Concrete-domain Reasoning Techniques in Knowledge Cartography

Krzysztof Goczyła*, Aleksander Waloszek*, Wojciech Waloszek*

Abstract. Many phenomena in the real world are best described using symbols or numbers. The paper takes as a base the novel cartographic approach, which allows for efficient reasoning and storing conclusions for large number of individuals. In the paper an extension of the approach is presented that enables the cartographic reasoner to use in the course of inference also symbolic values like numbers and character strings. The proposed extension takes extensive use of mechanisms already present in the cartographic approach, so it can be seamlessly integrated with existing solutions. At the same time it offers expressivity covering the range of constructors exploited in DIG language and can be extended towards more expressive DL languages. The paper presents details of the methods, shows examples of its use, and compares the approach with ones utilized in other reasoning systems. The summary shows possible directions of further development of the method.

1. Introduction

In the last decades significance of knowledge management systems gradually increases. It is becoming obvious that modern systems have to use novel techniques of acquiring knowledge (especially from the Internet), managing it, and presenting in a required form to the user. In this process not only accuracy of the returned results but also the time of their retrieval is of great importance.

The authors of this paper are involved in the 6th European Union Framework Programme integrated project (priority “e-Health”) called PIPS (Personal Information Platform for Life and Health Services). PIPS’s main goal is to create a Web infrastructure to support health and promote healthy life style among European communities. PIPS concentrates on providing health-related assistance to its users: citizens, patients and healthcare professionals. PIPS serves citizens and patients by helping them in making decisions concerning their everyday practices and deciding whether they should consult their doctors. PIPS can also be used by healthcare professionals and can provide them with health-related knowledge and advise about course of therapies. Thus the way it manages the knowledge is critical for the quality of the system.

The general architecture of the Knowledge Management System in PIPS is depicted in Fig. 1. The data are acquired from external data sources and uploaded by syntactical tools into a reasoning system, which consequently processes them and publishes the results to the clients (users and other PIPS subsystems).

*Department of Software Engineering, Gdańsk University of Technology, ul. Narutowicza 11/12, Gdańsk, Poland, email: {kris, alwal, wowal}@eti.pg.gda.pl
The major component of the architecture is the reasoning system, called KASEA (name being an acronym of Knowledge Signature Analyser) Its performance is crucial for the clients to obtain interesting results in satisfying time. Due to huge number of individuals that need to be handled by the reasoning system (including, but not limited to: diseases, allergies, drugs, food products etc.) we decided to exploit in the reasoning system the novel knowledge representation scheme called knowledge cartography [3][4]. Knowledge cartography is based on Description Logics formalism for manipulating knowledge, the standard which has also been use in OWL language, part of Semantic Web initiative. Knowledge cartography focuses on storing in the knowledge base as many conclusions about concepts and individuals as possible. The conclusions can be quickly retrieved in the process of query answering, considerably improving the performance of the system. It is worth noting that the method is general and its use is not in any way constrained to the field of medicine.

The paper presents the next stage of development of cartographic approach. The development is towards handling the concrete-domain data, like character strings and numbers. The original version of knowledge cartography covered basic Description Logics \( \mathcal{ALC} \)
\(^1\), which does not cover concrete domains. The approach turned out to be limited. Many phenomena had to be modeled in somewhat artificial way. For example, results of a blood pressure tests were divided into several classes (corresponding to numerical intervals) and an external mechanism performed translation of numeric result into one of the classes.

The extension presented in this paper allows for integration of numerical and textual data into cartographic framework. The integration embraces storing such values and reasoning over them in order to obtain non-explicit conclusions concerning e.g. future treatment of a patient. The outcome of the inference is stored in the knowledge base, preserving its main virtue - the capability of quick retrieval of obtained results. Although especially focused on numerical and textual data the presented solution

\(^1\) \( \mathcal{ALC} \) is limited wrt. \( \mathcal{SHOIN} \mathcal{(D)} \) – a Description Logics dialect used in OWL-DL in several ways: e.g. it does not support role hierarchies, transitive, symmetric and inverse roles. However constant effort is made to broaden the expressivity of Description Logics handled by Knowledge Cartography. This paper is an example of such effort.
can be extended towards covering another concrete domains. In this way the possible range of use of KASeA system is substantially broadened, enabling its use in other knowledge processing systems which focus on performance of retrieving results.

The rest of the paper is organized as follows: Section 2 presents the theoretical background, Section 3 describes the method, Section 4 presents some tests results, and Section 5 concludes the paper.

2. Theoretical background

The present state-of-the-art in representing knowledge by ontologies [5] has been established with the release of OWL (Web Ontology Language) Specification [8]. OWL is a part of Semantic Web [11] initiative, which strives for development of standard technologies for knowledge management over the Web, particularly for making Web sources semantics machine-readable. OWL bases on RDF (Resource Description Framework) [10] and inherits from RDF the way of describing worlds with triples (subject, predicate, object). Two (out of three) sublanguages of OWL (especially its OWL-DL sublanguage) are also based on mathematical foundations of Description Logics.

The basic Description Logics [1] languages do not tackle with the problem of handling numerical and textual (or more generally: symbolic) values. The domain of interest (world) in Description Logics is modeled as a set of *individuals*, connected with *roles* and being members of distinguished classes of individuals called *concepts*. Relationships among concepts are expressed in the form of axioms in a *terminology*. Symbolic values can be simulated by individuals belonging to special concepts, e.g. describing some ranges of numbers, like in the following example:

\[
\text{PatientInDangerOfHypertension} \equiv \exists \text{hasBPResult}.\text{BPResultHigh}
\]  

(1)

In the example a member of the concept *PatientInDangerOfHypertension* is defined as an individual connected with role *hasBPResult* (has blood pressure test result) to a member of a concept *BPResultHigh*. The members of the latter concept are individuals simulating numbers, but, from the knowledge base point of view, there is nothing which distinguishes them from other individuals (for proper inference, there can be only one member of this concept).

However lack of more adequate support for numerical and textual values was quickly found problematic. For this reason DL introduces an extension containing the notion of a concrete domain. The definition is as follows (see [1]):

**Definition 1** A concrete domain \( D \) consists of a set \( \Delta^D \), the domain of \( D \), and a set \( \text{pred}(D) \), the predicate names over \( D \). Each predicate name \( P \in \text{pred}(D) \) is associated with an arity \( n \) and an \( n \)-ary predicate \( P^D \subseteq (\Delta^D)^n \).

[1] gives many examples of concrete domains. We show some of them in Tab. 1.

Because of reasoning problems there are some additional restrictions over the set of predicate names, making a concrete domain *admissible*.

First, the predicates have to be closed under negation, i.e. for every \( n \)-ary predicate name \( P \in \text{pred}(D) \) there is a predicate name \( Q \in \text{pred}(D) \) such that \( Q^D = (\Delta^D)^n \setminus P^D \).
Concrete Domain Predicates

<table>
<thead>
<tr>
<th>Concrete domain</th>
<th>Domain</th>
<th>Predicates</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathcal{N} )</td>
<td>( \Delta^\mathcal{N} ): The set ( \mathbb{N} ) of all nonnegative integers</td>
<td>binary predicates: ( &gt;, \leq, \geq ) unary predicates: ( &gt;_n, &lt;_n, \geq_n, \leq_n ) (( n \in \mathbb{N} ))</td>
</tr>
<tr>
<td>( \mathcal{R} )</td>
<td>( \Delta^\mathcal{R} ): The set ( \mathbb{R} ) of all real numbers</td>
<td>equalities and inequalities between polynomials with several indeterminates, e.g.: ( x + y^2 = y, x &gt; y ), etc.</td>
</tr>
<tr>
<td>( \mathcal{Z} )</td>
<td>( \Delta^\mathcal{Z} ): The set of all integer numbers</td>
<td>as above</td>
</tr>
<tr>
<td>( \mathcal{DB} )</td>
<td>( \Delta^{DB} ): The set of atomic values occurring in a relational database DB</td>
<td>Relations defined over DB using a query language (e.g. SQL)</td>
</tr>
</tbody>
</table>

Table 1. Examples of concrete domains

Second, for any finite conjunction of the form:

\[
P_1(x^{(1)}) \land P_2(x^{(2)}) \land \cdots \land P_k(x^{(k)})
\]

(2)

(where for \( 1 \leq i \leq k \): \( P_i \) is a predicate name of arity \( n_i \) in \( \text{pred}(\mathcal{D}) \) and \( x^{(i)} \) is \( n_i \)-tuple of variables) should be decidable whether it is satisfiable, i.e. if there exists an assignment of elements of \( \Delta^\mathcal{D} \) to the variables of all tuples such that the conjunction becomes true in \( \mathcal{D} \).

The restrictions mentioned above are important for proving properties of reasoning systems. From among concrete domains in Tab. 1, only \( \mathcal{Z} \) is not admissible, which is a consequence of undecidability of Hilbert’s 10th problem.

The usage of concrete-domains in DL languages is based on a special kind of concept constructors called agreement constructors. The language extension, named \( \mathcal{ALC}(\mathcal{D}) \), allows to declare new concepts using the form:

\[
\exists(u_1, \ldots, u_n).P
\]

(3)

where for \( 1 \leq i \leq n \) \( P \) is an \( n \)-ary predicate name in \( \text{pred}(\mathcal{D}) \) and \( u_i \) is a chain of roles.

In (3), the last role in the chain is an attribute, i.e. it may have only concrete individuals as fillers. An example of an attribute may be \( \text{hasAge} \) indicating an age of an individual in years. Then the concept \( \exists \text{hasAge}.<_{15} \) would describe all individuals younger than 15 years and all parents of children younger than 15 would be described by the concept \( \exists(\text{hasChild}, \text{hasAge}).<_{15} \).

Attributes are clearly distinguished from remaining roles, since any abstract and a given concrete domain have to be disjoint:

\[
\Delta^\mathcal{I} \cap \Delta^\mathcal{D} = \emptyset
\]

(4)
OWL itself takes advantage of the notion of attributes, which are in OWL called datatype properties, and are basically special roles that connect objects (abstract individuals) with values. A range of an attribute can be set to one of the types in XML Schemas. OWL, however, lacks of predicates that can be used to infer from concrete domains.

We decided to base our work on DIG [2], the language created by Description Logics Implementation Group, and more specifically on the extension of DIG developed at Gdansk University of Technology, called DIGUT [7]. DIGUT allows for use of attributes with range of types embracing integers, strings, float numbers, time and date expressions, and Boolean values. The set of possible predicates is restricted and for every concrete-domain embraces unary predicates: \( \geq a, \leq a, = a \), where \( a \) is a constant value belonging to the adequate domain\(^2\). However limited, the range of concrete-domain operators that may be used in DIGUT outdoes OWL and can be used to describe many interesting (from practical point of view) relationships among concepts and attributes values (for example, ranges of vital parameters that suggest occurrence of some pathologies can be distinguished). Although we focused on the range of constructors offered by DIGUT, the methods described here can be extended to cover broader range of concrete domains and constructors.

3. Concrete Domains in KASEA

3.1. Knowledge Cartography in a nutshell

KA\(S\)E\(A\) inference engine exploits inference methods based on the Cartographic Approach [3][4] developed at Gdansk University of Technology. The Knowledge Cartography takes its name after a map of concepts. A map of concepts is basically a description of interrelationships between concepts in a terminology. The map is created in the course of Knowledge Base creation. A map of concepts can be graphically represented in a form similar to a Venn diagram (Fig. 2). Each atomic region (i.e. a region that does not contain any other region) represents a unique valid intersection of base concepts.

Because any area in the map consists of some number of regions, any area can be represented by a

\(^2\)This limitation makes all DIGUT concrete domains (including integers) admissible.
Figure 3. A map of concepts with concepts of the form $\exists R.C$

string (array) of binary digits (bits) of length $n$ (where $n$ is the number of regions in the map) with “1”s at positions corresponding to contained regions and “0”s elsewhere. According to this rule, a concept $C$ from a terminology placed in the map is assigned a signature $s(C)$ that is a string of bits representing the area covered by the concept in the map. In the map of concepts there are placed atomic concepts (i.e. not defined by means of other concepts) and those concepts of the form $\exists R.C$ that are explicitly used in the terminology.\(^3\)

Signatures of complex concepts constructed by means of intersection, union and complement operators can be derived by performing boolean operations (AND, OR and NOT, denoted by $\land$, $\lor$, $\neg$) over elements of the arrays. Signatures for intersections can be calculated as follows $s_i(C \cap D) = s_i(C) \land s_i(D)$ (where $s_i$ denotes the $i$-th element of an array; the whole operation can be denoted briefly as $s(C \cap D) = s(C) \land s(D)$). Signatures for unions and complements are calculated respectively as $s_i(C \cup D) = s_i(C) \lor s_i(D)$ (i.e. $s(C \cup D) = s(C) \lor s(D)$) and as $s_i(\neg C) = \neg s_i(C)$ (i.e. $s(\neg C) = \sim s(C)$) (see Fig. 2 for an example of a signature of a union). This gives us a potential of calculating signature of any ALC concept explicitly specified in the terminology.

This shift from a field of symbolic concepts to a field of signatures gives us flexibility of performing operations that might be complicated to carry out in symbolic domain. For example, the subsumption problem (“Is a concept $C$ subsumed by a concept $D$?”) can be expressed as a simple comparison of signatures to check whether the signature of $C$ does not contain “1” in any position where the signature of $D$ contains “0” (such comparison we denote as $s(C) \leq s(D)$).

Analogical techniques as for TBox can be applied to reasoning over individuals. We assign each individual $a$ the signature of the most specific concept the individual is an instance of. We denote this concept as $C_a$. Actually, this concept need not be defined explicitly in TBox; the signature for this concept is built dynamically when new assertions are added to ABox (e.g. handling of $D(a)$ assertion triggers the operation $s(C_a) := s(C_a) \land s(D)$). In this way we can reduce all ABox reasoning problems to TBox reasoning problems, which in turn can be solved by operations on signatures. For example, checking if an individual $a$ is an instance of a concept $C$ is reduced to checking whether $C_a$ is subsumed by $C$.

In the process of calculating of a signature of an individual the concepts of the form $\exists R.C$ that are included in the terminology are of great importance. Let us focus on an exemplary map of concepts depicted in Fig. 3.

The information about relationships between a pair of concepts $\exists R.C$ and $C$ (depicted in the figure as block arrows) is held in the system in the form of tuples $(R, s_{\exists R.C}, s_C)$ where $R$ is the role name,

\(^3\)And concepts of the form $\forall R.C$ after conversion to the equivalent form of $\neg \exists R.\neg C$.\(^6\)
and $s_{\exists R.C}$ and $s_C$ are signatures of concepts $\exists R.C$ and $C$ respectively. For the terminology from Fig. 3 the tuples will be: $(\text{hasBPResult}, 01100, 00011)$ and $(\text{hasBPResult}, 00100, 00001)$.

Storing the tuples allows for performing reasoning about membership of individuals to concepts on the basis of instances of roles between them. An example of such reasoning is presented in Fig. 4. We can see that on the basis of the signature of the individual $\text{BPResult1}$ individual $\text{JohnSmith}$ (originally with signature 11111, which corresponds to knowing nothing about him except that he exists) has been identified as a member of the concept $\exists \text{hasBPResult}.\text{BPResultHigh}$.

If we assume that in the terminology it is stated that concepts $\exists \text{hasBPResult}.\text{BPResultHigh}$ and $\text{PatientInDangerOfHypertension}$ are equivalent, this process corresponds to making diagnosis on the basis of results of examinations.

### 3.2. Extension of Cartographic methods towards concrete-domains

To enable KASEA to execute new kind of inferences over concrete domains, the set of new methods have been developed. The map of concepts is extended by a part describing the general concrete domain ($\top_D$). In this part, instead of concepts, sets of numbers and symbols satisfying specific unary predicates are placed. An example of such extended map of concepts is shown in Fig. 5.

In the process of creation of a map of concepts concrete-domain unary predicates used in the terminology being processed are placed in the map. Interrelationships among these sets are calculated by KASEA and provided to the map-creation algorithm in the form of additional axioms.

Attributes are treated exactly as roles with a domain contained in abstract domain ($\top$) and with a range contained in concrete domain ($\top_D$). All the methods of handling roles are then valid for attributes.
The triples are held for each concept of the form \( \exists A.C \) where \( A \) is an attribute name. The example from Fig. 4 has been rewritten in Figure 6; \( \text{hasBPResult} \) is an attribute at this time. The triples will be \((\text{hasBPResult}, 011000, 000011)\) and \((\text{hasBPResult}, 001000, 000001)\).

The mechanism for inference from ABox works almost exactly the same as the one shown in Fig. 4. The only difference is that a special interpretation mechanism assigns each concrete-domain element to a region corresponding to all predicates that are satisfied for this element, so no additional information about it is needed. Figure 7 shows the process.

### 3.3. Case Study and algorithms

In this section we will present in more detail behavior of the system on the basis of a simple example.

At the moment of creation of a knowledge base, the system creates some predefined subset concepts to represent the range of attribute types assumed in DIGUT language. The list of the types and their predefined signatures are shown in Tab. 2. All the subconcepts of concepts describing whole concrete domains (like strings, floats, integers, etc.) have a predicate associated with them that describes the range of attribute values being members of the concept. A special unnamed concept \( \text{Zero} \) is added to reflect the intersection between non positive and non negative integers.

The first scenario describes the process of creating a new concrete-domain concept by the user. It is executed every time after reaching a tag describing a concrete-domain concept in a Tell request during constructing a T-Box.

Let us assume that a DL axiom to be processed is as follows:

\[
\text{HighSchoolStudent} \equiv \text{Student} \sqcap \exists \text{hasAge.} \geq 15 \land \leq 18
\]
Table 2. Predefined concrete-domain concepts in KAESA and concrete-domain part of their signatures

<table>
<thead>
<tr>
<th>DIGUT attribute type</th>
<th>Predicate</th>
<th>Signature section</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td></td>
<td>100000000</td>
</tr>
<tr>
<td>Boolean</td>
<td></td>
<td>010000000</td>
</tr>
<tr>
<td>Time</td>
<td></td>
<td>001000000</td>
</tr>
<tr>
<td>Date</td>
<td></td>
<td>000100000</td>
</tr>
<tr>
<td>DateTime</td>
<td></td>
<td>000010000</td>
</tr>
<tr>
<td>Float</td>
<td></td>
<td>000001000</td>
</tr>
<tr>
<td>Integer</td>
<td></td>
<td>000000111</td>
</tr>
<tr>
<td>PositiveInteger</td>
<td>$x &gt; 0$</td>
<td>000000011</td>
</tr>
<tr>
<td>NonPositiveInteger</td>
<td>$x \leq 0$</td>
<td>000000011</td>
</tr>
<tr>
<td>NegativeInteger</td>
<td>$x &lt; 0$</td>
<td>000000001</td>
</tr>
<tr>
<td>NonNegativeInteger</td>
<td>$x \geq 0$</td>
<td>000000110</td>
</tr>
<tr>
<td>(Zero)</td>
<td>$x = 0$</td>
<td>000000010</td>
</tr>
</tbody>
</table>

The unary predicate $P = \geq_{15} \land \leq_{18}$ has a determined proper concrete-domain of $\text{Integer}^4$. The concept covering the set of attribute values satisfying the predicate $P$ will be created. Next steps of the scenario would be as follows:

1. Find all the concepts $(I_1, \ldots, I_n)$ with the signature smaller than the signature of the concept $\text{Integer}$.
2. For every found concept $I_k$ (where $1 \leq k \leq n$) check the predicate associated with it, $P_k$, to determine the interpretation.
3. If there exists $I_k$ such that $(P_x)^I = (P_k)^I$ end the procedure.
4. If not, create a new concept $I$ such that: $I \sqsubseteq \text{Integer}$ and associate $P$ with $I$.
5. Examine all found predicates again (for $k \in \{1 \ldots n\}$), and do one of the following:
6. If there is an inclusion $(P_x)^I \subseteq (P_k)^I$ or $(P_x)^I \supseteq (P_k)^I$ then add the axiom $I \sqsubseteq I_k$ or $I \sqsupseteq I_k$ respectively.
7. If both interpretations are disjoint, $(P_x)^I \cap (P_k)^I = \emptyset$, then add the axiom $I \sqcap I_k \equiv \bot$.
8. If $(P_x)^I \cup (P_k)^I = \text{Integer}^I$ then add the axiom $I \sqcup I_k \equiv \text{Integer}$.

The algorithm sketched above is very general and can be used for every concrete domain and any unary predicate. The problem is decidable on the basis of the assumption that requires the concrete domain and sets of predicates to be admissible. In the case when all concrete-domain concepts describe ranges, performance of the algorithm can be substantially increased.

As a result of these steps the Cartographer will calculate correct signatures for every subset concepts of the concrete-domain. If our example is the first definition of a concrete-domain, the new signatures for all integer concepts will be as shown in Tab. 3.

---

4We know the concrete-domain of the predicate because DIGUT requires an ontology author to specify the type of an attribute used in such axioms
### Table 3. Signatures of concrete-domain concepts after recalculation

<table>
<thead>
<tr>
<th>DIGUT attribute type</th>
<th>Predicate</th>
<th>Signature section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td></td>
<td>0000001111</td>
</tr>
<tr>
<td>PositiveInteger</td>
<td>(x &gt; 0)</td>
<td>0000001001</td>
</tr>
<tr>
<td>NonPositiveInteger</td>
<td>(x \leq 0)</td>
<td>0000000110</td>
</tr>
<tr>
<td>NegativeInteger</td>
<td>(x &lt; 0)</td>
<td>0000000010</td>
</tr>
<tr>
<td>NonNegativeInteger</td>
<td>(x \geq 0)</td>
<td>0000001101</td>
</tr>
<tr>
<td>(Zero)</td>
<td>(x = 0)</td>
<td>0000000100</td>
</tr>
<tr>
<td>((\text{geq}<em>{15} \land \leq</em>{18}))</td>
<td>(x \geq 15) and (x \leq 15)</td>
<td>0000000100</td>
</tr>
</tbody>
</table>

The second scenario shows how assertions are analyzed during the creation of an A-Box. As an example let us consider a binary assertion \(\text{hasAge}(JohnSmith, 17)\).

Let us presume that we already know that the individual \(JohnSmith\) belongs to the concept \(\text{Student}\) and that the value of 17 has not been used until now. After processing this assertion the individual should get the signature of the concept \(\text{HighSchoolStudent}\). To obtain this the following steps are performed:

1. Find the attribute definition to determine the type of the attribute. From the previous example we know that this is integer.
2. Find the signature \(s\) of the concrete concept \(\text{Integer}\).
3. Find all the concepts \((I_1, \ldots, I_n)\) with the signature smaller then \(s\).
4. For every found concept \(I_k\) (where \(1 \leq k \leq n\)) find the smallest signature \(s_x\) neglecting the concepts where the expression \(P_k\) in the predicate field returns false for \(x = 17\). The proper answer would be the signature of the concept \(I_x\) described by the predicate \(\text{“}x \geq 15\text{ and }x \leq 18\text{”}\), as we remember from the previous example.
5. Create an individual (an instance of the class \(\text{AttributeIndividual}\)) for the value of 17 and assign it the signature \(s_x\).
6. Create an instance of the \(\text{AttributeRole}\) class and connect it to the individual \(JohnSmith\) and to the individual “17”.
7. Perform the role checking (neighborhood update, see [3]), i.e.: for all defined concepts of the form \(\exists\text{hasAge}.I_k\) if the signature of \(I_k\) is equal to or bigger than the signature \(s_x\), combine the signature of the individual \(JohnSmith\) with the signature of \(\exists\text{hasAge}.I_k\) using binary AND operation; if the signature of the individual \(JohnSmith\) has been changed continue role checking for its neighborhood (i.e. individuals connected to \(JohnSmith\) with an instance of any role).

The last step causes that the signature of the individual \(JohnSmith\) changes. As we have presumed this individual had been assigned to the concept \(\text{Student}\). The concept \(\exists\text{hasAge}.I\) is processed in the step 7 and when we combine the signature of the individual \(JohnSmith\) with the signature of \(\exists\text{hasAge}.I\), it will change into the signature of the concept \(\text{HighSchoolStudent}\); the result we wanted to attain.
### Table 4. Time of execution of various retrieval queries

<table>
<thead>
<tr>
<th>Kind of retrieval query</th>
<th>Execution time [ms] (Proposed method)</th>
<th>Execution time [ms] (Database)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Searching for all attributes of an individual</td>
<td>5.3</td>
<td>4.9</td>
</tr>
<tr>
<td>Searching for all names of individuals with a specific value of a specific attribute</td>
<td>4.1</td>
<td>4.1</td>
</tr>
<tr>
<td>Searching for all names of individuals with a value of a specific attribute in a given range</td>
<td>5.0</td>
<td>5.0</td>
</tr>
<tr>
<td>Searching for a value of the given attribute for the given individual</td>
<td>4.1</td>
<td>4.2</td>
</tr>
</tbody>
</table>

4. **Assessment of the method**

The method described allows for storing numerical and textual values of attributes in the knowledge base, enabling it to perform functions traditionally reserved for databases, and extending it by reasoning capabilities. The method has been implemented in KASea 2.0.

The performance of the solution was tested against the alternative one in which the attributes were stored in relational tables and bound with individuals by a database based mechanism capable only of storing and retrieving of attribute values, but not for reasoning. After loading the knowledge base and the database with individuals various retrieval tests have been performed. The results for an exemplary base holding 20000 individuals/300 attributes definitions/200000 attribute values are shown in Tab. 4. The tests were performed on a computer with the Intel Celeron 2.4GHz CPU and 512MB RAM. All the data were stored in the Oracle database version 9i. The server worked in the stand-alone mode.

The results show that the presented solution works only up to 10% slower that the one focused only on attribute values storage and retrieval; the proportion was the same for various sizes of the base. The performance of reasoning in the presented solution was undisrupted by introduction of attribute handling procedures and the scalability of the system was maintained (the system is likely to reach sub-linear scalability as it was shown in [3]).

The method itself can be relatively easily extended to cover broader range of unary predicates and predefined concrete domains. The research on introducing more complex predicates and user-defined concrete domains is underway. The basic method would be exactly the same as the one described, however efficient mechanisms of estimating interrelationships among \( n \)-ary predicates have to be developed.

Compared to other Description Logics reasoners wrt. concrete domains KASea 2.0 allows to achieve level of functionality exceeding the one offered by applications strictly following OWL, like Jena [6]. RACER [9], however, allows for use of more predicates in describing concepts (although it does not allow for defining custom concrete domains). The difference between KASea 2.0 and RACER is that RACER performs reasoning at the time of answering retrieval queries while KASea 2.0 preserves
the feature of Knowledge Cartography and performs the process of reasoning during data insertion and stores the results. Experiments show that in a specific class of knowledge bases the solution used in KASEA can greatly shorten the time of processing especially for knowledge base with more than 1000 individuals (see [3] or [4]). Moreover the proposed method can be extended to handling more complex predicates.

5. Summary

The proposed solution offers the way of broadening functionality of KASEA to turn it into a self contained subsystem which can be used in various application which, in order to offer the required quality level, need to efficiently perform reasoning over information acquired from Internet sources. The cartographic method offers distinctive capabilities of storing conclusions along with data explicitly inserted into the knowledge base, thus is especially useful for systems that focus on performance of query answering and scalability in terms of the number of assertions.

References


