Bulletin T. CLVII de l'Académie serbe des sciences et des arts – 2024
Classe des Sciences mathématiques et naturelles
Sciences mathématiques, № 49

DISTRIBUTED NON-AXIOMATIC REASONING SYSTEM

DEJAN MITROVIĆ*, MIRJANA IVANOVIĆ, PEI WANG

Dedicated to the 100th anniversary of the birth of Academician Bogoljub Stanković

(Accepted at the 9th Meeting, held on December 20, 2024)

A b s t r a c t. This paper presents an architecture and one concrete realization of a novel reasoning system named Distributed Non-Axiomating Reasoning System. The system uses the so-called Non-Axiomatic Logic, a formalism in the domain of artificial general intelligence designed for practical realizations of systems that work under the assumption of insufficient knowledge and resources. The main novelty of the proposed architecture is in the layered and distributed organization of its backend knowledge base. That is, the knowledge base is designed with scalability and fault-tolerance in mind. It allows the system to reason over very large knowledge bases with real-time responsiveness, while serving high numbers of concurrent users. Finally, one concrete practical application of the developed system is presented as well.

AMS Mathematics Subject Classification (2020): 03B42, 03B60, 68M14, 68T42

Key Words: reasoning architecture, non-axiomatic logic, artificial general intelligence, big data.

*This author defended his PhD at the University of Novi Sad (2015), under supervision of the second author.

1. *Introduction*

The term *Big Data* is used to describe large quantities of highly diverse information, often collected at high frequencies, which traditional approaches cannot efficiently process and analyze [1]. In recent years, the need for Big Data analytics has found its place in a wide range of industrial applications (e.g. [2]), and is often seen as a crucial asset in the world that produces an ever increasing amounts of information. This need has, in turn, resulted in a number of concrete, practical technologies, including *NoSQL* and graph databases (e.g. [3]), the *MapReduce* programming model [4], etc.

For an artificial intelligence system, the ability to efficiently process large amounts of knowledge is one of the key requirements [5]. This paper presents a novel general-purpose reasoning architecture named *Distributed Non-Axiomatic Reasoning System* (DNARS) which fullfills this requirement by relying on a number of Big Data concepts.

For example, DNARS includes a distributed, highly-scalable backend knowledge base, which also features fault-tolerance through data replication. Thus, the main advantage of DNARS, when compared to all other existing reasoning and cognitive architectures, is that it leverages state-of-the-art techniques for large-scale distributed data management and processing. This approach allows our system to operate on top of very large knowledge bases, while serving large numbers of external clients with real-time responsiveness.

The overall architecture of DNARS consists of two main components. Its *backend knowledge base* is used to store the system's knowledge and experience on a large scale. In order to work with these kinds of knowledge bases, DNARS includes a set of efficient algorithms in form of *inference engines*. As the endresult, for example, DNARS can provide thousands of answers per second from a knowledge base of over 75 million statements.

For its reasoning capabilities, DNARS relies on the *Non-Axiomatic Logic* (NAL). NAL represents a formalism for reasoning systems in the domain of Artificial General Intelligence (AGI) [6, 7]. NAL is used as the underlying formalism in DNARS because its "philosophy" fits nicely into the DNARS' overall goals. That is, the term *non-axiomatic* in NAL indicates that the logic is constructed around the notion of *insufficient knowledge and resources* [6]. In fact, this notion is one of defining characteristics of NAL, and it encompasses several important concepts.

First of all, knowledge is uncertain, and not necessarily consistent. New evidence can be accepted at any time, it can include any content, and can affect the truth-value of any existing statement. But, a truth-value is not expected to converge to any limit. At the same time, the system usually does not have enough resources, in terms of space and time, to consult its entire knowledge base when solving a

problem. It cannot apply the full set of inference rules, nor follow a predefined algorithm. Finally, the problem-solving process is localized, in the sense that only a fraction of statements is used to reach the conclusion.

NAL includes built-in mechanisms for dealing with the aforementioned issues. It can efficiently manage uncertainty and statement inconsistencies, and summarize existing knowledge in order to reduce the sheer amount of statements. Besides providing sound theoretical basis for logical reasoning, NAL is, therefore, highly practical, and can be efficiently realized using the present-day technology.

The rest of the paper is organized as follows. Section 2 provides a brief introduction into NAL, necessary for a better understanding of DNARS. Section 3 presents a detailed overview of the overall architecture of DNARS and its concrete realization. Section 4 presents DNARS performance evaluation in order to validate the statements about its runtime efficiency. One concrete practical application of the proposed system is given in Section 5. Section 6 discusses the related work, while Section 7 discusses the overall conclusions and possible future research directions.

2. *Non-axiomatic logic*

Non-Axiomatic Logic (NAL) includes a symbolic grammar, a set of inference rules, and a semantic theory. However, it is different from many other formalisms used to define reasoning in intelligent systems, in the sense that it is a *term* logic [6, 7]. NAL sentences are given in the form of *subject-copula-predicate*, where *subject* and *predicate* are terms.

The basic and most common type of a statement is *inheritance* in the form of $S \rightarrow P$, where S and P are terms denoting the subject and object, respectively, and $\rightarrow$ denotes the inheritance copula. The inheritance statement $S \rightarrow P$ can be read as S *is a type of* P, for example: *cat is a type of animal*. Subject and predicate are *atomic* or *compound* terms[1]. An atomic term is a word consisting of characters from a finite alphabet. Compound terms are build by connecting atomic or compound terms.

NAL has an *experience-grounded semantic* [8], which is based on the concepts of *specializations* and *generalizations*. In an inheritance statement $S \rightarrow P$, S is said to be the specialization of P, while P is said to be the *generalization* of S [6].

Let V_K be the set of all terms appearing in system's experience K, which is a set of inheritance statements. The *extension* T^E and *intension* T^I of a term $T \in V_K$ can be defined using specializations and generalizations as follows [9, 6]:

$$T^E = \{x \mid (x \in V_K) \wedge (x \rightarrow T)\}, \tag{1}$$

[1] In higher NAL layers, subject and object themselves can also be statements.

$$T^I = \{x \mid (T \in V_K) \wedge (T \rightarrow x)\} \tag{2}$$

The evidence for a term T (or, its *meaning*) consists of both T^E and T^I. That is, the meaning of a term is defined through its relations with other terms. The term has a meaning for the system only if it appears in its experience; otherwise, it is meaningless and has no interpretation.

For a statement $S \rightarrow P$, *positive* and *negative* evidence, denoted as E^+ and E^-, respectively, are defined as follows [9, 6]:

$$E^+ = \{S^E \cap P^E\} \cup \{P^I \cap S^I\}, \tag{3}$$

$$E^- = \{S^E \setminus P^E\} \cup \{P^I \setminus S^I\}. \tag{4}$$

The amount of positive evidence w^+ represents the cardinality of E^+, while the amount of negative evidence w^- represents the cardinality of E^-. The amount of total evidence w is calculated as $w^+ + w^-$.

Positive and negative evidence is used to determine the *truth-value* of a NAL statement. The truth-value of a NAL statement is represented by a pair of real numbers in $[0, 1]$, named *frequency* (f) and *confidence* (c) [6]. Frequency is the ratio of positive and total evidence, while confidence describes how stable this frequency will be when the system gains new evidence [6]:

$$f = \frac{w^+}{w}, \tag{5}$$

$$c = \frac{w}{w + k}. \tag{6}$$

Here, k is the *evidential horizon*, a constant used to prevent the system from comparing possibly infinite future to the relatively short past.

The formal reasoning in NAL is based on *inference rules* that are used to derive new knowledge, to provide answers to questions, or to deal with statement inconsistencies, referred to as *forward, backward*, and *local*, respectively [6]. Most of the rules take the *syllogistic* form. A syllogistic inference rule takes two premises that share a term, and then derives a conclusion consisting of the other two terms [6]. Depending on the copulas and positions of the shared term in premises, different inference rules can be applied.

NAL itself is organized into 9 layers. Each layer builds on top of the previous one, by introducing new concepts, grammar, and/or inference rules. For the purpose of this paper and the realization of DNARS, the first four layers are of special importance. *NAL-1* introduces inference rules on inheritance statements. Among others, there are *deduction*, *induction*, and *abduction* [6].

NAL-2 extends the grammar with the similarity copula, defined as a symmetric inheritance: $(S \leftrightarrow P) \Leftrightarrow (S \rightarrow P) \wedge (P \rightarrow S)$. This new copula results in three new forward inference rules: *comparison, analogy*, and *resemblance* [6, 7, 10].

Compound terms are added in *NAL-3*. Their general form is as follows[2]:

$$\{T_1 \; con \; T_2 \; con \; \ldots \; con \; T_n\}.$$

Here, *con* is the connector and $T_1, \ldots, T_n$ are terms, $n \geq 1$ [6, 7]. Four connector types are recognized: *extensional intersection* $(\cap)$, *intensional intersection* $(\cup)$, *extensional difference* $(-)$, and *intensional difference* $(\ominus)$. Finally, *composition* is introduced to exploit these four connectors. The rule is used to build new compound terms, and, by doing so, to summarize the system's experience.

Finally, *NAL-4* introduces arbitrary relations among terms. An additional connector, *product* $(\times)$, is first introduced. It defines inheritance among individual components of a compound term [6, 7]:

$$((S_1 \times \cdots \times S_n) \rightarrow (P_1 \times \cdots \times P_n)) \Leftrightarrow ((S_1 \rightarrow P_1) \wedge \cdots \wedge (S_n \rightarrow P_n)).$$

A *relational term* R is defined as a term related to a product term by inheritance: $(T_1 \times T_2) \rightarrow R$ or $R \rightarrow (T_1 \times T_2)$ [6, 7].

As noted earlier, this section presents a very brief overview of NAL; only the basic theory necessary for understanding DNARS is given. More information is available in the references used throught this section.

3. *DNARS*

The general architecture of DNARS is outlined in Fig. 1. The main components of the system are:

- *Resolution engine*: answers client's questions.
- *Forward inference engine*: derives new knowledge.
- *Short-term memory*: contains statements relevant to the active processing cycles, and problems that need to be solved.
- *Knowledge domain*: a sub-set of the overall knowledge base, containing mutually dependent or related statements.
- *Backend knowledge base*: the system's overall knowledge base, representing its entire experience.
- *Event manager*: a handler for events generated by changes in (parts of) the knowledge base.

[2]Infix notation can be used as well: $\{con \; T_1 \; T_2 \; \ldots \; T_n\}$.

These components are broadly organized into two categories. Resolution and Forward inference engines are referred to as *DNARS Inference engines*, while the remaining components are described as the *Backend knowledge base*. Each external client is associated with one set of inference engines, but there is only a single Backend knowledge base for all of them. However, the knowledge base is designed to be highly scalable, and can be partitioned to support multiple isolated and cooperating clients.

3.1. DNARS inference engines

Resolution and *Forward inference engines* are used to, respectively, answer questions and derive new knowledge. Therefore, they represent the core of DNARS' inference capabilities. Two types of questions are supported:

- Questions containing "?", i.e., "S *copula* ?" or "? *copula* P". In this case, the Resolution engine inspects the system's knowledge base and finds the best substitute for "?".

- Questions in the form of "S *copula* P". The answer to this question is a statement "S *copula* $P\langle f, c\rangle$". If the answer is not directly available in the system's knowledge base, the engine will try to derive it using backward inference rules.

In general, the Resolution engine should provide answers in real time for the first type of questions. For the second type, the answer is given in real time only if it is directly available in the system's knowledge base. Otherwise, the backward inference process is started in the background and the client is notified of the solution later on.

The Forward inference engine provides direct implementation of forward inference rules defined by NAL. It operates in *inference cycles*, as follows. The execution of each cycle is triggered either by an external client or by an internal process. In case of the former, input statements provided by an external client are used to load all *relevant* statements from the system's knowledge base. Relevant statements for the input S *copula* $P\langle f, c\rangle$ are statements that have S or P as subject or predicate. The unified set of input and relevant statements serves as a starting point for forward inference rules, which may produce another set of conclusions. As the final step of the inference cycle, input, relevant statement and conclusions are merged together, any inconsistencies are resolved, and the final output is stored back in the system's knowledge base.

However, the Forward inference engine can also operate on its own. When idle and not receiving any new input from external clients, the engine will select a (random) statement from the knowledge base and use it as the input. As noted

by [6], this continuous internally-triggered inference is one of the main differences between inference engines and advanced knowledge retrieval systems.

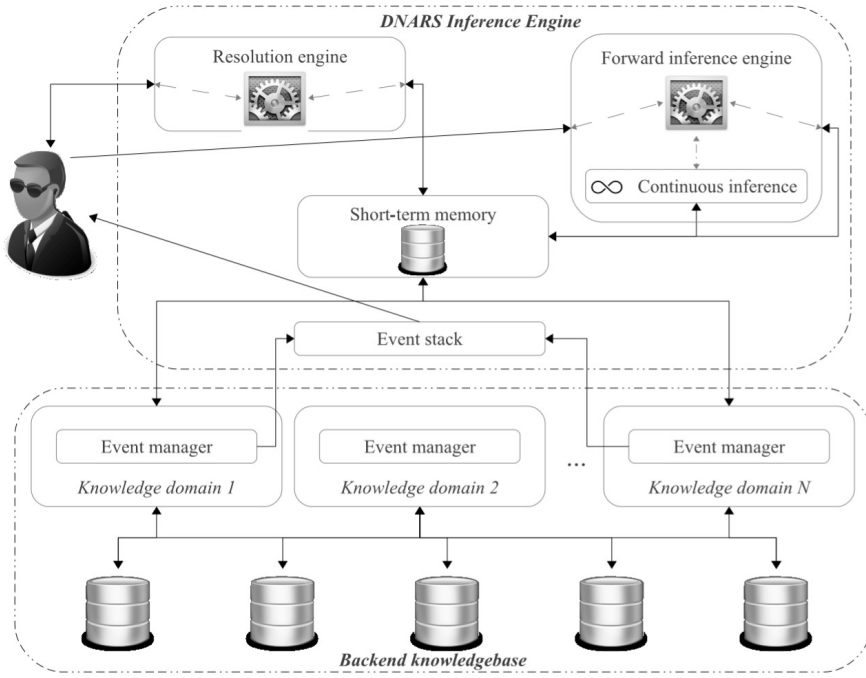

Figure 1: The general architecture of the proposed Distributed Non-Axiomatic Reasoning System and the organization of knowledge

3.2. Backend knowledge base overview

One of the main design goals for DNARS is to develop a system that can efficiently handle large quantities of knowledge[3]. Therefore, the Backend knowledge base of DNARS is designed as a distributed, scalable architecture that consists of three layers. At the bottom-most layer, the entire knowledge base is physically partitioned and distributed across a number of machines. It uses *horizontal scaling*: as the amount of data increases, the runtime performance is maintained by simply adding more processing nodes to the underlying cluster [11]. This design approach has two main benefits:

[3]In today's terms, *large quantities of knowledge* refer to zettabytes, yottabytes, . . ., even quettabytes of data.

- It enables the backend storage to manage large amounts of data. By introducing proper data distribution rules, faster lookups and retrievals of relevant statements can be achieved.

- It provides fault-tolerant features, as the data is replicated across cluster nodes. There is no single point of failure, and the knowledge base can remain intact in case of hardware and software failures.

On top of this layer, the entire knowledge base is organized into one or more *Knowledge domains*. Domains partition the system's knowledge base into distinct categories. This enables the system to work with and focus on a subset of its knowledge. At runtime, one or more domains can be consulted. This organization also supports the multi-client nature of DNARS. The knowledge belonging to one external client can be stored in a separate domain. However, many clients can also work with same domains (one or more), and, by doing so, exhibit cooperated behavior through knowledge and experience sharing. As shown in Fig. 1, the content of one domain can be physically distributed across many machines of the bottom layer, and can be intermingled with the content of other domains. The task of properly storing and retrieving the domain data is delegated to the bottom layer.

Finally, the *Short-term memory* (STM) module is placed at the top layer. This is the knowledge base directly available to DNARS inference engines, and represents the basis for their inference cycles. The main purpose of STM is to serve as the optimization module. Its content should entirely fit in the runtime memory of the host machine, acting as a fast in-memory storage. Once a set of related inference cycles is completed, the STM content is merged back into the corresponding (again, one or more) domains.

It is important for clients, especially in cooperative domain sharing mode, to learn about changes in the knowledge base. For example, the client may wish to perform certain actions in response to newly derived conclusions. To support this feature, DNARS incorporates the *Event manager* module. A change in a knowledge domain will generate one or more events, which will be collected by the manager and delivered to clients connected to the domain.

Knowledge bases containing NAL statements can be represented by *property graphs*. A property graph is a directed, multi-relational graph with any number of properties attached to vertices and edges [12, 3]. That is, it is a graph which can include different types of edges (e.g. for representing inheritance and similarity), and in which each vertex or an edge can have any number of *key* $\rightarrow$ *value* pairs attached to it. Fig. 2 shows an example of a set of NAL statements and the corresponding property graph.

In relatively recent times, large-scale analysis and processing of (property) graphs

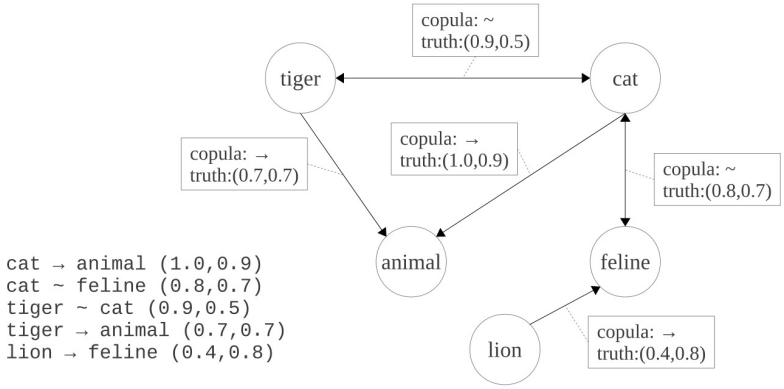

Figure 2: A set of arbitrary NAL statements and the corresponding property graph. Note that edges representing similarities are bidirectional, expressing the symmetric nature of the copula.

has become especially important, due to increasing demands of modern web applications such as social networks. Therefore, there exists a range of algorithms and technologies for efficient graph analytics. These solutions provide a strong basis for an efficient realization of the proposed DNARS architecture. One notable open-source implementation is *Aurelius Titan*[4].

Titan stores each vertex of a graph as a separate row in the database. Vertex identifier (i.e. a hash) represents the row-key, while individual columns hold vertex properties and edges (along with their properties). This means that the edge is stored twice in the database – once for each vertex. However, this approach increases the system's runtime performance. During the process of loading relevant statements into the Short-Term Memory (STM), DNARS selects vertices representing subjects and/or predicates of input statements and loads them along with their corresponding adjacency lists. Therefore, the actual STM content is graph vertices and their adjacency lists.

Titan actually represents a layer on top of "regular" non-graph-oriented databases, both relational and NoSQL. One such database is *Apache Cassandra*[5]. When under heavy loads, Cassandra will sacrifice data consistency in order to provide high-availability and partition-tolerance [13]. This possible lack of consistency in Cassandra is more in line with the NAL's assumption of insufficient knowledge and resources. In addition, Cassandra provides lower latency in random read operations [14], which is essential for real-time responsiveness of the Resolution engine. Due

[4] `https://github.com/thinkaurelius/titan`, retrieved on December 15, 2024.
[5] `http://cassandra.apache.org/`, retrieved on December 15, 2024.

to these reasons, DNARS Backend knowledge base is based on Titan over Cassandra.

Once the correct backend system has been chosen and configured, very little needs to be done in order to satisfy functional requirements of the Backend knowledge base. All functionalities and possible issues are efficiently handled by the Titan-Cassandra combination, allowing the development focus to be placed on realizing the remaining parts of DNARS.

3.3. Forward inference engine

The Forward inference engine in DNARS implements a subset of forward inference rules of the first four NAL layers. This approach is suffcient for developing a first version of the system with practical reasoning abilities (demonstrated in Section 5).

Table 1 summarizes the syllogistic forward inference rules from the first four layers that are directly supported in DNARS. All supported rules are organized into groups and each group is realized as a separate function. For example, in one group, the first premise is $Prem_1 : P \rightarrow M\langle f_1, c_1\rangle$, while the second premise is either $Prem_2 : S \rightarrow M\langle f_2, c_2\rangle$ or $Prem_3 : S \leftrightarrow M\langle f_2, c_2\rangle$. In this case, abduction, comparison and analogy can be applied as follows:

$$\{Prem_1, Prem_2\} \vdash \{S \rightarrow P\langle F_{abd}\rangle, S \leftrightarrow P\langle F_{cmp}\rangle\} \tag{7}$$

$$\{Prem_1, Prem_3\} \vdash P \rightarrow S\langle F_{ana}\rangle. \tag{8}$$

Table 1: Summary of syllogistic forward inference rules currently supported in DNARS. Each rule accepts two premises and derives a conclusion: $\{P_1\langle f_1, c_1\rangle, P_2\langle f_2, c_2\rangle\} \vdash C\langle f, c\rangle$. The top-most row contains P_2 statements, while the left-most column contains P_1 statements [6, 7].

	$S \rightarrow M\langle f_2, c_2\rangle$	$S \leftrightarrow M\langle f_2, c_2\rangle$	$M \rightarrow S\langle f_2, c_2\rangle$
$M \rightarrow P\langle f_1, c_1\rangle$	$S \rightarrow P\langle F_{ded}\rangle$	$S \rightarrow P\langle F_{ana}\rangle$	$S \rightarrow P\langle F_{ind}\rangle$ $S \leftrightarrow P\langle F_{cmp}\rangle$
$M \leftrightarrow P\langle f_1, c_1\rangle$	$S \rightarrow P\langle F_{ana'}\rangle$	$S \leftrightarrow P\langle F_{res}\rangle$	$P \rightarrow S\langle F_{ana'}\rangle$
$P \rightarrow M\langle f_1, c_1\rangle$	$S \rightarrow P\langle F_{abd}\rangle$ $S \leftrightarrow P\langle F_{cmp}\rangle$	$P \rightarrow S\langle F_{ana}\rangle$	

In the given table $\langle F_{ana}\rangle$ denotes a function for calculating the truth-value of the conclusion derived using the analogy rule [6, 7]. The use of ' in $\langle F_{ana'}\rangle$ indicates that the truth-values of the premises have been switched [10]. The same can be done for other inference rules, e.g. $\{P \rightarrow M, M \rightarrow S\} \vdash P \rightarrow S\langle F_{ded'}\rangle$ [6] but this feature is currently unsupported in DNARS for practical reasons.

Listing 1 shows how these three forward inference rules have been realized in the graph-based implementation of DNARS. The execution sequence of this function can be summarized as follows:

- The expression `graph.getV(judgment.pred)` selects the vertex that corresponds to the input judgment's predicate (denoted here as `m`), since this is the shared term in the given three syllogistic rules.

- The expression `m.inE(Inherit)` loads all incoming edges for the shared term. Each edge, along with its source and target vertices, represents the existing statement to be used as the first premise. Therefore, a helper function `inferForEdge` is called for each edge.

- The helper function first retrieves the source vertex for the given edge (denoted here as `p`). That is, in the function `inferForEdge` the entire first premise $P \rightarrow M\langle f_1, c_1\rangle$ is retrieved.

- Finally, based on the input judgment's copula, the new conclusions are derived through abduction and comparison, or through analogy. To avoid generating grammatically incorrect statements, the validity of each conclusion is also checked.

Listing 1: A function that accepts either $S \rightarrow M\langle f_2, c_2\rangle$ or $S \leftrightarrow M\langle f_2, c_2\rangle$ as the second premise (judgment), uses an existing statement $P \rightarrow M\langle f_1, c_1\rangle$ as the first premise, and produces conclusions shown in Eq. 7 and Eq 8.

```
def abductionComparisonAnalogy(judgment: Statement): List[Statement] =
  graph.getV(judgment.pred) match {
    case Some(m) => // m is the shared term
      val incomingEdges = m.inE(Inherit).toList
      incomingEdges.flatMap { e: Edge => inferForEdge(judgment, e) }
    case None =>
      List()
  }

def inferForEdge(judgment: Statement, e: Edge): List[Statement] = {
  val p = e.getVertex(Direction.OUT).term
  if (judgment.subj == p) {
    List() // avoid tautologies
  } else if (judgment.copula == Inherit) {
    abduction(p, judgment, e) ::: comparison(p, judgment, e)
  } else {
    analogy(p, judgment, e)
  }
}
```

```
def abduction(p: Term, judg: Statement, e: Edge): List[Statement] = {
  val truth = e.truth.abduction(judg.truth)
  val derived = Statement(judg.subj, Inherit, p, truth)
  keepIfValid(derived)
} // ... simlarly for comparison and analogy
```

All remaining forward inference rules are realized in a similar pattern. As a concrete example of the forward inference process, Listing 2 shows a starting knowledge base comprised of three inheritance and two similarity statements. The graph representation of this knowledge base is shown in Fig. 3(a).

Listing 2: Starting knowledge base of the forward inference example.

```
tiger ↔ cat ⟨0.9, 0.5⟩
tiger → animal ⟨0.7, 0.7⟩
cat → animal ⟨1.0, 0.9⟩
cat ↔ feline ⟨0.8, 0.7⟩
lion → feline ⟨0.4, 0.8⟩
```

The forward inference process starts as the system receives a new input judgment, $cat \rightarrow mammal\langle 1.0, 0.9\rangle$ and adds it to the knowledge base (Fig. 3(b)). In this scenario, three forward inference rules are applicable: induction, extensional comparison, and analogy. They are defined respectively as follows [6, 7]:

$$\{M \rightarrow P\langle f_1, c_1\rangle, M \rightarrow S\langle f_2, c_2\rangle\} \vdash S \rightarrow P\langle f, c\rangle \tag{9}$$

$$\{M \rightarrow P\langle f_1, c_1\rangle, M \rightarrow S\langle f_2, c_2\rangle\} \vdash S \leftrightarrow P\langle f, c\rangle \tag{10}$$

$$\{M \leftrightarrow P\langle f_1, c_1\rangle, M \rightarrow S\langle f_2, c_2\rangle\} \vdash P \rightarrow S\langle f', c'\rangle \tag{11}$$

Since induction and extensional comparison take very similar premises and produce similar conclusions, they belong the same group and are applied in parallel. When applying forward inference rules, DNARS always takes the first premise from the knowledge base, while the new input represents the second premise. In case of induction and extensional comparison, the first execution step determines that the shared term `m` is *cat*. In the second step, the system selects cat's outgoing edges, along with their respective incoming vertices; more concretely, it determines that statements 3 and 4 in Listing 2 should be used in place of the first premise. The two inference rules can now be applied, deriving that *mammal is a type of animal* ($mammal \rightarrow animal\ \langle 1.00, 0.45\rangle$) and that *mammal is animal* ($mammal \leftrightarrow animal\langle 1.00, 0.45\rangle$). The new knowledge base, i.e., with two new conclusions, is shown in Fig. 3(c).

In case of analogy, the shared term `m` is also *cat*. The two existing similarity statements that include this term are $tiger \leftrightarrow cat\langle 0.9, 0.5\rangle$ and $cat \leftrightarrow feline\langle 0.8, 0.7\rangle$. Although the rule in Eq. 11 is directly applicable only to the second premise, DNARS uses the fact that similarity is by definition symmetric [6, 7]. Therefore, it automatically transforms the first statement into $cat \leftrightarrow tiger\langle 0.9, 0.5\rangle$ and derives two new

conclusions: *tiger* $\rightarrow$ *mammal*$\langle 0.90, 0.41\rangle$, and *feline* $\rightarrow$ *mammal* $\langle 0.80, 0.50\rangle$. The final knowledge base is shown in Fig. 3(d).

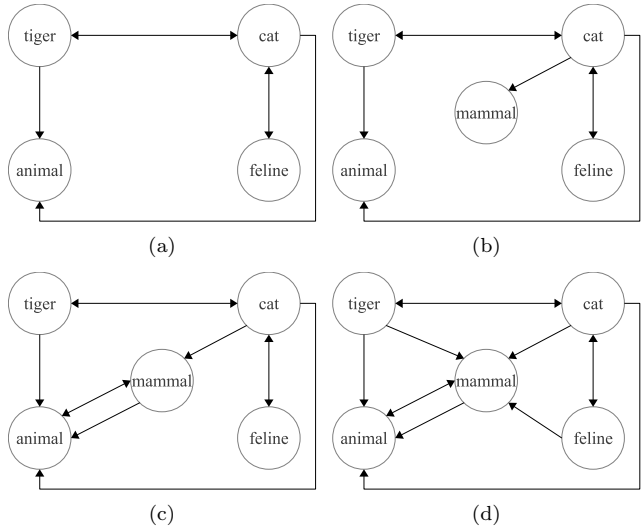

Figure 3: An example of a forward inference process in DNARS, from the initial knowledge base (a), after the addition of a new judgment *cat* $\rightarrow mammal\langle 1.0, 0.9\rangle$ (b), after applying induction and extensional comparison (c), and finally, after applying analogy (d). Unidirectional arrows represent inheritance, while bidirectional arrows represent similarity.

3.4. Resolution engine

The Resolution engine is in charge of answering questions in form of $S \rightarrow ?$ and $? \rightarrow P$. It also needs to perform this task as fast as possible. To accommodate this requirement the knowledge base in DNARS includes edge indexes.

In NAL, if there are multiple answers to a question, the *choice* rule is used to select the answer with the *higher expectation of frequency*, $e = (f - 1/2)c + 1/2$ [6, 7]. If two terms have the same expectation, the rule considers syntactic simplicity of terms, $s = 1/n^r$, where n is syntactic complexity of the term, and $r > 0$ is a system parameter. The syntactic complexity is further defined to be 1 for atomic terms or 1 plus complexities of compound term's components. If two answers have similar expectations, the simpler one is chosen [6, 7].

DNARS encodes these expressions into a numeric value that represents the edge index. More concretely, an edge between vertices S and P has two indexes: one

including the expectation of the statement and the simplicity of S, and one including the expectation of the statement and the simplicity of P. When posed a question, for example $S \rightarrow ?$, the Resolution engine sorts all candidate answers C by the indexes of edges that come out of S and into C, and then returns the best one (or n best ones).

The use of indexes speeds up the Resolution engine's execution significantly. It, however, may slow down the forward inference, the indexes need to be updated as edges are added or as the truth-value of an existing edge is changed. However, there are no strict time constraints for the forward inference [7] and it is usually executed in the background, so this is a suitable trade-off.

The question answering process of the DNARS' Resolution engine is shown in Listing 3. The main function (answer) accepts the question and the desired number of answers and returns a list of terms that fit the missing element. It relies on two helper functions, `bestSubjects` for $? \rightarrow P$ and `bestPredicates` for $S \rightarrow ?$.

Listing 3: The question answering process performed by the DNARS' Resolution engine.

```
def answer(question: Statement, limit: Int = 1): List[Term] = {
  if (question.subj == Question) {
    bestSubjects(question.pred, question.copula, limit)
  } else if (question.pred == Question) {
    bestPredicates(question.subj, question.copula, limit)
  } else {
    throw new IllegalArgumentException("Invalid question format.")
  }
}

def bestPredicates(subj: Term, copula: String, limit: Int): List[Term] =
  getV(subj) match {
    case Some(v) =>
      v.asInstanceOf[TitanVertex].query()
        .labels(copula)
        .direction(Direction.OUT)
        .orderBy("predExp", Order.DESC)
        .limit(limit)
        .vertices()
    case None =>
      List()
  }
```

The execution sequence of the helper function `bestPredicates` can be summarized as follows. First, the expression `getV(subj)` returns the vertex corresponding to the question's known term S. If this vertex does not exist in the knowledge base, there are no possible answers. Otherwise:

- Keep only the edges that match the question's copula;
- Out of those, keep only the edges that come out of the known term;
- Sort them in a descending order by the value that encodes the expectation of frequency and the syntactic simplicity of the missing predicate term;
- Keep only the first *limit* edges and get their target vertices.

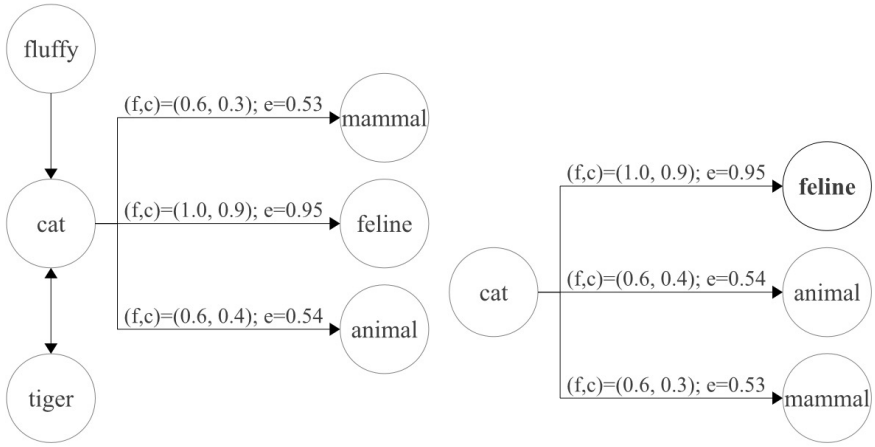

Figure 4: For the question $cat \rightarrow ?$, the best answer is *feline*. Knowledge base initially includes three *cat*-related statements (a), which are sorted during the question answering process, and according the coresponding edges' indexes (b).

As a concrete example, Listing 4 shows a knowledge base of five statements describing a cat. Fig. 4(a) shows how these statements are stored in the graph. When the system receives the question $cat \rightarrow ?$, the Resolution engine will:

- Exclude *tiger*, as it is related to *cat* through similarity;
- Exclude *fluffy*, as in this relation *cat* represents the target vertex;
- Sort the remaining edges as described previously. The resulting graph is shown in Fig. 4(b); and
- Keep only the first edge, and return *feline* as the best possible answer.

Listing 4: Initial knowledge base of five statements describing a cat.

```
fluffy → cat ⟨1.0, 0.9⟩
cat ↔ tiger ⟨1.0, 0.9⟩
cat → mammal ⟨0.6, 0.3⟩
cat → feline ⟨1.0, 0.9⟩
cat → animal ⟨0.6, 0.4⟩
```

The Resolution engine is also in charge of performing backward inference. In this case, it accepts a question in form of $S \rightarrow P$. If the answer is not directly available in the system's knowledge base, it will try to derive it using the backward inference process [6, 7]. Since this process can take longer time to complete, it will be performed asynchronously, and the client will be notified of the result through the Event stack, described next.

3.5. Event manager

Event manager in DNARS is designed using the well-known *Observer* design pattern. In this pattern, the *subject* maintains a list of *observers*, and notifies them of state changes. The Observer pattern is most commonly used in event notifications; for example, in the Java *Swing* GUI library, observers are built by implementing corresponding listener interfaces.

The internal functioning of the Event manager is shown in Fig. 5. Each Knowledge domain has a single Event manager associated with it, and the domain publishes descriptions of changes to the manager. At the same time, interested clients are registered to receive notifications from the manager.

The Event manager holds two lists: the list of pending events, and the list of observers. Both lists are directly controlled by internal *Event Dispatch Threads* (EDTs). An EDT polls pending events and notifies all registered observers. It uses simple synchronization primitives in order to prevent data corruption (e.g., missed events). The actual event dispatching is designed to be extensible and can therefore support a variety of external observers.

4. *Speed evaluation*

As discussed in the previous section, the Resolution engine of DNARS is responsible for answering questions. For questions containing "?" (i.e., S *copula* ? or ? *copula* P), the engine returns the best possible candidate for the missing term.

For questions in the form of "S *copula* P" it checks whether the corresponding statement exists in the knowledge base, or whether it can be derived using NAL's inference rules.

The backward inference engine can take an undetermined amount of time to execute [6]. On the other hand, one of the functional requirements of DNARS is to answer the first type of questions as quickly as possible, in real-time. The following case study has been designed to evaluate this capability of the Resolution engine.

The large knowledge base needed for this evaluation has been extracted from the *DBpedia* datasets [15], which are briefly discussed next.

4.1. DBpedia

DBpedia is a community-driven project aimed at organizing and structuring the information extracted from the free *Wikipedia* encyclopedia[6]. The project's goal is to "... make it easier for the huge amount of information in Wikipedia to be used in some new interesting ways. Furthermore, it might inspire new mechanisms for navigating, linking, and improving the encyclopedia itself."[7]

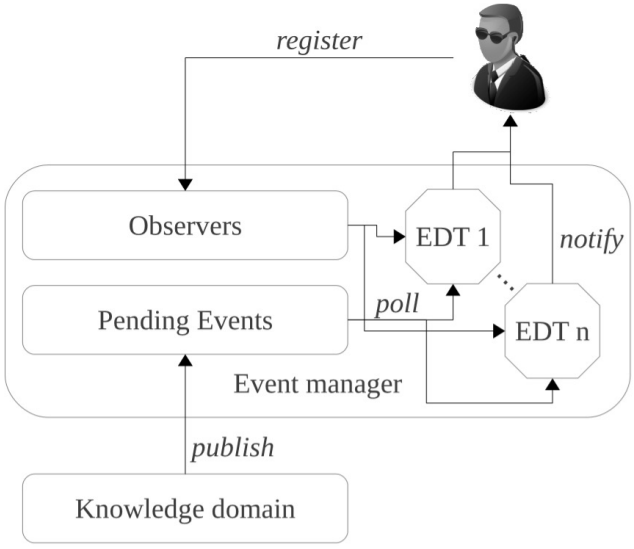

Figure 5: Internal organization of the Event manager.

DBpedia information is available in form of *Resource Description Framework* (RDF) statements. Each DBpedia entity is identified using an *International Re-*

[6] `http://www.wikipedia.org/`, retrieved on December 15, 2024.
[7] `http://dbpedia.org`, retrieved on December 15, 2024.

source Identifier (IRI), which is derived from the corresponding Wikipedia entry. The data is organized into a number of datasets[8].

The DBpedia dataset used in this case study is named *Mapping-based Properties (Cleaned)*. It includes information extracted from Wikipedia *infoboxes*. Here, the infobox is a summary of the entire article, and contains important facts and statistics. It is usually present in form of a table on the right side of the article. The dataset is cleaned and improved through the use of heuristic inference [16].

In essence, each DBpedia dataset contains different information (or different form of information) about a particular entity. As an example, Listing 5 shows how Albert Einstein is described in the dataset.

Listing 5: Descriptions of Albert Einstein in the *Mapping-based Properties (Cleaned)* DBpedia dataset.

```
<http://dbpedia.org/resource/Albert_Einstein>
  <http://xmlns.com/foaf/0.1/name> "Albert
  Einstein"@en .
<http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/ontology/birthDate>
  "1879-03-14"^^<http://www.w3.org/2001/XMLSchema#date> .
<http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/ontology/birthPlace>
  <http://dbpedia.org/resource/German_Empire> .
<http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/ontology/residence>
  <http://dbpedia.org/resource/Switzerland> .
  <http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/ontology/residence> <http://dbpedia.org/resource/United_States> .
  <http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/ontology/spouse> <http://dbpedia.org/resource/Mileva_Mari%C4%87>
  .
  <http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/ontology/field> <http://dbpedia.org/resource/Physics> .
```

DBpedia is being actively developed and improved, and has inspired several interesting projects. For example, the research presented in [17] shows how DBpedia datasets can be used to improve natural language processing. Similarly, the *DBpedia Spotlight* project [18] can be used to identify DBpedia resources in unstructured texts.

[8] `https://www.dbpedia.org/resources/databus/`, retrieved on December 15, 2024.

4.2. Representing DBpedia statements

The RDF-based statements from the DBpedia dataset have been imported into the DNARS Backend knowledge base by using arbitrary relations of NAL-4 [6, 7]. For example, the following RDF statement describes one property of Albert Einstein:

```
<http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/ontology/field>
  <http://dbpedia.org/resource/Physics> .
```

In the given statement, the first line represents the subject (*Albert Einstein*), the second line represents the predicate (his *field* of study), while the third line represents the object (*physics*). The field of study is the arbitrary relation, so the corresponding NAL-4 statement is written as follows:

```
(× <http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/ontology/field>) →
  <http://dbpedia.org/resource/Physics>
```

In order to more easily apply inference rules, the NAL-4 statement can be structurally transformed into so-called *extensional* and *intensional images* [6, 7]. These images are just different forms of the same statement; they show how individual atomic terms from the original statement link to the remaining parts of the statement. For the statement given above, the two extensional images are written as follows:

```
<http://dbpedia.org/resource/Albert_Einstein> →
  (/ <http://dbpedia.org/ontology/field> ◇
  <http://dbpedia.org/resource/Physics>)
<http://dbpedia.org/resource/Physics> →
  (/ http://dbpedia.org/ontology/field>
  <http://dbpedia.org/resource/Albert_Einstein> ◇)
```

Instead of creating images at runtime, DNARS stores three NAL statements per one RDF statement. This design approach was made in order to improve the runtime efficiency of the Resolution engine. It represents a standard practice – systems that work with NoSQL databases often repeat the stored information in order to improve their runtime efficiency [19]. As noted earlier, NAL-based knowledge bases are actually property graphs: directed multi-relational graphs with any number of properties attached to vertices and edges [12, 3]. Once the entire *Mapping-based Properties (Cleaned)* dataset was imported into DNARS, the resulting graph consisted of approximately 60 million vertices and 77 million edges. According to today's standards, the graph can be called *large* (e.g. [20]).

4.3. Speed benchmarks

The experiments were performed in clusters provided by the *Microsoft Azure* cloud computing platform[9]. Two types of machines were used (both using an SSD storage):

- D3 : 4 virtual CPUs, 14 GB of RAM.
- D4 : 8 virtual CPUs, 28 GB of RAM.

In order to simulate large numbers of concurrent users, the *Yahoo! Cloud Serving Benchmark* (YCSB) was used [21]. YCSB is an open-source tool[10] designed for load-testing of (primarily) NoSQL databases. It can be configured through a range of parameters, most important of which is the desired number of operations per second (throughput), but also the number of concurrent threads, maximum execution time, etc.

Two types of scenarios were examined: read-only and read-write. In the read-only scenario, clients only ask questions and no writing to the DNARS knowledge base is performed. The read-write scenario, on the other hand, is more realistic (and computationally more demanding), since some clients ask questions, while others add new knowledge to the system.

Within each scenario, DNARS was deployed on three different hardware configurations. The goal was to determine how the underlying hardware affects the system's performance.

YCSB client was executed on a separate D4 machine, and was configured to use 100 threads. The CPU utilization on the client machine was never over 20%, so it did not represent the bottleneck.

The YCSB client executed a number of test-cases, each lasting for 1 hour. The desired throughput (i.e., the number of questions per second) was increased for each test-case, until the system could not reach it anymore. The efficiency of DNARS is expressed in terms of average, 95th percentile and 99th percentile latencies. The later two values indicate the maximum latencies exhibited by, respectively, 95% and 99% of clients [21].

Finally, DNARS was restarted before each test-case. Questions were constructed by selecting random statements from the dataset. Approximately 80% of questions that were asked were new, while the remaining 20% were repeated questions. This put an additional strain on the system, as it could not fully benefit from answer caching.

The question answering capabilities of DNARS in the read-only scenario are shown in Fig. 6. More specifically, Fig. 6(a) shows the performance of DNARS

[9] `http://azure.microsoft.com/en-us/`, retrieved on December 26, 2024.
[10] `https://github.com/brianfrankcooper/YCSB/`, retrieved on December 26, 2024.

on a single D3 node, Fig. 6(b) shows its performance on a single D4 node, while Fig. 6(c) shows how the system performs when it's distributed over two D3 nodes.

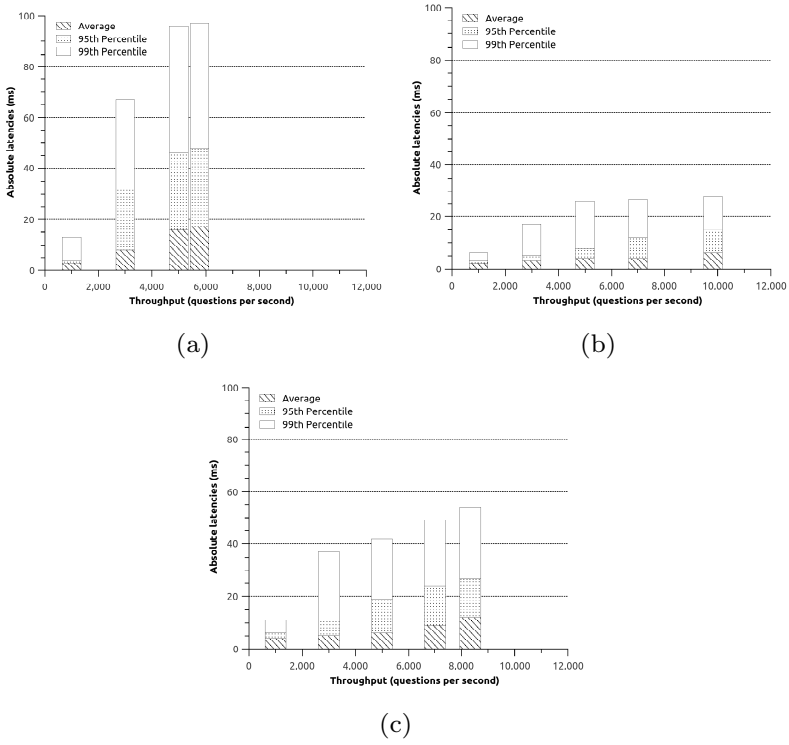

Figure 6: Runtime performance of DNARS in the read-only scenario, on (a) a single D3 node, (b) a single D4 node, and (c) two D3 nodes.

The obvious conclusion for all three configurations is that DNARS performs exceptionally well. On the lowest hardware configuration (Fig. 6(a)), the system is capable of answering almost 5800 questions per second, with the 99th percentile latency being 100 milliseconds. Once the number of virtual CPUs is doubled (Fig. 6(b)), the maximum number of answers per second jumps to over 9200, with 99% clients having to wait no more than 30 milliseconds. In the final hardware configuration (Fig. 6(c), two D3 machines), DNARS can provide answers to approximately 8300 questions per second, in which case the 99th percentile latency is just over 50 milliseconds.

The underlying Apache Cassandra database was obviously an excellent choice for the backend storage, as it can efficiently use all the available hardware resources. Vertical scaling (i.e., adding more virtual CPUs) yields better performance than horizontal scaling (i.e., adding more machines). However, in addition to practical

limitations of vertical scaling, horizontal scaling has one major advantage – it can provide fault-tolerance through data replication.

For the second, read-write scenario, an additional YCSB client was launched on a separate machine. Its task was to add 100 statements per second to the DNARS knowledge base, throughout the duration of the experiment. Moreover, it added only statements that already existed in the knowledge base. This is because adding an existing statement is slower than adding a new statement. In the first case, the system needs to read the existing truth-value from the hard- disk, perform revision, and write the new value back (which will also update the database indexes).

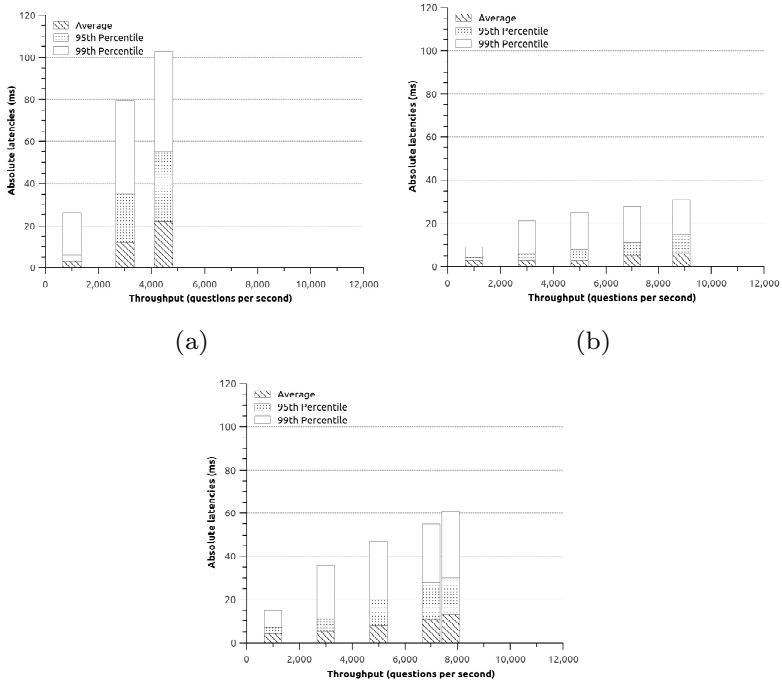

Figure 7: Runtime performance of DNARS in the read-write scenario, on (a) a single D3 node, (b) a single D4 node, and (c) two D3 nodes.

Again, three different hardware configurations were deployed – one D3, one D4, and two D3 machines – and the results are shown in Fig. 7. Obviously, simultaneous writing to the database incurs some runtime penalty, and the latencies are generally higher than in the read-only scenario. Nonetheless, the results can still be considered excellent.

The most affected configuration is the single D3 node (Fig. 7(a)), but it can still deliver 4400 answers per second, with 99% clients having to wait up to 100 milliseconds. The simultaneous writing to the database did not affect the D4 node as

much, since it was still capable of answering over 8800 questions per second, while keeping the 99th percentile latency at 30 milliseconds. Finally, when distributed over two D3 nodes, in the read-write scenario DNARS can answer 7700 questions per second with the 99th percentile latency at 60 milliseconds. These experiments

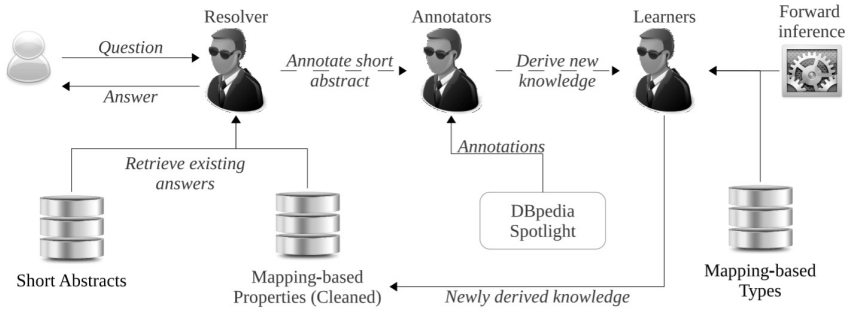

Figure 8: Execution flow of the case study for deriving new structured knowledge.

confirm that the functional requirement imposed on the Resolution engine in Section 3 has been fulfilled. That is, the engine is capable of supporting a large knowledge base and providing real-time responses to high numbers of external clients.

5. *Practical application*

One final question still remains – can the current implementation of DNARS solve a concrete practical problem? The case study presented in this section provides an affirmative answer and validates the overall work of the paper.

The main goal of this case study is to derive new structured knowledge base for DBpedia using information available in unstructured texts. More concretely, the case study derives new knowledge for the *Mapping-based Properties (Cleaned)*, using information in *Short Abstracts* and *Mapping-based Types*. The *Short Abstracts* dataset contains short abstracts of Wikipedia articles. The same version of the *Mapping-based Types* dataset describes types/classes of approximately 28 million entities.

The case study is shown graphically in Fig. 8. It uses a number of intelligent agents to fulfill the design goal. The overall execution sequence can be described in 5 distinctive steps.

Step 1. The case study is started by an end-user, who asks a question about a specific resource. The question ends up in the Resolver agent, which returns all the information available in *Short Abstracts* and *Mapping-based Properties (Cleaned)*. The agent relies on the Resolution engine to find the required answers.

Step 2. Once the answers are returned the Resolver activates the *Annotator* agent. This new agent annotates the unstructured text obtained from *Short Abstracts*, by invoking the DBpedia Spotlight RESTful web service. In response, the agent receives a list of DBpedia resources found in the text. Now, the system needs to determine the exact relations between the properties of the initial resource and the received annotated resources.

Step 3. For each annotated resource, the Annotator creates an instance of the *Learner* agent. The Learner agent first retrieves all statements relevant to its resource. Relevant statements are answers to questions $R \rightarrow ?$ and $? \rightarrow R$, where R denotes the agent's resource. The answers are retrieved from the *Mapping-based Properties (Cleaned)* dataset.

Step 4. Now, the Learner agent employs the Forward inference engine to derive *intermediary* conclusions. Known properties of the initial resource are used as the knowledge base, while the relevant statements represent new judgments. Intermediary conclusions derived in this step serve as initial links between prop- erties of the initial resource and properties of annotated resources.

Step 5. Finally, intermediary conclusions obtained in the previous step are again matched against properties of the initial resource, to derive the set of conclusions. This set is first filtered, merging duplicate statements using the *revision* rule [6, 7], and resulting in the final, new structured knowledge about the initial resource.

5.1. A concrete execution example

In this sub-section, we will illustrate the above steps on a concrete example. Let the end-user ask: *Albert Einstein* $\rightarrow ?$ ("Who was Albert Einstein?"). In Step 1, the Resolver uses the Resolution engine to retrieve the short abstract and the list of existing properties from the knowledge base. In Step 2, the Annotator sends the short abstract to the DBpedia Spotlight web service and receives the four DBpedia resources shown in Listing 6[11].

Listing 6: Annotated resources detected in the short abstract for *http://dbpedia.org/resource/Albert_Einstein*.

```
<http://dbpedia.org/resource/General_relativity>
<http://dbpedia.org/resource/Max_Born>
<http://dbpedia.org/resource/Quantum_mechanics>
<http://dbpedia.org/resource/Theoretical_physics>
```

[11] Only resources available in the *Mapping-based Properties (Cleaned)* dataset are shown.

At this point the system knows that these resources are somehow related to Albert Einstein. Now it needs to determine the exact relations. This is initiated in Step 3. A new Learner agent is created for each annotated resource in Listing 6. The agent first retrieves statements relevant to its resource. Then, in Step 4, it uses forward inference with known properties of Albert Einstein as the knowledge base, and the relevant statements as new input judgments. This step derives a set of intermediary conclusions (the total of 249 for all Learners). For example, intermediary conclusions include 13 statements stating that general relativity is similar to physics, that is:

<http://dbpedia.org/resource/General_relativity> $\leftrightarrow$
  <http://dbpedia.org/resource/Physics> $\langle 1.00, 0.45 \rangle$

In the final step (Step 5) each Learner again applies the forward inference using known properties of Albert Einstein as the knowledge base and intermediary conclusions now as new input judgments. Conclusions derived in this step are first filtered to merge duplicate statements and to exclude already known properties.

The end-result – the newly derived structured knowledge about Albert Einstein – is shown in Listing 7. Only statements with the confidence level of 0.9 or higher are taken into account as this is the value assigned to existing statements [6].

Listing 7: The newly derived structured knowledge about Albert Einstein.

```
(× <http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/resource/General_relativity>) →
    <http://dbpedia.org/ontology/field> ⟨1.00, 0.90⟩

(× <http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/resource/Quantum_mechanics>) →
    <http://dbpedia.org/ontology/field> ⟨1.00, 0.92⟩

(× <http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/resource/Theoretical_physics>) →
    <http://dbpedia.org/ontology/field> ⟨1.00, 0.99⟩
```

Manual inspection of these results confirms that they are correct. General relativity, quantum mechanics, and theoretical physics were indeed Einstein's fields[12]. This information is present in the *Short Abstracts* but not in the *Mapping-based Properties (Cleaned)* dataset (Listing 5) and represents new structured knowledge.

The fourth annotated resource denoting the physicist Max Born could not be linked to Einstein in a confident manner. The *Mapping-based Properties (Cleaned)*

[12]However, he was displeased with the principles of quantum mechanics and was trying to disprove the theory [22].

dataset includes statements about Albert Einstein. These statements use the following set of relations: *doctoral advisor, academic advisor, name, birth place, birth date, death place, death date, residence, spouse*, and *field*. Since at the current level DNARS cannot derive new relations, it has incorrectly concluded that Born was Einstein's doctoral advisor[13]. Although incorrect, it would have been much worse if the systemhad derived:

```
(× <http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/resource/Max_Born>) →
    <http://dbpedia.org/ontology/birthPlace> ⟨1.00, 0.90⟩
```

Therefore, DNARS has still selected the arguably best possible relation among the available ones. It is also worth noting that this conclusion had a lower confidence value then the required 0.9.

5.2. Analysis of the reasoning process

Let us now analyze how the first conclusion in Listing 7 was derived. At some point during the reasoning process, the Forward inference engine takes the two premises shown in Listing 8.

Listing 8: The initial premises that will lead to the conclusion that General relativity was Einstein's research field.

```
(× <http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/resource/Physics>) →
    <http://dbpedia.org/ontology/field>

(× <http://dbpedia.org/resource/Charles_W._Misner>
  <http://dbpedia.org/resource/General_relativity>) →
    <http://dbpedia.org/ontology/field>
```

In this case, the system can use *intensional comparison* to derive a similarity between two compound terms. Intensional comparison takes two inheritance premises and derives a similarity statement as follows [6, 7]:

$$\{M \rightarrow P\langle f_1, c_1\rangle\}, M \rightarrow S\langle f_2, c_2\rangle\} \vdash S \leftrightarrow P\langle f, c\rangle \tag{12}$$

The result is shown as the first statement in Listing 9. Since the relation between compound terms defines relations between their respective components, this statement can be transformed into the latter two intermediary conclusions in Listing 9. The *Mapping-based Types* dataset is employed during this transformation

[13]The two famous physicist were, however, colleagues and friends [22].

step. The transformation will be applied only if the related components belong the same type, preventing the system to conclude that, for example, a person is similar to a geographic location.

This process of deriving and transforming intermediary conclusions is repeated for many other physicists in the *Mapping-based Properties (Cleaned)* dataset, and the revision rule steadily increases the system's confidence about the fact that General relativity is similar to Physics. On the other hand, the similarity between Charles W. Misner and Albert Einstein is derived only once, retaining a relatively low confidence (although the two are similar to some respect).

Listing 9: The first statement represents an intermediary conclusion derived by applying intensional comparison to the premises from Listing 8, while the latter two statements are obtained by transforming the first one.

```
(× <http://dbpedia.org/resource/Charles_W._Misner>
  <http://dbpedia.org/resource/General_relativity>) ↔
    (× <http://dbpedia.org/resource/Albert_Einstein>
      <http://dbpedia.org/resource/Physics>)

<http://dbpedia.org/resource/Charles_W._Misner> ↔
  <http://dbpedia.org/resource/Albert_Einstein>

<http://dbpedia.org/resource/General_relativity> ↔
  <http://dbpedia.org/resource/Physics>
```

In the final inference step the system uses the analogy rule [6, 7, 10]:

$$\{M \rightarrow P\langle f_1, c_1\rangle, S \leftrightarrow M\langle f_2, c_2\rangle\} \vdash S \rightarrow P\langle f, c\rangle \tag{13}$$

The first premise in the rule is the following known statement:

```
(× <http://dbpedia.org/resource/Albert_Einstein>
  <http://dbpedia.org/resource/Physics>) →
    <http://dbpedia.org/ontology/field>
```

or, more concretely, its extensional image:

```
<http://dbpedia.org/resource/Physics> →
  (/ <http://dbpedia.org/ontology/field>
    <http://dbpedia.org/resource/Albert_Einstein> ◇)
```

The second premise is the following intermediary conclusion:

```
<http://dbpedia.org/resource/General_relativity> ↔
  <http://dbpedia.org/resource/Physics>
```

At the end, the rule derives the final, highly-confident conclusion that General relativity was an additional research field of Albert Einstein.

6. *Related work*

Non-axiomatic logic (NAL) is different from many other logics used by the artificial (general) intelligence researchers. Its differences stem from the fact that NAL is a term logic with syllogistic inference rules and the experiencegrounded semantics. Comparisons of NAL and other formalisms can be found in e.g. [9, 6, 7, 23] and related research papers.

Throughout the literature, concrete system architectures developed as part of the AGI research are referred to as *cognitive* or *reasoning*. Although the two terms denote similar things, there are some differences. As discussed in [6, 7], reasoning is performed at a higher-level of abstraction and includes one or more cognitive functions, such as decision making and learning. It is not concerned with lower-level details, such as perceptual and motor skills, often found in cognitive architectures. However, NAL can be directly applied to lower-level cognitive functions, like perception and motion, as soon as the terms involved directly represent percepts and actions[14].

Cognitive architectures can generally be organized into three categories: *symbolic*, *emergent* or *connectionist,* and *hybrid* [24, 25]. Symbolic architectures manipulate symbols at a higher level of abstraction, whereas the emergent architectures incorporate individual units for processing lower-level signals that flow through the network. Hybrid architectures represent combinations of the earlier two.

Because of its semantics, DNARS is not a traditional symbolic architecture, but has many connectionist features, through it is not a hybrid [26]. This section first analyzes a number of well-established symbolic and hybrid reasoning and cognitive architectures. For additional information on these and other systems, see e.g. [24, 27, 25].

Intelligent agents represent one of the most-thriving fields of artificial intelligence with numerous practical applications [28, 29]). One of the main intended purposes of DNARS is to serve as an underlying reasoning engine in our multiagent environment named *Siebog* [30, 31]. This is a major departure from the *Belief-Desire-Intention* (BDI) model commonly used by the agent technology researchers and practitioners [32]. Therefore, the second part of this section discusses several influental BDI implementations.

[14] `https://github.com/opennars/opennars/wiki/Perception-In-NARS`, retrieved on December 15, 2024.

6.1. Symbolic and hybrid architectures

ACT-R is a hybrid cognitive architecture, based on the so-called *Unified Theories of Cognition* [33], as well as cognitive neuroscience research [24, 34, 35]. For example, the ACT-R operation is based to a certain extent on the experimental data obtained from neuroimaging, such as *Functional Magnetic Reasoning Imaging* (fMRI) and by observing how different parts of the brain interact during the reasoning process. As such, ACT-R can also be used as a framework for emulating human reasoning.

The most important components of the ACT-R architecture include the *perceptual-motor* sub-system, for obtaining visual information about the world and performing physical actions, the *goal module*, which manages the system's intentions, and the *declarative module*, which holds the system's overall knowledge [35]. A limited amount of the information from each component is stored into corresponding *buffers*, to be used by the *central production system* for component coordination. Symbolic pieces of information (*chunks* in declarative, or *productions* in procedural knowledge) are also described by numerical parameters, allowing the construction of an *associative memory/network* [24, 34].

ICARUS is a symbolic cognitive architecture with several types of memories [24]. Its *perceptual* memory includes descriptions of observed objects, the *belief* memory describes relations among objects, while the *conceptual* memory holds general knowledge. In each inference cycle, the system observes its environment and creates a set of percepts. The percepts are then matched against the conceptual knowledge to deduce new beliefs. Additional two memories are introduced to guide and control the system's behavior. *Goal* memory includes the system's actively managed goals, while the *skill* memory describes complex, hierarchical activities that the system can perform. It is worth noting that, among the cognitive architectures described in this sub-section, the architecture of ICARUS bears the closes resemblance to the BDI agent architecture.

OpenNARS is a reference open-source implementation[15] of non-axiomatic reasoning [6, 7]. The latest version implements the logic of all 9 layers of NAL as defined in [6]. Its architecture consists of the memory module, the inference engine, and a *control* mechanism, which handles the system's reasoning cycles [7].

Soar is one of the earliest, and a well-known symbolic cognitive architecture [39, 24]. As ACT-R, it represents a concrete realization of the Unified Theories of Cognition. Soar programs are specified in the form of *if-then production rules*, which, in turn, are used to select and apply *operators* and execute actions. The system's knowledge is divided into the *long-term* and *working memory*. The long-

[15] `https://github.com/opennars/opennars`, retrieved on December 15, 2024.

term memory can be *procedural*, containing the knowledge on how to do things, *semantic*, containing declarative knowledge about the world, and *episodic*, which summarizes the previous experience.

The working memory of Soar contains knowledge that is relevant to the current situation, and is directly tied to the perception, action, and decision making modules. Several extensions of the core Soar architecture have been proposed as well, including the use of Reinforcement learning in operator selection, visual imagery modules, semantic and episodic learning, etc. [24].

OpenNARS and DNARS represent concrete realization of non-axiomatic reasoning. Their differences from other cognitive and reasoning architectures stem from the use of NAL as the underlying formalism. For example, no other system deals with the issue of insufficient knowledge and resources to the degree done in NAL. In addition, unlike many systems described here, OpenNARS and DNARS are more focused on emulating the human thought processes at a higher level of abstraction, rather than trying to accurately model the human brain [7]. However, it remains to be seen which approach works the best, as all the systems are yet far from reaching the goal of building a "thinking machine."

DNARS is built by combining NAL and the Big Data paradigm, because the two try to solve a similar issue: how to handle and process large amounts of information with limited time and resources. NAL, for example, includes inference rules that deal with knowledge inconsistencies only when necessary, e.g., when there are different answers to the same question. It also includes constructs for combining individual pieces of information and reducing the amount of raw information. Similarly, the NoSQL database used in DNARS includes a number of techniques for dealing with large amount of information and strict time constraints, and temporarily sacrifice information consistency if needed. Therefore, in DNARS we combine the "best of both worlds" in an efficient manner.

There are some important differences between OpenNARS and DNARS. OpenNARS has been developed for a significantly longer period of time, and is a more mature product. In the latest version, OpenNARS implements all layers of NAL, and includes more advanced control mechanisms. The main advantage of DNARS, however, is in the organization of its backend knowledge base. That is, DNARS is currently capable of reasoning over much larger knowledge bases than OpenNARS. By utilizing modern approaches to large-scale data processing, DNARS can easily be used to, for example, realize the case study presented in Section 5.

6.2. Concrete BDI implementations

As noted, BDI is the most popular model for developing intelligent agents. Over time, several interesting concrete realization of the model has been proposed [40–44].

BDI4JADE extends JADE with the support for BDI agents [40]. Its authors argue that, although sometimes convenient, agent-oriented programming languages usually represent a barrier that limits the wider adoption of the BDI model. Therefore, the BDI4JADE framework is based on pure Java.

BDI4JADE agents are defined through their *capabilities*, which include plans and relevant beliefs along with public interfaces. Additional essential components include desires, intentions and goals, with their usual meanings, *events* that signal changes in the goal and belief bases, as well as *strategies* for customizing the reasoning cycles. A reasoning cycle includes a number of steps [40], which can be summarized as follows. The agent first revises its belief base, removes completed goals, and then proceeds to choosing a set of *applicable* goals (i.e., desires). A subset of desires is selected for achievement becoming the agent's intentions. Finally, active intentions are associated with plans that can fulfil them.

Procedural Reasoning System (PRS) is one of the earliest agent architectures based on the BDI model [41]. It includes four databases, containing agent's beliefs, goals, declarative procedures (i.e., plans), and intentions (i.e., active plans). These databases are managed the *interpreter*, which operates in reasoning cycles. In each cycle it selects *applicable* plans, whose pre-conditions match the current beliefs and goals. One applicable plan is then selected, placed on the intention stack, and then executed. During the execution, new beliefs and/or goals may be generated, which will create new intentions. Finally, it is worth noting that multiple PRS interpreters can operate in parallel and communicate with each other.

GOAL is a practical agent-oriented programming language [45, 42]. The *mental state* of a GOAL agent is defined through a static knowledge base, a dynamic belief base, as well as different types of goals. Active goals are removed from the agent's mental state using the *blind commitment strategy*, which means that only successfully achieved goals are dropped [45, 42]. The action execution strategy is guided by so-called *action rules*. They are specified in the form of *IF mental state THEN action*. If the given mental state is true, the action is said to be *applicable*. An action that is both applicable and enabled is called an *option*. Action rules can be checked in several way (e.g. in the order they are written, randomly, etc.), and the first action that becomes an option is executed.

Jadex follows the object-oriented model for representing beliefs and goals, instead of the more common approach based on logical formulae [43]. The Jadex infrastructure consist of the agent platform (e.g. standalone or JADE), *active components*, and *kernels*. Active components, broadly speaking, represent the merger of agents and service-oriented systems, while kernels define internal workings of active components. Here, the most important is the *BDI agent kernel* [43]. It is based on the PRS described earlier, with the addition of the *goal deliberation* technique for maintaining a consistent set of goals.

Jason is a popular interpreter for an agent-oriented programming language *AgentS-*

peak and a reasoning engine for BDI agents [44]. Agents are defined in terms of beliefs, goals, and plans. The interpreter operates in reasoning cycles, divided into 10 individual steps. First, the agent perceives its environment (generating a perceptual information), processes a single message received from another agent, while filtering-out "socially unacceptable" messages, and updates its belief base accordingly. The remaining six steps represent the core of agent's reasoning and acting:

- A single *event* is selected to be processed. An event represents a change in the agent's mental state (e.g. a new belief has been added).
- A set of *relevant* plans, i.e., plans corresponding to the selected event, is constructed.
- Of those, a set of *applicable* plans (also called *options*) is determined.
- An applicable plan is put on a stack to become an *intention*. This is the plan to which the agent will *commit*.
- An intention is selected from the stack.
- A single step of the selected intention is executed.

Jason is designed as a highly-customizable architecture, and has been integrated with a number of other agent-based systems.

NAL provides a number of advantages over the traditional BDI model. First and foremost, NAL statements are associated with truth-values. In concrete BDI implementations discussed earlier, there is no way of expressing the agent's confidence in a belief; it is left to the agent developer to somehow handle the notion that a belief might not be true. NAL statements, on the other hand, are *beliefs* in their true definition.

Additionally, unlike the BDI model, inconsistency resolutions (through backward inference), learning (through forward inference), and working under the assumption of insufficient knowledge and resources (e.g. compound terms [6, 7]), represent inherent features of NAL-based agents.

These are the main advantages of DNARS over the presented BDI systems. Finally, as in OpenNARS-DNARS comparison, DNARS offers the possibility of reasoning over much larger knowledge bases than any existing BDI system. This opens up DNARS to a wider range of possible practical applications, as demonstrated by the case study in Section 5.

7. *Conclusions and future work*

This paper has presented a new general-purpose reasoning architecture named *Distributed Non-Axiomatic Reasoning System* (DNARS). DNARS uses the *NonAxiomatic Logic* (NAL) as its formal reasoning framework. NAL provides a well-defined syntax, experience-grounded semantics, and a set of inference rules, while working under the *assumption of insufficient knowledge and resources* [6]. The main novelty of DNARS, when compared to other similar systems, is its ability to efficiently handle large quantities of knowledge, while providing service to high numbers of external clients. This ability was achieved by a uniquely designed back-end knowledge base, and a set of algorithms that adequately realize NAL inference rules in these distributed, highly-scalable settings.

This runtime performance of DNARS has been demonstrated in practice through the speed evaluation case-study presented in Section 4. A large knowledge base of DBpedia statements has been built, and then a number of both read-only and read-write scenarios have been executed. In all settings, DNARS has performed exceptionally well and has fulfilled its design goal.

Finally, one practical application of DNARS has been presented in Section 5. It has been shown how the Forward inference engine in DNARS [46, 47] can be used to derive new structured knowledge from unstructured texts, and integrate it with existing known facts about a topic.

In the future, obviously, the remaining layers of NAL need to be added to DNARS. Although the first four layers currently implemented in DNARS are sufficient for simple reasoning tasks, the remaining layers would provide it with higher-level reasoning capabilities. In particular [6]:

- Starting from layer NAL-5, statements can be used as terms, and new copulas (such as *implication* and *equivalence*) are supported.
- NAL-6 adds support for variables, and would enable DNARS to work with more general rules.
- NAL-7 introduces the concepts of time and events, as well as temporal connectors (e.g. *sequential* and *parallel*) and relations (e.g., *before* and *when*).
- Procedural knowledge, in form of operations and goals, is added in NAL-8.
- Finally, NAL-9 would add the capability of processing emotions, and exhibiting self-monitoring and self-control.

Inspired by DNARS, a new high level programming language ALAS has been developed [48]. This agent-oriented domain-specific language supports Distributed Non-Axiomatic Reasoning for simpler development of intelligent distributed multi agent systems. ALAS allows programmers to develop intelligent agents easier by using domain specific constructs. Further improvement of ALAS and its applications offer new research challenges.

Acknowledgement. This work was partially supported by Ministry of Science, Technological Development and Innovation of the Republic of Serbia (Grants No. 451-03-66/2024-03/200125 & 451-03-65/2024-03/200125).

REFERENCES

[1] B. Baesens, *Analytics in a Big Data World: The Essential Guide to Data Science and its Applications*, Wiley and SAS Business Series, 2014.

[2] L. Taerim, L. Hyejoo, R. Kyung-Hyune, S. U. Sang, *The efficient implementation of distributed indexing with hadoop for digital investigations on big data*, Computer Science and Information Systems **11** (3) (2014), 1037–1054.

[3] I. Robinson, J. Webber, E. Eifrem, *Graph Databases*, O'Reilly Media, Inc., 2013.

[4] J. Dean, S. Ghemawat, *MapReduce: Simplified data processing on large clusters*, Communications of the ACM **51** (1) (2008), 107–113; doi:10.1145/1327452.1327492.

[5] E. Hovy, R. Navigli, S. P. Ponzetto, *Collaboratively built semi-structured content and artificial intelligence: The story so far*, Artificial Intelligence **194** (2013), 2–27.

[6] P. Wang, *Non-axiomatic Logic: A Model of Intelligent Reasoning*, World Scientific Publishing Co. Pte. Ltd., 2013.

[7] P. Wang, *Rigid Flexibility: The Logic of Intelligence*, Vol. **34** of Applied Logic Series, Springer, Dordrecht, The Netherlands, 2006.

[8] M. A. Rodriguez, J. Geldart, *An evidential path logic for multi-relational networks*, in: Proceedings of the Association for the Advancement of Artificial Intelligence Spring Symposium: Technosocial Predictive Analytics Symposium, Vol. SS-09-09, AAAI Press, 2009, pp. 114–119.

[9] P. Wang, *Formalization of evidence: a comparative study*, Journal of Artificial General Intelligence **1** (1) (2011) 25–53; doi:10.2478/v10229-011-0003-7.

[10] P. Wang, *Analogy in a general-purpose reasoning system*, Cognitive Systems Research **10** (3) (2009), 286–296; doi:10.1016/j.cogsys.2008.09.003.

[11] M. Michael, J. E. Moreira, D. Shiloach, R. W. Wisniewski, *Scale-up x scale-out: A case study using Nutch/Lucene*, in: IEEE International Parallel and Distributed Processing Symposium, 2007, pp. 1–8.

[12] M. A. Rodriguez, J. Shinavier, *Exposing multi-relational networks to single-relational network analysis algorithms*, Journal of Informetrics **4** (1) (2010), 29–41; doi:http://dx.doi.org/10.1016/j.joi.2009.06.004.

[13] E. Hewitt, *Cassandra: The Definitive Guide*, O'Reilly Media, Inc., 2010.

[14] T. Rabl, S. Gomez-Villamor, M. Sadoghi, V. Muntes-Mulero, H.-A. Jacobsen, S. Mankovskii, *Solving big data challenges for enterprise application performance management*, Proceedings of the VLDB Endowment **5** (12) (2012), 1724–1735; doi:10.14778/2367502.2367512.

[15] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, C. Bizer, *DBpedia – a large-scale, multilingual knowledge base extracted from wikipedia*, Semantic Web Journal.

[16] H. Paulheim, C. Bizer, *Improving the quality of linked data using statistical distributions*, International Journal of Semantic Web and Information Systems **10** (2) (2014), 63–86.

[17] P. N. Mendes, M. Jakob, C. Bizer, *DBpedia for NLP: A multilingual cross- domain knowledge base*, in: Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12), 2012.

[18] P. N. Mendes, M. Jakob, A. García-Silva, C. Bizer, *Dbpedia spotlight: Shedding light on the web of documents*, in: Proceedings of the 7th International Conference on Semantic Systems (I-Semantics), 2011.

[19] A. Schram, K. M. Anderson, *MySQL to NoSQL: Data modeling challenges in supporting scalability*, in: Proceedings of the 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity, SPLASH '12, ACM, New York, NY, USA, 2012, pp. 191–202; doi:10.1145/2384716.2384773.

[20] R. C. McColl, D. Ediger, J. Poovey, D. Campbell, D. A. Bader, *A performance evaluation of open source graph databases*, in: Proceedings of the First Workshop on Parallel Programming for Analytics Applications, PPAA '14, ACM, New York, NY, USA, 2014, pp. 11–18; doi:10.1145/2567634.2567638.

[21] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears, *Bench-marking cloud serving systems with YCSB*, in: Proceedings of the 1st ACM Symposium on Cloud Computing, SoCC '10, ACM, New York, NY, USA, 2010, pp. 143–154.

[22] M. Kumar, *Quantum: Einstein, Bohr and the Great Debate About the Nature of Reality*, Icon Books Ltd., 2009.

[23] P. Wang, *Theories of artificial intelligence – meta-theoretical considerations*, in: P. Wang, B. Goertzel (Eds.), Theoretical Foundations of Artificial General Intelligence, Vol. 4 of Atlantis Thinking Machines, Atlantis Press, 2012, pp. 305–323.

[24] W. Duch, R. J. Oentaryo, M. Pasquier, *Cognitive architectures: Where do we go from here?*, in: Proceedings of the 2008 Conference on Artificial General Intelligence 2008: Proceedings of the First AGI Conference, IOS Press, Amsterdam, The Netherlands, The Netherlands, 2008, pp. 122–136.

[25] B. Goertzel, R. Lian, I. Arel, H. de Garis, S. Chen, *A world survey of artificial brain projects, Part ii: Biologically inspired cognitive architectures*, Neurocomputing **74** (1–3) (2010), 30–49.

[26] P. Wang, *Artificial general intelligence and classical neural network*, in: IEEE International Conference on Granular Computing, 2006, pp. 130–135.

[27] K. R. Thorisson, H. P. Helgasson, *Cognitive architectures and autonomy: A comparative review*, Journal of Artificial General Intelligence **3** (2) (2012), 1–30.

[28] C. Bădică, N. Bassiliades, S. Ilie, K. Kravari, *Agent reasoning on the web using web services*, Computer Science and Information Systems **11** (2) (2014), 697–721.

[29] W. Jian, C. Baigen, L. Jiang, S. Wei, *A lane-changing behavioral preferences learning agent with its applications*, Computer Science and Information Systems **12** (2) (2015), 349–374.

[30] D. Mitrović, M. Ivanović, Z. Budimac, M. Vidaković, Radigost, *Interoperable web-based multi-agent platform*, Journal of Systems and Software **90** (2014), 167–178; doi:http://dx.doi.org/10.1016/j.jss.2013.12.029.

[31] D. Mitrović, M. Ivanović, M. Vidaković, Z. Budimac, *Extensible Java EE-based agent framework in clustered environments*, in: J. Mueller, M. Weyrich, A. L. C. Bazzan (Eds.), 12th German Conference on Multiagent System Technologies, Vol. 8732 of Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 202–215.

[32] A. S. Rao, M. P. Georgeff, *Bdi agents: From theory to practice*, in: Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), 1995, pp. 312–319.

[33] A. Newell, *Unified Theories of Cognition*, Harvard University Press, 1994.

[34] C. Lebiere, J. R. Anderson, *A connectionist implementation of the ACT-R production system*, in: Proceedings of the 15th Annual Conference of the Cognitive Science Society, 1993, pp. 635–640.

[35] J. R. Anderson, D. Bothell, M. D. Byrne, *An integrated theory of mind*, Psychological Review **111** (4) (2004), 1036–1060.

[36] J. E. Laird, *The Soar Cognitive Architecture*, MIT Press, 2012.

[37] I. Nunes, C. J. de Lucena, M. Luck, *BDI4JADE: a BDI layer on top of JADE*, in: Proceedings of the Workshop on Programming Multiagent Systems, 2011, pp. 88–103.

[38] M. P. Georgeff, A. L. Lansky, *Reactive reasoning and planning*, in: Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87), 1987, pp. 677–682.

[39] K. Hindriks, *Programming cognitive agents in GOAL*, http://mmi.tudelft.nl/trac/goal/raw-attachment/wiki/WikiStart/Guide.pdf, retrieved on February 7, 2014 (March 2014).

[40] L. Braubach, A. Pokahr, W. Lamersdorf, *Jadex active components: A unified execution infrastructure for agents and workflows*, Advanced Computational Technologies (2013), 128–149.

[41] R. H. Bordini, J. F. Hubner, M. Wooldridge, *Programming Multi-Agent Systems in AgentSpeak Using Jason*, Wiley Series in Agent Technology, John Wiley & Sons Ltd, 2007.

[42] K. V. Hindriks, *Programming rational agents in GOAL*, in: A. El Fallah Seghrouchni, J. Dix, M. Dastani, R. H. Bordini (Eds.), Multi-Agent Programming: Languages, Tools and Applications, Springer US, 2009, pp. 119–157.

[43] A. Pokahr, L. Braubach, *From a research to an industry-strength agent platform: Jadex V2*, In 9. Internationale Tagung Wrtschaftsinformatik, pp. 769–778, 2008.

[44] R. H. Bordini, M. Dastani, J. Dix, A. E. F. Seghrouchni, eds. *Multiagent Programming: Languages, Tools and Applications*, Springer, 2009; doi: 10.1007/978-0-387-89299-3.

[45] Y. Yang, T. Holvoet, *Making model checking feasible for GOAL*, Ann. Math. Artif. Intell. **92** (2024), 837–853; https://doi.org/10.1007/s10472-023-09898-3.

[46] D. Mitrović, M. Ivanović, M. Vidaković, Z. Budimac, *Siebog: An enterprise-scale multiagent middleware*, Inf. Technol. Control. **45** (2) (2016), 164–174.

[47] D. Mitrović, M. Ivanović, M. Vidaković, Z. Budimac, *The Siebog multiagent middleware*, Knowl. Based Syst. **103** (2016), 56–59.

[48] D. Sredojević, M. Vidaković, M. Ivanović, *ALAS: agent-oriented domain-specific language for the development of intelligent distributed non-axiomatic reasoning agents*, Enterp. Inf. Syst. **12** (8-9) (2018), 1058–1082.

Department of Mathematics and Informatics
Faculty of Sciences
University of Novi Sad, Serbia
&
Department of Mathematics and Informatics
Faculty of Sciences
University of Novi Sad, Serbia
e-mail: mira@dmi.uns.ac.rs (Mirjana Ivanović)
&
Department of Computer and Information Sciences
Temple University, USA
e-mail: pei.wang@temple.edu (Pei Wang)