Proposal of a New Rule-based Inference Scheme for the Semantic Web Applications[⋆]

Grzegorz J. Nalepa[1] and Weronika T. Furmańska[1]

Institute of Automatics,
AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
gjn@agh.edu.pl, wtf@agh.edu.pl

Abstract. The current challenge of the Semantic Web is the development of an expressive yet effective rule language. In this paper an overview of an integration proposal for Description Logics (DL) and Attributive Logics (ALSV) is presented. These two formalisms stem from the field of Knowledge Representation and Artificial Intelligence, and provide different description and reasoning capabilities. The contribution of the paper consists in introducing a possible transition from ALSV to DL. This opens up possibilities of using well-founded expert systems modeling languages to improve the design of Semantic Web rules.

1 Introduction

The Semantic Web proposal of the next generation Web with rich semantics and automated inference is based on number of formal concepts and practical technologies. This includes Description Logics (DL) [1] as the formalism describing formal ontologies. Currently, the development is focused on providing a flexible rule language for the Web. It should be RIF-compatible [1] on the rule interchange level, and conceptually compatible with ontologies described in OWL with use of Description Logics. The SWRL proposal recently submitted to W3C aims at meeting these requirements.

The Semantic Web initiative is based of previous experiences and research of Knowledge Engineering [2] in the field of Artificial Intelligence [3]. In this field the rule-based expert systems technologies are a prime example of effective reasoning systems based on the rule-based paradigm [4]. The formal description of these systems is based on the propositional calculus, or restricted predicate logic – it is worth considering how the Semantic Web community could benefit from some classic expert systems tools and solutions.

A recent proposal of a new logical calculus for rules aims at extending the expressiveness of the rule language, by introducing an attributive language [5,6]. This solution seems superior to the simple propositional systems, and easier to

⋆ The paper is supported by the HeKatE Project funded from 2007–2009 resources for science as a research project.

1 See http://www.w3.org/2005/rules/wiki/RIF_Working_Group.

reason with than the predicate logic. The XTT^2 rule language is based on this solution [6]. It provides visual design and formal analysis methods for decision rules. This would eventually allow to design rules for the Semantic Web. The current problem with Attributive Logic with Set Values over Finite Domain (ALSV(FD)) is the lack of conceptual compatibility with DL.

In this paper selected important DL concept are discussed in Sect. 2, and a brief introduction to ALSV(FD) is given in Sect. 3. This gives a motivation presented in Sect. 4 for the research aiming at translation from the ALSV(FD), to DL which provides a formalized foundation for ontologies and SWRL. Possible integration approaches of these two calculi are discussed in Sect. 5, using a simple example in Sect. 6. The paper ends with future work in Sect. 7.

2 Description Logics Overview

Description Logics are a family of knowledge representation languages [1]. Historically related to semantic networks and frame languages, they describe the world of interest by means of concepts, individuals and roles. However, contrary to their predecessors, they provide a formal semantics and thus enable for automated reasoning. Basic Description Logics take advantage of their relation to predicate calculus. On one hand they adopt its semantics, which makes them more expressive than a propositional logic. On the other, by restricting the syntax to formulae with maximum two variables, they remain decidable and more human-readable. These features have made Description Logics a popular formalism used for designing ontologies for the Semantic Web. There exist a number of DL languages. They are defined and distinguished by allowed concept descriptions, which influences the languages' expressivity.

The building blocks of vocabulary in DL languages are *concepts*, which denote sets of individuals and *roles* which denote the binary relations between individuals. Elementary descriptions in DL are *atomic concepts* and *atomic roles*. More complex descriptions can be built inductively from them using *concept constructors*. Respective DL languages are distinguished by the constructors they provide. A minimal language of practical interest is the *Attributive Language*.

Definition 1. *Let A denote an atomic concept and R an atomic role. In basic $\mathcal{AL}$ concept descriptions C and D can be formed according to the following rules:*

$$C, D \rightarrow A| \qquad \textit{atomic concept} \qquad (1)$$
$$\top| \qquad \textit{universal concept} \qquad (2)$$
$$\bot| \qquad \textit{bottom concept} \qquad (3)$$
$$\neg A| \qquad \textit{atomic negation} \qquad (4)$$
$$C \sqcap D| \qquad \textit{intersection} \qquad (5)$$
$$\forall R.C| \qquad \textit{value restriction} \qquad (6)$$
$$\exists R.\top \qquad \textit{limited existential quantification} \qquad (7)$$

In order to define a formal semantics, an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ is considered. The interpretation consists of the *domain of interpretation* which is a non-empty set and an interpretation function, which to every atomic concept A assigns a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and for every atomic role R a binary relation $R^{\mathcal{I}} = R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended over concept descriptions by the Definition 2.

Definition 2.

$$\top^{\mathcal{I}} = \Delta^{\mathcal{I}} \tag{8}$$

$$\bot^{\mathcal{I}} = \emptyset \tag{9}$$

$$(\neg A)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus A^{\mathcal{I}} \tag{10}$$

$$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}} \tag{11}$$

$$(\forall R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} | \forall b, (a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \tag{12}$$

$$(\exists R.\top)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} | \exists b, (a,b) \in R^{\mathcal{I}}\} \tag{13}$$

Basic DL allow only atomic roles (i.e. role names) in role descriptions.

The basic language can be extended by allowing other concept constructors, such as *union ($\mathcal{U}$)* , *full negation ($\mathcal{C}$)*, *full existential quantification ($\mathcal{E}$)* or *number restriction ($\mathcal{N}$)*. Resulting formalisms are called using the letters indicating the allowed constructors, e.g. $\mathcal{ALC}$, $\mathcal{ALCN}$, $\mathcal{ALUE}$ etc. The smallest propositionally closed language is $\mathcal{ALC}$.

Different extensions to basic Description Logics are introduced by allowing *role constructors*. They enable for introducing various constraints and properties of roles, such as *transitive closure*, *intersection*, *composition* and *union*, or *complement* and *inverse* roles. Another kind of extension is obtained by allowing *nominals* in concept definitions and introducing primitive datatypes. These modifications proved to be extremely valuable and important in the context of the Semantic Web and ontologies. However, they are sources of high computational complexity of reasoning in the resulting ontologies.

For expressive DLs the abovementioned naming convention would be too long. Hence, for the basic $\mathcal{ALC}$ language extended with transitive roles, $\mathcal{S}$ is often used. The letter $\mathcal{H}$ is used to represent role hierarchy, $\mathcal{O}$ to indicate nominals in concept descriptions, $\mathcal{I}$ represents inverse roles, $\mathcal{N}$ number restrictions, and (**D**) indicates the integration of some concrete domain/datatypes. The DL underlying OWL-DL language includes all of those constructs and is therefore called $\mathcal{SHOIN}(\mathbf{D})$.

Description Logics provide tools to build a knowledge base and to reason over it. The knowledge base consists of two parts, namely TBox and ABox.

TBox provides a terminology and contains a taxonomy expressed in a form of set of axioms. The axioms define concepts, specify relations between them and introduce set constraints. Therefore, TBox stores implicit knowledge about sets of individuals in the world of interest. Formally, a terminology $\mathcal{T}$ is a finite set of terminological axioms. If C and D denote concept names, and R and S role names, then the terminological axioms may be in two forms: $C \sqsubseteq D$ ($R \sqsubseteq S$)

or $C \equiv D$ ($R \equiv S$). Equalities that have an atomic concept on the left-hand side are called *definitions*. Axioms of the form $C \sqsubseteq D$ are called *specialization* statements. Equalities express necessary and sufficient conditions, whereas specialization statements specify constraints (necessary conditions) only. An interpretation (function) $\mathcal{I}$ maps each concept name to a subset of the domain. The interpretation satisfies an axiom $C \sqsubseteq D$ if: $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. It satisfies a concept definition $C \equiv D$ if: $C^{\mathcal{I}} = D^{\mathcal{I}}$. If the interpretation satisfies all the definitions and all axioms in $\mathcal{T}$, it satisfies the terminology $\mathcal{T}$ and is called a *model* of $\mathcal{T}$.

ABox contains explicit assertions about individuals in the conceived world. They represent extensional knowledge about the domain of interest. Statements in ABox may be: concept assertions, e.g. $C(a)$ or role assertions, $R(b, c)$. An interpretation $\mathcal{I}$ maps each individual name to an element in the domain. With regards to terminology $\mathcal{T}$ the interpretation satisfies a concept assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and a role assertion $R(b, c)$ if $\langle b^{\mathcal{I}}, c^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$. If it satisfies all assertions in ABox $\mathcal{A}$, then it satisfies $\mathcal{A}$ and $\mathcal{I}$ is a model of $\mathcal{A}$.

Although terminology and world description share the same model-theoretic semantics, it is convenient to distinguish these two parts while designing a knowledge base or stating particular inference tasks.

With regards to terminology $\mathcal{T}$ one can pose a question if a concept is *satisfiable*, if one concept *subsumes* another, if two concepts are *equivalent* or *disjoint*. A concept C is satisfiable with respect to $\mathcal{T}$ if there exists a model (an interpretation) $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is not empty. A concept C is subsumed by a concept D w.r.t. $\mathcal{T}$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$. Two concepts C and D are equivalent w.r.t. T if $C^{\mathcal{I}} = D^{\mathcal{I}}$ for every model I of T. Finally, two concepts C and D are disjoint w.r.t. $\mathcal{T}$ if $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for every model $\mathcal{I}$ of $\mathcal{T}$.

Satisfiability and subsumption checking are the main reasoning tasks for TBox; other can be reduced to them, and either can be reduced to the other.

For ABox there are four main inference tasks: *consistency checking*, *instance checking*, *realization* and *retrieval*. An ABox $\mathcal{A}$ is consistent w.r.t. a TBox $\mathcal{T}$, if there is an interpretation that is a model of both $\mathcal{A}$ and $\mathcal{T}$. Furthermore, we say that an ABox is consistent, if it is consistent w.r.t. the empty TBox. One can acquire an empty TBox by expanding concepts in (acyclic) TBox which means replacing concepts with the right-hand side of their definitions. Instance checking tests if a given assertion is entailed by the ABox. Realization tasks consist in finding the most specific concept for a given individual and retrieval returns individuals which are instances of a given concept.

All these tasks can be reduced to *consistency checking* of the ABox with respect to the TBox.

3 Attributive Logics Concepts

Knowledge representation based on the concept of *attributes* is one of the most common approaches [2]. It is one of the foundations for relational databases, attributive decision tables and trees [7], as well as rule-based systems [5].

A typical way of thinking about attributive logic for knowledge specification may be put as follows. Knowledge is represented by *facts* and *rules*. A fact could be written as $A := d$ or $A(o) := d$, where A is a certain attribute (property of an object), o an object of interest and d is the attribute value. Facts are interpreted as propositional logic atomic formulae. This basic approach is sometimes extended with use of certain syntax modifications [7]. On top of these facts simple decision rules are built, corresponding to conditional statements.

In a recent book [5] the discussion of attributive logic is extended by allowing attributes to take *set values* and providing some formal framework of the *Set Attributive Logic* (SAL). The basic idea for further discussion is that attributes should be able to take not only *atomic* but *set* values as well, written as $A(o) = V$, where V is a certain set of values.

In [6] the language of SAL has been extended to provide an effective knowledge representation tool for decision rules, where the state description uses finite domains. The proposed calculus *Attribute Logic with Set Values over Finite Domains* (ALSV(FD)) is proposed to simplify the decision rules formulation for classic rule-based expert system, including, business rules, using the so-called XTT^2 rule language. While being a general solution, the language is oriented towards forward chaining intelligent control systems.

Here, an extended notation for ALSV(FD) is introduced, including: 1) the explicit differentiation of equality relation, denoted as $=$ used for comparison in the ALSV(FD) formulas and the fact definition operator, denoted as $:=$, and 2) the formalization of constants that simplify effective formula notation.

The basic element of the language of *Attribute Logic with Set Values over Finite Domains* (ALSV(FD) for short) are: attribute names and attribute values. $\mathbf{A}$ – a finite set of attribute names, $\mathbf{D}$ – a set of possible attribute values (their *domains*), and $\mathbf{C}$ – a set of constants.

Let $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$ be all the attributes such that their values define the state of the system under consideration. It is assumed that the overall domain $\mathbf{D}$ is divided into n sets (disjoint or not), $\mathbf{D} = D_1 \cup D_2 \cup \ldots \cup D_n$, where D_i is a domain related to attribute A_i, $i = 1, 2, \ldots, n$. Any domain D_i is assumed to be a finite (discrete) set.

We also consider a finite set of constants $\mathbf{C} = \{C_1, C_2, \ldots, C_m\}$. Every constant C_j $j = 1, 2, \ldots, m$ is related to a single domain D_i, $i = 1, 2, \ldots, n$ such that $C_j \subset D_i$ There might be multiple constants related to the same domain.

Definition 3. ***Attributes*** *As we consider dynamic systems, the values of attributes can change over time (or state of the system). We consider both simple attributes of the form $A_i \colon T \to D_i$ (i.e. taking a single value at any instant of time) and generalized ones of the form $A_i \colon T \to 2^{D_i}$ (i.e. taking a set of values at a time); here T denotes the time domain of discourse.*

Let A_i be an attribute of $\mathbf{A}$ and D_i the domain related to it. Let V_i denote an arbitrary subset of D_i and let $d \in D_i$ be a single element of the domain. V_i can also be a constant related to D_i.

The legal atomic formulae of ALSV along with their semantics are presented in Def. 4 for simple and in Def. 5 for generalized attributes.

Definition 4. ***Simple attribute formulas syntax***

$$A_i = d \qquad \textit{the value is precisely defined} \tag{14}$$

$$A_i \in V_i \qquad \textit{the current value of } A_i \textit{ belongs to } V_i \tag{15}$$

$$A_i \neq d \qquad \textit{shorthand for } A_i \in D_i \setminus \{d\} \tag{16}$$

$$A_i \notin V_i \qquad \textit{is a shorthand for } A_i \in D_i \setminus V_i \tag{17}$$

Definition 5. ***Generalized attribute formulas syntax***

$$A_i = V_i \qquad \textit{equals to } V_i \textit{ (and nothing more)} \tag{18}$$

$$A_i \neq V_i \qquad \textit{is different from } V_i \textit{ (at at least one element)} \tag{19}$$

$$A_i \subseteq V_i \qquad \textit{is a subset of } V_i \tag{20}$$

$$A_i \supseteq V_i \qquad \textit{is a superset of } V_i \tag{21}$$

$$A \sim V \qquad \textit{has a non-empty intersection with } V_i \tag{22}$$

$$A_i \not\sim V_i \qquad \textit{has an empty intersection with } V_i \tag{23}$$

In case V_i is an empty set (the attribute takes in fact no value) we shall write $A_i = \{\}$. In case the value of A_i is unspecified we shall write $A_i = \texttt{NULL}$ (a database convention). If we do not care about the current value of the attribute we shall write $A = \_$ (a PROLOG convention). More complex formulae can be constructed with *conjunction* ($\wedge$) and *disjunction* ($\vee$); both symbols have classical meaning and interpretation.

There is no explicit use of negation. The proposed set of relations is selected for convenience and as such they are not completely independent. Various notational conventions extending the basic notation can be used. For example, in case of domains being ordered sets, relational symbols such as $>$, $>=$, $<$, $=<$ can be used with the straightforward meaning.

The semantics of the proposed language is presented below in an informal way. The semantics of $A = V$ is basically the same as the one of SAL [5]. If $V = \{d_1, d_2, \ldots, d_k\}$ then $A = V$ if the attribute takes all the values specified with V (and nothing more). The semantics of $A \subseteq V$, $A \supseteq V$ and $A \sim V$ is defined as follows:

- $A \subseteq V \equiv A = U$ for some U such that $U \subseteq V$, i.e. A takes *some* of the values from V (and nothing out of V),
- $A \supseteq V \equiv A = W$, for some W such that $V \subseteq W$, i.e. A takes *all* of the values from V (and perhaps some more), and
- $A \sim V \equiv A = X$, for some X such that $V \cap X \neq \emptyset$, i.e. A takes *some* of the values from V (and perhaps some more).

As it can be seen, the semantics of ALSV is defined by means of relaxation of logic to simple set algebra.

To simplify the formula notation *constants* can also be defined as: $C := V$. A knowledge base in ALSV(FD) is composed of simple formulas forming rules and facts defining the current systems state.

The current values of all attributes are specified in the contents of the knowledge-base. From logical point of view the state is represented as a logical formula of the form:

$$(A_1 := S_1) \wedge (A_2 := S_2) \wedge \ldots \wedge (A_n := S_n) \qquad (24)$$

where $S_i := d_i$ $(d_i \in D_i)$ for simple attributes and $S_i := V_i$, $(V_i \subseteq D_i)$ for complex ones. In order to cover realistic cases an explicit notation for covering unspecified, unknown values is proposed; for example to deal with the data containing the NULL values imported from a database.

Consider a set of n attributes $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$. Any XTT^2 rule is assumed to be of the form:

$$(A_1 \propto_1 V_1) \wedge (A_2 \propto_2 V_2) \wedge \ldots (A_n \propto_n V_n) \longrightarrow \mathit{RHS}$$

where $\propto_i$ is one of the admissible relational symbols in ALSV(FD) (see legal formulas in Def. 4,5), and *RHS* is the right-hand side of the rule covering conclusion, including state transition and actions, for details see [5]. In order to fire a rule the preconditions have to be satisfied. The satisfaction of rule preconditions is verified in an algebraic mode, using the rules specified in Fig. 1 The full discussion of ALSV inference for all formulas can be found in [6].

To summarize, main assumptions about the XTT^2 representation are:

- current system *state* is explicitly represented by a set of *facts* based on attributive representation,
- facts are denoted in ALSV(FD) as: $A := V$,
- constants are facts defined as: $C := V$ to simplify compact rule notation,
- system dynamics (state transition) is modelled with *rules*,
- the conditional part of the XTT^2 rule is conjunction built from simple ALSV(FD) formulas,
- rule decision contains state modification – new facts, and optionally external actions execution statements,
- to fire the rule, the condition has to be satisfied, which means that certain *inference task* in ALSV(FD) has to be solved.

$$\begin{array}{l}
A = d_i, d_i = d_j \models A = d_j \\
A = d_i, d_i \neq d_j \models A \neq d_j \\
A = d_i, d_i \in V_j \models A \in V_j \\
A = d_i, d_i \notin V_j \models A \notin V_j
\end{array}
\qquad
\begin{array}{l}
A = V, V = W \models A = W \\
A = V, V \neq W \models A \neq W \\
A = V, V \subseteq W \models A \subseteq W \\
A = V, V \supseteq W \models A \supseteq W \\
A = V, V \cap W \neq \emptyset \models A \sim W \\
A = V, V \cap W = \emptyset \models A \not\sim W
\end{array}$$

Fig. 1. Inference rules

4 Motivation

Description Logics provide an effective formal foundation for the Semantic Web application based on ontologies described with OWL. They allow for simple inference tasks, e.g. corresponding to concept classification. Currently the main challenge for DL is the rule formulation.

Rules are the next layer in the Semantic Web stack that has to be provided in order to make the Semantic Web applications operate on the knowledge expressed in ontologies. There are several approaches to the integration of rules and ontologies. An example of a homogeneous approach are Description Logic Programs (DLP) [8]. A heterogeneous one can be found in the design of the Semantic Web Rule Language (SWRL) [9]. Other problems, that so-far have rarely been considered in the Semantic Web research include effective design methodologies for large rule bases, as well as knowledge quality issues. To maintain larger knowledge bases is not a trivial task; prone to design errors an logical misformulations. This is why scalable design methods have to be considered.

In the classic AI field of rule-based expert systems numerous solutions for rule formulation and inference, as well as the design and analysis are considered [4,5]. Number of visual knowledge representations equivalent to decision rules are considered, including decision trees and decision tables [2]. They prove to be helpful in the visual design of rules, as well as providing CASE tools for rule-based systems. The theory and practice of rules verification is a well studied field, for details see [10]. Therefore, evaluating the use of mature rule-based systems solutions is a natural choice for the Semantic Web applications.

The XTT^2 [6] rule formulation and design language based on the ALSV(FD) is an example of an expert system design and analysis framework. It offers flexible knowledge representation for forward chaining decision rules, as well as visual design tools. Thanks to the formal description in ALSV(FD) a formalized rule analysis is also possible. As a rule framework XTT^2 provides a subset of functionality that SWRL aims for, at least for the production systems based on decision rules. On the other hand, it has ready design and analysis solutions.

The primary goal of this research is to allow the use of the XTT^2 rule design framework for the Semantic Web rules. This would open up possibility to use the visual design tools for XTT^2 to design Semantic Web rules, as well as exploit the existing verification solutions. The following phases are considered:

1. a transition from ALSV(FD) to a subset of a selected DL language,
2. XTT^2 rules formulation compatible with the above transition procedure,
3. visual design of XTT^2 rules for the Semantic Web,
4. XTT^2 rules translation to SWRL, with possible RIF-only extensions,
5. rule inference on top of OWL-based ontologies.

Ultimately it should be possible to design Semantic Web rules with the XTT^2 visual design tools and provide a formal analysis of rules. XTT^2 could be run with a dedicated XTT^2 engine, or a SWRL runtime (whenever it is available).

In this paper the focus is on the first phase, with some considerations for the 2nd and the 3rd one. In order to provide a transition, a discussion of syntax and semantics of both calculi DL and ALSV(FD) is given in the next section.

5 Syntax and Semantics Analysis

Description Logics enable for complex descriptions of objects in the universe of discourse and the relations between them. The static part of the system is expressed in TBox part of a DL Knowledge Base. The actual state is represented by means of facts asserted in ABox. ABox in DL is limited in terms of its syntax and semantics. There are only simple assertions there, which together with knowledge expressed in TBox lay the ground for inferencing.

The language of DL consists of *concepts*, *roles* and *constants*. The meaning of the symbols is defined by an *interpretation function*, which to each concept assign a set of objects, to each role a binary relation, and to each individual an object in the universe of discourse.

The main goal of ALSV(FD) is to provide an expressive notation for dynamic system state transitions in case of rule-based systems. Thus, the knowledge specification with ALSV(FD) is composed of: state specification with facts, and transition specification with formulas building decision rules.

Both logics describe the universe of discourse by identifying certain entities. In ALSV(FD) they are called attributes and in DL – concepts. Here we will concentrate on simple attributes.

Every attribute in ALSV(FD) has its domain, which constraints the values of the attribute. In DL this kind of specification is done by means of TBox axioms. In order to express a finite domain in DL, a *set of* constructor, denoted by $\mathcal{O}$ is needed (see Table 1).

Table 1. Formulas in AL and terminological axioms in DL – domain definitions

Attributive Logic		Description Logic	
Attribute Name	Attribute domain	Concept Name	Concept constructors
A_i	D_i $D = \{a_1, a_2, \dots, a_n\}$	A_i	$A_i \equiv D_i$ $D \equiv \{a_1, a_2, \dots, a_n\}$

Once the attributes and concepts are defined, they are used in system rules specification. Legal AL formulas (see Defs. 4,5) specify the constraints that an attribute value has to match in order for a rule to be fired. They constitute a schema, to which states of a system in certain moments of time are matched. ALSV(FD) formulas used in rule preconditions can be represented as terminological axioms in DL as in Fig. 2.

Attribute values define the state of the system under consideration. State is represented as a logical formula (24) built from a conjunction of formulas specifying the values of respective attributes. A statement in AL that the attribute A_i holds certain value, in DL language corresponds to a statement that a certain object is an instance of concept A_i. Such statements build the ABox in DL systems. State specification is shown in Tab. 2.

Fig. 2. Simple attributes formulas in AL rules and respective axioms in DL

Attributive Logic	Description Logic
Formula	Axiom
$A_i = d$	$A_i \equiv \{d\}$
$A_i \in V_i$	$A_i \equiv V_i$
$A_i \neq d$	$A_i \equiv \neg\{d\}$
$A_i \notin V_i$	$A_i \equiv \neg V_i$

Table 2. State specification in AL and assertions in DL

Attributive Logic	Description Logic
Formula	Assertion in ABox
$A_i := d_i$	$A_i(d_i)$

At a given moment of time, the state of the system is represented as a conjunction of formulas or concept assertions in DL respectively. In order to check the rules satisfiability, appropriate inference tasks have to be executed. The inference rules in AL are presented in Fig. 1. For DL such a task is *consistency checking*. For each rule a *consistency checking* of the state assertions with regards to the rule preconditions is performed. If the consistency holds, the rule can be fired. Every rule is loaded into the DL reasoner together with the actual state of the system. The rule constitues a schema, which is temporary joined with existing TBox (in which concept definitions are stored). The state is a *temporary ABox*. The DL reasoner checks the consistency of the ontology resulting from the TBox and ABox. If it is consistent, then the rule can be fired. The actions specified in the heads of rules may change the system state, which is then represented as a new ABox.

6 An Example of Translation

Let us consider a simple example of a forward chaining rule based system. The goal of the system is to set a thermostat temperature based on the condition, namely the time specification.
Consider a set attributes
$\mathbf{A} = \{day, month, hour, today, season, operation, therm\_setting\}$,
with corresponding domains:
$\mathbf{D} = \{D_{day}, D_{month}, D_{hour}, D_{today}, D_{season}, D_{operation}, D_{therm\_setting}\}$,
defined as $D_{day} = \{sun, mon, tue, wed, thr, fri, sat\}$,
$D_{month} = \{jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec\}$,
$D_{hour} = \{1 - 24\}$, $D_{today} = \{workday, weekend\}$,

$D_{season} = \{winter, spring, summer, autumn\}$,
$D_{operation} = \{bizhours, notbizhours\}$, $D_{therm\_setting} = \{12 - 30\}$,
In such a system we can consider simple production rules (in Australia!):
$R1 : month \in \{dec, jan, feb\} \longrightarrow season := summer$
$R2 : day \in \{mon, tue, wed, thr, fri\} \longrightarrow today := workday$
$R3 : today \in \{workday\} \wedge hour \in \{9 - 17\} \longrightarrow operation := bizhours$
$R4 : operation = \{bizhours\} \wedge season = \{summer\} \longrightarrow therm\_setting := 24$
In Description Logics language the following concepts are distinguished:
Day,*Month*, *Hour*, *Today*, *Season*, *Operation*, *Therm_setting.*
The definition of the concepts is as follows:
$Day \equiv \{sun, mon, tue, wed, thr, fri, sat\}$,
$Month \equiv \{jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec\}$,
$Hour \equiv \{1, 2, \ldots, 24\}$,
$Today \equiv \{workday, weekend\}$,
$Season \equiv \{winter, spring, summer, autumn\}$,
$Operation \equiv \{bizhours, notbizhours\}$,
$Therm\_setting \equiv \{12, 13, \ldots, 30\}$
To simplify the notation one can introduce the following concepts:
$SummerMonths \equiv \{dec, jan, feb\}$,
$Bizhours \equiv \{9, 10, \ldots, 17\}$,
$WorkingDays \equiv \{mon, tue, wed, thr, fri\}$.
and using the transition specified in Fig. 2 write the rules as follows:
$R1 : (Month \equiv SummerMonths) \rightarrow Season(summer)$,
$R2 : (Day \equiv WorkingDays) \rightarrow Today(workday)$,
$R3 : (Today \equiv \{workday\}) \wedge (Hour \equiv BizHours) \rightarrow Operation(bizhours)$,
$R4 : (Operation \equiv \{bizhours\}) \wedge (Season \equiv \{summer\}) \rightarrow Therm\_setting(24)$

Let the state in moment 0, be represented as ABox_0 as follows:

$$Month(jan).Day(mon).Hour(11). \tag{25}$$

The inference process is as follows: The state ABox_0 (25) and the preconditions of rule R1 are loaded into a DL reasoner. The DL reasoner performs the consistency checking of the state with respect to rule preconditions. Because the ontology built from the state assertions and R1 precondition formulas is consistent ($Month(jan)$ is consistent w.r.t. $Month \equiv SummerMonths$ ($SummerMonths \equiv \{dec, jan, feb\}$)) the rule is fired. The conclusion of the rule generates a new assertion in the state formula. The new ABox_1 replaces the old one (ABox_0). The new state ABox_1 is as follows:

$$Month(jan).Day(mon).Hour(11).Season(summer). \tag{26}$$

The state ABox_1 (26) and the preconditions of rule R2 are loaded into the DL reasoner. The DL reasoner performs the consistency check of the state with respect to rule preconditions. Because this time the ontology built from the state assertions and R2 precondition formulas again is consistent ($Day(mon)$ is consistent w.r.t. $Day \equiv WorkingDays$ ($WorkingDays \equiv \{mon, tue, wed, thr, fri\}$) the rule is fired. The conclusion of the rule generates a new assertion. The reasoning continues with new state and the next rules.

7 Future Work

The research presented in the paper affects integration of the selected expert system rule design method and the Semantic Web. Several design tools have been implemented for the design and verification of XTT^2-based systems. These could be used for building such systems in the context of the Semantic Web.

To use DL modelling and reasoning capabilities to a larger extent, the ontology of the example used above and other systems could be built in a different manner. Some of the rules could be included into subsumption relations such that those rules would be executed as an intrinsic DL inference.

The XTT^2 toolset (see `hekate.ia.agh.edu.pl`) also provides a custom inference engine able to execute the XTT^2 rules. The engine has an extensible architecture and is implemented in Prolog. Future work includes developing an interface able to use some dedicated DL reasoners such as Pellet. This would allow to reason with XTT^2 rules on top of existing OWL ontologies.

Acknowledgements The authors wish to thank Claudia Obermaier for her valuable remarks concerning the final version of the paper.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
2. van Harmelen, F., Lifschitz, V., Porter, B., eds.: Handbook of Knowledge Representation. Elsevier Science (2007)
3. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. 2nd edn. Prentice-Hall (2003)
4. Giarratano, J., Riley, G.: Expert Systems. Principles and Programming. Fourth edition edn. Thomson Course Technology, Boston, MA, United States (2005) ISBN 0-534-38447-1.
5. Ligęza, A.: Logical Foundations for Rule-Based Systems. Springer-Verlag, Berlin, Heidelberg (2006)
6. Nalepa, G.J., Ligęza, A.: Xtt+ rule design using the alsv(fd). In Giurca, A., Analyti, A., Wagner, G., eds.: ECAI 2008: 18th European Conference on Artificial Intelligence: 2nd East European Workshop on Rule-based applications, RuleApps2008: Patras, 22 July 2008, Patras, University of Patras (2008) 11–15
7. Klösgen, W., Żytkow, J.M., eds.: Handbook of Data Mining and Knowledge Discovery. Oxford University Press, New York (2002)
8. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: combining logic programs with description logic. In: Proceedings of the Twelfth International World Wide Web Conference, WWW2003. (2003) 48–57
9. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: Swrl: A semantic web rule language combining owl and ruleml, w3c member submission 21 may 2004. Technical report, W3C (2004)
10. Ligęza, A., Nalepa, G.J.: Logical Representation and Verification of Rules. In: Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches. Information Science Reference (2009)