rrbracket$ and $\llbracket D, \chi \rrbracket$; accept if true.
5. Check whether there is a database $D' \in \mathscr{D}$ such that D extends D' and that M^* accepts $\llbracket D', \phi \rrbracket$; accept if true.
6. Check whether there is a constant renaming (in fact, only need to consider injective functions from C to Δ where C is the set of constants appearing in either $DC(D)$ or ϕ) τ such that M^* accepts $\llbracket \tau(D), \tau(\phi) \rrbracket$; accept if true.

A direct implementation of the above procedure is generally impossible due to the following issues: Firstly, computations on Parts 1-6 may not terminate. Secondly, Part 3 needs to enumerate all sentences ψ in $\mathscr{Q}$, and Part 6 involves generating all possible constant renamings for D, both enumerations are infinite. Thirdly, Parts 3-6 also involve recursive invocations of M^*, which makes the situation even worse.

To address these issues, we introduce a task array T where the i-th task is stored in $T[i]$. Although there could be an infinite number of nonterminating tasks in T, a standard technique can be applied to sequentially simulate multiple such tasks. This can be visualized by a Cartesian coordinate system where the x-axis represents numbers of steps and the y-axis denotes task indices. The simulation is actually a procedure to traverse the first quadrant of this coordinate system, which is implemented by Lines 6-9 of Procedure 1.

An *atomic task* is a tuple $t := (N, p_1, \ldots, p_k)$, where N is a Turing machine and $p_1, \ldots, p_k$ its parameters. Performing t involves simulating N on the input $\llbracket p_1, \ldots, p_k \rrbracket$, with t being *successful* if N accepts. Every *task* is either an atomic task or a finite sequence of atomic tasks $t := \langle t_1, \ldots, t_n \rangle$. Executing t means sequentially performing the atomic tasks $t_1, \ldots, t_n$. The task t is said to be *successful* or to *succeed* if all its constituent atomic tasks are successful.

Moreover, let M_e denote a Turing machine that accepts $\llbracket \phi, \psi \rrbracket$ iff ϕ, ψ are a pair of FO-sentences such that $\phi \models \psi$. Let M_n denote a Turing machine that accepts $\llbracket \tau \rrbracket$ iff τ is a partial injective functions on Δ with a finite domain.

As there might be an infinite number of tasks, we require M^* to perform current tasks and generate new tasks at the same time. For example, in Part 1, we perform the first step of the task (M, D, ϕ) and add new tasks such as $(M_e, \top, \phi)$ to T in the same **for**-loop, see Lines 20-41 of Procedure 1.

Using the task array T, infinite enumerations can then be removed. For instance, to handle the infinite enumeration of queries in $\mathscr{Q}$ in Part 3, we start by letting ψ be the first query (in the order of a natural encoding) in $\mathscr{Q}$, and add the task

$$t := \langle (M_e, \psi, \phi), (M, D, \psi) \rangle$$

to T. When t is eventually executed, we add the task

$$\langle (M_e, \chi, \phi), (M, D, \chi) \rangle$$

to T, where χ is the next query after ψ in $\mathscr{Q}$. By repeating this process, we effectively enumerate all queries in $\mathscr{Q}$. For details, see Lines 22-23, 27-28, and 42-47 of Procedure 1.

Moreover, recursive invocations can be eliminated by utilizing the task array T, too. For instance, to compute M^* on the input $\llbracket D, \psi \rrbracket$ in Part 3, we simply add the task (M, D, ψ) to T. As per Lines 20-42 of Procedure 1, when (M, D, ψ) is initiated, all tasks of computing the closure will be added to T orderly to implement the computation of (M^*, D, ψ).

A flag array F records the completion status of each task as either **true** or **false**. The relationships between tasks are tracked using arrays p (parent) and c (cooperation). Specifically, $p[n] = i$ indicates that the task $T[n]$ is generated from the task $T[i]$, making $T[i]$ the parent of $T[n]$. A value of $c[i] = -1$ signifies that the task $T[i]$ operates independently. If $T[i]$ succeeds, its parent task's flag, $F[p[i]]$, is set to **true**. Otherwise, if $c[i] = j \neq -1$, the task $T[i]$ must cooperate

with the task $T[j]$. Only when both $T[i]$ and $T[j]$ succeed does the parent task's flag, $F[p[i]]$, get set to **true**. For details on truth propagation, refer to Lines 14-16 of Procedure 1.

According to the construction of $(\cdot)^*$, it is not difficult to prove the following proposition.

Lemma 1. $\mathbb{K}(M^*) = cl(\mathbb{K}(M))$ *for every Turing machine* M.

We are now in the position to define the desired KRF.

Definition 9. Let Θ denote the mapping which maps $[\![M^*]\!]$ to $\mathbb{K}(M^*)$ for every Turing machine M.

Does Θ serve as the required KRF? The subsequent theorem confirms this with a positive answer.

Theorem 1. Θ *is a universal KRF over* $(\mathscr{D}, \mathscr{Q})$. *In addition,* Θ *is also expressively complete.*

Proof. We first show that Θ satisfies Conditions 1-3 of Definition 5. Condition 2 immediately follows from Lemma 1. Let $\mathcal{M}$ be the set that consists of $[\![M]\!]$ for all Turing machines M. Clearly, $\mathcal{M}$ is recursive. By the definition of M^*, we can effectively convert each $[\![M]\!] \in \mathcal{M}$ to $[\![M^*]\!]$. Consequently, $dom(\Theta)$ is recursive, and we thus obtain Condition 1. Condition 3 is assured by the fact that there is a universal Turing machine which simulates each M^* on any input $[\![D, \phi]\!]$, and this proves that Θ is indeed a KRF.

Next, let us consider the universality of Θ. Let Γ be an arbitrary KRF over $(\mathscr{D}, \mathscr{Q})$. By definition, it is recursively enumerable to determine, given $\pi \in dom(\Gamma)$, $D \in \mathscr{D}$ and $\phi \in \mathscr{Q}$, whether $(D, \phi) \in \Gamma(\pi)$ or not. Let M be a Turing machine which solves that problem. Clearly, given any theory π of Γ, one can construct a Turing machine M_π that accepts the KB $\Gamma(\pi)$. Let p be a function that maps π to $[\![M_\pi^*]\!]$. It is easy to verify that p is recursive and $\Gamma(\pi) = \Theta(p(\pi))$. Thus Γ is reducible to Θ, which yields the universality.

Now it remains to prove the expressive completeness. Let K be any recursively enumerable KB. There is then a Turing machine M such that, for all $D \in \mathscr{D}$ and $\phi \in \mathscr{Q}$, $(D, \phi) \in K$ iff M accepts $[\![D, \phi]\!]$. By Lemma 1, we thus have

$$K = cl(K) = cl(\mathbb{K}(M)) = \mathbb{K}(M^*) = \Theta([\![M^*]\!]).$$

Since K is arbitrary, Θ must be expressively complete. □

Interestingly, the reduction p from any universal KRF to Θ given in the above proof is computable in linear time. This indicates that, despite Θ being a procedural KRF, *no (declarative) universal KRF can be linearly more succinct than* Θ.

Thanks to the transitivity of reducibility, the following proposition is an immediate corollary of Theorem 1.

Corollary 1. *Every KRF* Γ *over* $(\mathscr{D}, \mathscr{Q})$ *is universal iff* Θ *is reducible to* Γ.

With the above result, we thus call Θ the *canonical KRF*. One might wonder why we do not use a logical (or declarative) KRF as the canonical KRF. The main reasons are as follows: Firstly, proving the recursion theorem for a logical language appears to be challenging. Secondly, and more importantly, while the logical language DED has been shown to be a universal KRF when databases contain only positive OWA-facts and queries are limited to CQs or UCQs (Zhang,

Procedure 1: The Workflow of M^*

Input: $D \in \mathscr{D}$ and $\phi \in \mathscr{Q}$
Output: Accept iff $(D, \phi) \in cl(\mathbb{K}(M))$

1 Initialize all positions in F to **false**;
2 Initialize all positions in c to -1;
3 $T[0] \leftarrow (M, D, \phi)$; /* Set the root task */
4 $p[0] \leftarrow -1$; /* The root has no parent node */
5 $n \leftarrow 1$; /* There is one task in the current array */
6 **for** $k \leftarrow 0$ **to** ∞ **do**
7 **for** $i \leftarrow 0$ **to** $\min(n-1, k)$ **do**
8 /* Simulate multiple tasks in parallel */
9 Perform the $(k-i)$-th step of $T[i]$ if it exists;
10 **if** $T[i]$ *has just succeeded* **then**
11 $F[i] \leftarrow$ **true**;
12 $j \leftarrow i$;
13 /* Propagate truth values upward */
14 **while** $j > 0$ *&* $(c[j] = -1$ *or* $F[c[j]])$ **do**
15 $j \leftarrow p[j]$;
16 $F[j] \leftarrow$ **true**;
17 **if** $F[0]$ */* a support found */* **then** accept;
18 /* Generate tasks */
19 **if** (M, D_0, ψ) *in* $T[i]$ *has just started* **then**
20 $T[n] \leftarrow (M_e, \top, \psi)$; /* Property 1 */
21 $\chi \leftarrow$ the first query in $\mathscr{Q}$; /* Property 2 */
22 $T[n+1] \leftarrow \langle (M_e, \chi, \psi), (M, D_0, \chi) \rangle$;
23 /* Generate tasks for Property 5 */
24 $\tau \leftarrow$ the first constant renaming;
25 $T[n+2] \leftarrow \langle (M_n, \tau), (M, \tau(D_0), \tau(\psi)) \rangle$;
26 $p[n+2] \leftarrow p[n+1] \leftarrow p[n] \leftarrow i$;
27 $n \leftarrow n+3$;
28 /* Generate tasks for Property 3 */
29 **if** $\psi = \chi \wedge \eta$ *&* $\{\chi, \eta\} \subseteq \mathscr{Q}$ **then**
30 $T[n] \leftarrow (M, D_0, \chi)$;
31 $T[n+1] \leftarrow (M, D_0, \eta)$;
32 $c[n] \leftarrow n+1$; /*Coop. with $T[n+1]$ */
33 $c[n+1] \leftarrow n$; /*Coop. with $T[n]$ */
34 $p[n] \leftarrow p[n+1] \leftarrow i$;
35 $n \leftarrow n+2$;
36 /* Generate tasks for Property 4 */
37 **forall** $D_1 \in \mathscr{Q}$ *s.t.* D_0 *extends* D_1 **do**
38 $T[n] \leftarrow (M, D_1, \psi)$;
39 $p[n] \leftarrow i$;
40 $n \leftarrow n+1$;
41 **else if** $T[i] = \langle (M_e, \chi, \psi), (M, D_0, \chi) \rangle$
42 *&* $k - i = 1$ */* $T[i]$ has just started */* **then**
43 $\eta \leftarrow$ the query next to χ in $\mathscr{Q}$;
44 $T[n] \leftarrow \langle (M_e, \eta, \psi), (M, D_0, \eta) \rangle$;
45 $p[n] \leftarrow p[i]$;
46 $n \leftarrow n+1$;
47 **else if** $T[i] = \langle (M_n, \tau), (M, \tau(D_0), \tau(\psi)) \rangle$
48 *&* $k - i = 1$ */* $T[i]$ has just started */* **then**
49 $\tau_1 \leftarrow$ the constant naming next to τ;
50 $T[n] \leftarrow \langle (M_n, \tau_1), (M, \tau_1(D_0), \tau_1(\psi)) \rangle$;
51 $p[n] \leftarrow p[i]$;
52 $n \leftarrow n+1$;

Zhang, and You 2016; Zhang et al. 2020), it remains an open question *whether there exist universal KRFs induced by natural logical languages for more general cases*.

Next, we present some interesting properties enjoyed by Θ, which will play key roles in proving the main theorem. The following is a KRF-version of the padding lemma.

Lemma 2. *For every theory π of Θ, we can effectively find an infinite set S_π of theories of Θ such that $\Theta(\pi) = \Theta(\omega)$ for all theories $\omega \in S_\pi$.*

The recursion theorem can also be generalized to KRFs.

Theorem 2. *Given any Turing machine that computes some recursive function $p : dom(\Theta) \to dom(\Theta)$, we can effectively find a theory π of Θ such that $\Theta(p(\pi)) = \Theta(\pi)$.*

Proof. Let N_Θ be a Turing machine that accepts $[\![\pi, D, \phi]\!]$ iff $(D, \phi) \in \Theta(\pi)$ for all $\pi \in dom(\Theta)$, $D \in \mathscr{D}$ and $\phi \in \mathscr{Q}$. Let $\mathcal{M}$ be a set consisting of $[\![M]\!]$ for all Turing machines M. Take $\kappa \in \mathcal{M}$ arbitrarily. We use φ_κ to denote the function computed by the Turing machine encoded by κ. Now let us construct a Turing machine M_κ which works as follows:

> Given any input $[\![D, \phi]\!]$, first try to simulate the computation of φ_κ on κ. If it halts with an output ω, then simulate the computation of N_Θ on $[\![\omega, D, \phi]\!]$.

Let M^*_κ be the Turing machine obtained from M_κ by implementing Procedure 1. Thus, we have

$$\Theta([\![M^*_\kappa]\!]) = \begin{cases} \Theta(\varphi_\kappa(\kappa)) & \text{if } \varphi_\kappa(\kappa) \text{ is defined;} \\ cl(\emptyset) & \text{otherwise.} \end{cases}$$

Let q be a mapping that maps κ to $[\![M^*_\kappa]\!]$. Clearly, q is recursive, which implies that $p \circ q$ is a recursive function from $\mathcal{M}$ to $dom(\Theta)$. Let M be a Turing machine that computes $p \circ q$, and let $\upsilon = [\![M]\!]$. It is easy to see that υ can be effectively obtained from p. Since $\varphi_\upsilon = p \circ q$ is recursive, we know that $\varphi_\upsilon(\upsilon)$ is defined. It is easy to verify that

$$\Theta(q(\upsilon)) = \Theta([\![M^*_\upsilon]\!]) = \Theta(\varphi_\upsilon(\upsilon)) = \Theta(p(q(\upsilon))).$$

Let $\pi = q(\upsilon)$. Clearly, $\Theta(\pi) = \Theta(p(\pi))$, and π can be effectively obtained from υ. These compete the proof. □

To present the main theorem, some notions are needed.

Definition 10. Let Γ and Γ_0 be KRFs over $(\mathscr{D}, \mathscr{Q})$, We say Γ and Γ_0 are *recursively isomorphic* if there is a recursive bijection $p : dom(\Gamma) \to dom(\Gamma_0)$ such that $\Gamma = \Gamma_0 \circ p$.

We are now in the position to establish the main theorem.

Theorem 3. *All universal KRFs over $(\mathscr{D}, \mathscr{Q})$ are recursively isomorphic.*

Sketched Proof. It suffices to show that every universal KRF Γ is recursively isomorphic to Θ. This is achieved by adopting a method used in the proof of Rogers's isomorphism theorem (Rogers 1958). The main challenge is to find canonical universal KRFs, which we have resolved in Theorem 1. The rest of the proof proceeds as follows: We find an injective reduction p from Γ to Θ via Lemma 2, and an injective reduction q from Θ to Γ via Theorem 2. With p and q, we construct the desired recursive isomorphism from Γ to Θ. □

Subrecursive KRFs

In the last section, we focused on universal KRFs. However, KRFs with low complexity, called *subrecursive KRFs*, might be useful in practice. A natural question thus arises as to under what condition subrecursive KRFs are recursively isomorphic. We first present a candidate one as follows.

Definition 11. We say a KRF Γ *admits the padding property* if for each theory π of Γ, we can effectively find an infinite set $S_\pi \subseteq dom(\Gamma)$ such that $\Gamma(\pi) = \Gamma(\omega)$ for all $\omega \in S_\pi$.

By Lemma 2, the canonical KRF Θ admits the padding property. Actually, the property holds for almost all the natural expressive formalisms. The following is an example.

Example 5. Every FO-sentence ϕ is logical equivalent to $\phi \wedge \phi$. The defined KBs are thus the same, which provides a way to effectively find theories specified to the same KB. Thus, Γ_{FO} (see Example 4) admits the padding property.

Clearly, all recursively isomorphic KRFs should have the same expressive power. We adopt a slightly stronger condition that requires the KRFs here to be intertranslatable.

Definition 12. A pair of KRFs over $(\mathscr{D}, \mathscr{Q})$, Γ and Γ_0, are *equally strong* if Γ is reducible to Γ_0 and vice versa.

The main result for subrecursive KRFs is as follows. The proof mirrors that of Theorem 3, with the only difference being the use of the padding property instead of Theorem 2.

Theorem 4. *All equally strong KRFs over $(\mathscr{D}, \mathscr{Q})$ that admit the padding property are recursively isomorphic.*

Conclusions and Related Work

A general framework for studying KRFs has been proposed. Within the framework, all universal KRFs (respectively, all pairwise intertranslatable KRFs that admit the padding property) have been proven to be recursively isomorphic.

Regarding the "declarative vs. procedural controversy", our findings indicate that equally expressive natural KRFs exhibit similar performance such as reasoning efficiency. Moreover, no declarative KRF is linearly more succinct than the procedural KRF Θ. Thus, labeling a KRF as declarative or procedural becomes less meaningful. Instead, declarativeness is better understood as a characteristic of specific representations rather than of the formalisms themselves.

For the debate between symbolic AI and connectionist AI, the existence of recursive isomorphisms between KRFs implies that for any knowledge operator (e.g., gradient descent) in a KRF we can effectively find an operator in another KRF to perform the same transformation. From a theoretical perspective, all these representation methodologies either pave the way to AGI or none, with core challenges being universal and advancements in one methodology benefiting others.

A closely related work is Rogers' isomorphism theorem, which states that all Gödel numberings are recursively isomorphic (Rogers 1958). This theorem plays a crucial role in computability theory. The proof of Theorem 3 involves an argument similar to that in (Rogers 1958), but the challenges we encounter are quite different. While the existence of Gödel numberings is readily apparent, establishing the existence of universal KRFs presents significant difficulties.

Acknowledgements

We would like to thank Professor Fangzhen Lin and anonymous referees for their helpful comments and suggestions. This work was supported by the Leading Innovation and Entrepreneurship Team of Zhejiang Province of China (Grant No. 2023R01008) and the Key Research and Development Program of Zhejiang, China (Grant No. 2024SSYS0012).

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Baader, F.; Horrocks, I.; Lutz, C.; and Sattler, U. 2017. *An Introduction to Description Logic*. Cambridge University Press.

Calvanese, D.; Giacomo, G. D.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007. Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. *J. Autom. Reason.*, 39(3): 385–429.

Delgrande, J. P.; Glimm, B.; Meyer, T.; Truszczynski, M.; and Wolter, F. 2024. Current and Future Challenges in Knowledge Representation and Reasoning (Dagstuhl Perspectives Workshop 22282). *Dagstuhl Manifestos*, 10(1): 1–61.

Fagin, R.; Kolaitis, P. G.; Miller, R. J.; and Popa, L. 2005. Data Exchange: Semantics and Query Answering. *Theor. Comput. Sci.*, 336(1): 89–124.

Lindström, P. 1969. On Extensions of Elementary Logic. *Theoria*, 35(1): 1–11.

Lynch, N. A. 1974. Approximations to the Halting Problem. *J. Comput. Syst. Sci.*, 9(2): 143–150.

McCarthy, J. 1980. Circumscription - A Form of Non-Monotonic Reasoning. *Artif. Intell.*, 13(1-2): 27–39.

McCarthy, J.; and Hayes, P. 1981. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In Webber, B. L.; and Nilsson, N. J., eds., *Readings in Artificial Intelligence*, 431–450. Morgan Kaufmann.

Newell, A.; and Simon, H. A. 1976. Computer Science as Empirical Inquiry: Symbols and Search. *Commun. ACM*, 19(3): 113–126.

Pearl, J. 1985. Bayesian Networks: A Model of Self-activated Memory for Evidential Reasoning. In *Proceedings of the 7th conference of the Cognitive Science Society*.

Reiter, R. 1980. A Logic for Default Reasoning. *Artif. Intell.*, 13(1-2): 81–132.

Rogers, H. 1958. Gödel Numberings of Partial Recursive Functions. *J. Symb. Log.*, 23(3): 331–341.

Rogers, H. 1987. *Theory of Recursive Functions and Effective Computability*. MIT Press.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning Representations by Back-propagating Errors. *Nature*, 323.6088: 533–536.

Smith, B. C. 1982. *Procedural Reflection in Programming Languages*. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Sowa, J. F., ed. 1991. *Principles of Semantic Networks - Explorations in the Representation of Knowledge*. Morgan Kaufmann.

van Emden, M. H.; and Kowalski, R. A. 1976. The Semantics of Predicate Logic as a Programming Language. *J. ACM*, 23(4): 733–742.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All You Need. In *NeurIPS 2017*, 5998–6008.

Zhang, H.; and Jiang, G. 2022. Characterizing the Program Expressive Power of Existential Rule Languages. In *AAAI 2022*, 5950–5957.

Zhang, H.; Zhang, Y.; and You, J. 2016. Expressive Completeness of Existential Rule Languages for Ontology-Based Query Answering. In *IJCAI 2016*, 1330–1337.

Zhang, H.; Zhang, Y.; You, J.; Feng, Z.; and Jiang, G. 2020. Towards Universal Languages for Tractable Ontology Mediated Query Answering. In *AAAI-2020*, 3049–3056.