Critical evaluation of the XTT2 rule representation through comparison with CLIPS[1]

Krzysztof Kaczor and **Grzegorz J. Nalepa**[2]

Abstract. There are two main approaches to the design Business Rules. The first one involves formalized methods that strictly define the syntax and semantics of rules. This approach usually requires technical skills or conceptual knowledge and therefore is not appropriate for everyone. In the second approach, dedicated rule languages are used for facilitating rules specification. Nevertheless, such languages are usually programming solutions without a precisely defined semantics. This may cause ambiguities in knowledge interpretation and thus, the efficient rule interoperability becomes impossible. The goal of our work is to develop a formalized model for a rule representation which will allow for an effective rule interchanging. For this purpose, we want to combine the above mentioned approaches by tailoring the formalized rule representation called XTT2 to languages provided by CLIPS or Drools. This paper is the first step in our research providing an identification of the most important differences between the XTT2 and CLIPS rule languages.

1 Introduction

Rule-Based Systems constitute a mature technology in the field of Artificial Intelligence. Over the years, they were applied in many domains like medicine, engineering [7] or decision support [10]. Despite their maturity, many ideas, algorithms and solutions that are applied in new technologies, such as Business Rules (BR) [22], Semantic Web [2] or Complex Event Processing [12], are derived from the classic Rule-Based Systems [9].

Business Rules are one of the latest application of classic rules. They are intended to be created by business people in order to define logical aspects of business. Despite the fact that business people may not have any technical skills or scientific knowledge, BR must be appropriate to be used by such users. Currently, many techniques are used for the specification of BR, from description in natural language to design by using formalized methods. There is no single method that is considered to be the best. Usually, a designer chooses one according to his or her own preferences.

Rules specification in natural language is very intuitive and does not require any specialized skills. Moreover, such a method allows for an easy specification of very complex rules. However, such informal description may be very vague, especially in case of complex rules which may be hard to understand or in the worst case may be misunderstood. This type of problems can be prevented by using formalized methods having the following advantages:

- they provide a clear framework enabling uniform knowledge modeling with well-defined expressive power,
- speed up the design process – formalized rule language opens possibility to partially formalize the design process which can, in turn, lead to better detection of design errors, possibly at early development stages,
- allow for a superior knowledge base quality control – formal methods can be used to identify logical errors in rule formulation,
- simplify knowledge interoperability – partially formalized translation to other knowledge representation formats are possible, and
- allow for custom inference modes – structured rule bases require inference strategies alternative to the classic inference algorithms.

This paper is organized as follows: Section 2 gives a short motivation for our work. A short introduction to the XTT2 method is provided by Section 3. Section 4 is the main part of this paper discussing the most important differences between XTT2 and CLIPS. The paper is concluded with Section 5 providing short summary and information concerning future works.

2 Motivation

Together with the development of BR design methods, a number of development tools also increases. Among them very important are Drools [4] and OpenRules[3]. Sometimes, it is desirable to have a mechanism for exchanging knowledge between different tools. This makes maintenance of the rule bases easier and allows for more efficient usage of these tools. Nevertheless, the existing tools usually allow for BR modeling in an informal way and do not provide any common rule representation model. This provides ambiguity in the rule semantics and in turn, does not allow for efficient interchanging.

A problem of knowledge interoperability is known since classic rule-based expert systems and still remains an open issue. During the years, several approaches to this problem were proposed. The most important of them are: Knowledge Interchange Format[4], Rule Markup Language [3], Rule Interchange Format [8] and REWERSE Rule Markup Language [23]. Nevertheless, the above mentioned methods provide a very general model of rule-based knowledge representation, what makes their practical application hard or even impossible. Hence, practical tools supporting any of these methods do not exist or provide only partial support.

The main objective of our current research is to develop a formalized method for an efficient rule interoperability. We assume that this can be done by providing a common and logic-based rule representation model. Thanks to such a model, the semantics of rules, specified in other representations, can be clarified or defined. What

[1] The paper is supported by the AGH UST Grant.
[2] AGH University of Science and Technology, Poland, email: kk,gjn@agh.edu.pl

[3] See: `http://openrules.com`
[4] See: `http://www.upv.es/sma/teoria/sma/kqml_kif/kif.pdf`

is more, this model will allow to specify which representation can be losslessly translated to another and how this translation should be performed. We assume that the model will be based on the formalized rule representation method called XTT2 [19] which is provided by the Semantic Knowledge Engineering (SKE) methodology [16]. The XTT2 method (see Section 3) is a visual method for modeling structured rule bases. This method is intended to be a rigorously formalized rule language. Nevertheless, a rigorous formalization has restricted the expressiveness of the language. Thus, in comparison with other methods, XTT2 has several limitations and significant differences. This is why our current work is focused on the extension of XTT2 towards such languages as CLIPS[5] [6] or Drools. These two languages have been selected as the reference because they proved to be successful implementations of rule-based systems.

CLIPS is a classic rule-based expert system shell developed by NASA in 1984. The original intent for CLIPS was to gain useful insight and knowledge about the construction of expert system tools and to lay the groundwork for the construction of a replacement tool for the commercial tools being used in that time. Because of its portability, extensibility, capabilities, and low cost, it has received widespread acceptance throughout the government, industry, and academia. Development of this tool has improved the accessibility to expert system technology throughout the public and private sectors for a wide range of applications and diverse computing environments. CLIPS became one of the most commonly known rule language that was used for e.g. image processing or recognition.

As a classic rule-based tool, CLIPS became a reference tool also for other tools like Jess which is a rule engine and scripting environment providing rule language. It is written in Java. Jess was originally a clone of the essential core of CLIPS, but has begun to acquire a Java-influenced flavor. Therefore, it is a convenient tool for giving Java applets and applications the ability to reason.

Drools is a much younger project which was started in 2001. Currently Drools is widely used by Business environment as Business Logic Integration Platform providing a unified and integrated platform for Rules, Workflow and Event Processing. Drools-based rules are specified using dedicated rule language and processed by dedicated rule engine called Drools Expert. Similarly to Jess, this engine is also written in Java and allows for easy integration with other applications written in this language.

The above mentioned tools are not intended to provide a formalized rule representation that is necessary for efficient rule interchange preserving their semantics. The provide only programming solutions for rapid development of the rule bases. In our work, we try to combine the advantages of formalized methods and programming solutions. This paper describes the first step of this work. The main contribution is the comparison of the XTT2 method with the CLIPS language, by identifying the differences and limitations of XTT2 in terms of CLIPS. It describes the most important aspects of extending XTT2 towards CLIPS and challenges that must be overcome for an efficient rule interoperability between these two representations.

3 Overview of XTT2

This section gives a short introduction to XTT2 (*eXtended Tabular Trees*) [17, 18]. XTT2 can be considered a multidimensional concept. It involves many aspects of the rule-based systems design:

- Rule Base — this aspect involves issues related to structure and maintenance of a rule base.
- Rule Syntax — defines how the knowledge can be expressed and what are the limitations of a provided rule language i.e. this issue concerns rule language syntax as well as its expressiveness.
- Rule Semantics — defines the semantics of the rules and how they should be interpreted.
- Rule Processing — is related to inference mechanism as well as the way how the knowledge processing is performed.

This section is divided into four subsections describing XTT2 in terms of above mentioned aspects.

3.1 Rule Base

An XTT2-based rule base contains attributes that store values. Each attribute-value pair can be considered as a single fact. The set of all pairs is called system state and is defined as follows:

$$s\colon (A_1 = S_1) \wedge (A_2 = S_2) \wedge \ldots \wedge (A_n = S_n) \tag{1}$$

where A_i are the attributes and S_i are their current values.
It is important to notice, that the number of attributes (facts) is constant during the inference process. The knowledge base modification can be made only by changing attribute value.

XTT2 provides modularized rule base, where rules working together are placed in one context. Contrary to the majority of other systems, where a basic knowledge item is a single rule, in the XTT2 formalism the basic component displayed, edited and managed at a time is a single *context*. A single context corresponds to a single decision table. Thus, only those rules which have the same conditional and decisions attributes can be placed in one context i.e. each rule in a decision table determines values of the same set of attributes.

XTT2 is a hybrid knowledge representation combining a decision network and decision tables. Tables are linked together forming a network-like structure of the XTT2 decision tables. Links define order in which tables should be processed. Considering a single table as a blackbox for determining attribute value, the links corresponds to functional dependencies between attributes.

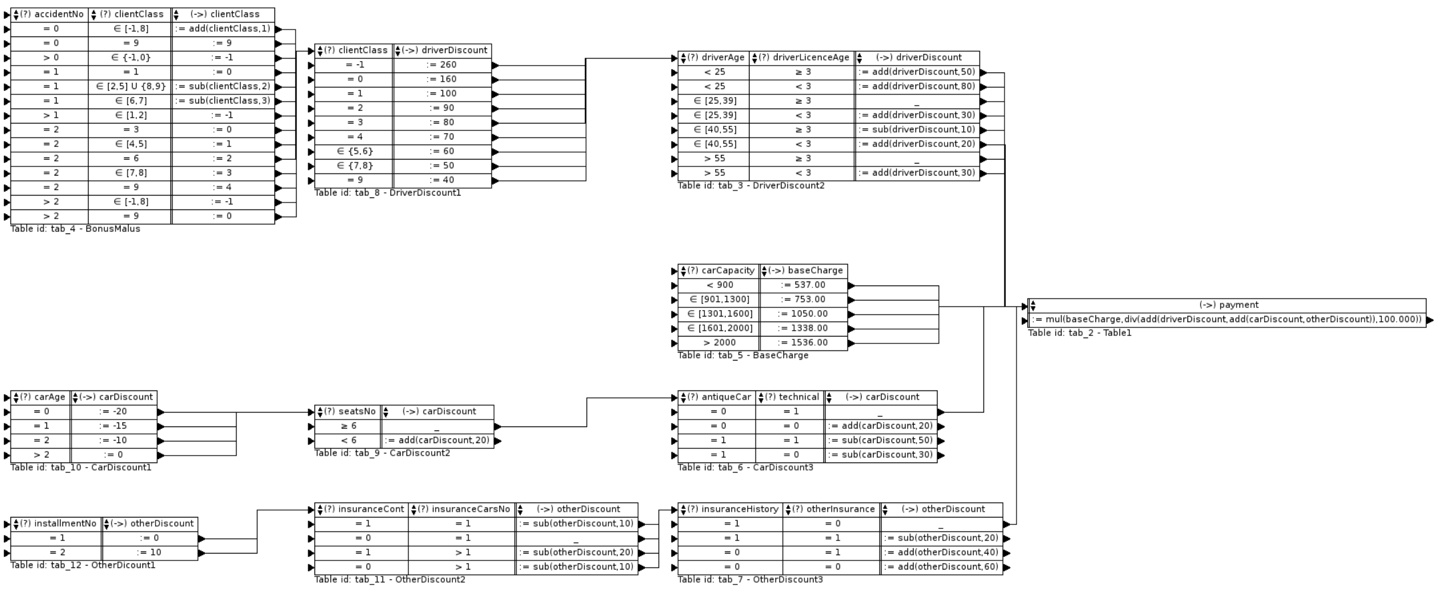
3.2 Rule Syntax

The XTT2 rule language provide a dedicated syntax called *HeKatE Meta Representation* (HMR). This is a textual representation that can be easily read by human and automatically processed by an inference engine. Moreover, the HMR language is suitable for visual representation (see Figure 1). Thanks to this, such a knowledge representation provides not only high density of knowledge visualization, but assures transparency and readability. Additionally, the visual representation is fully supported by the HQEd [20] graphical editor. Using this editor, a HMR-based representation can be automatically generated for a given visual model. Then, the HMR representation is processed by the HeaRT [15] tool which is the dedicated inference engine for reasoning with the XTT2 rule bases [1].

For study purposes, an example below presents the same rule in three representations: natural language, HMR representation and XTT2-based visual representation. The rule comes from Cashpoint case study [5] and is as follows:

```
if
  driver is younger than 25 years
  and
  it has driving licence at least three years
then
  increase the driver current discount by 50%
```

[5] See: http://clipsrules.sourceforge.net



(?) accidentNo	(?) clientClass	(->) clientClass
= 0	∈ [-1,8]	:= add(clientClass,1)
= 0	= 9	:= 9
> 0	∈ {-1,0}	:= -1
= 1	= 1	:= 0
= 1	∈ [2,5] U {8,9}	:= sub(clientClass,2)
= 1	∈ [6,7]	:= sub(clientClass,3)
> 1	∈ [1,2]	:= -1
= 2	= 3	:= 0
= 2	∈ [4,5]	:= 1
= 2	= 6	:= 2
= 2	∈ [7,8]	:= 3
= 2	= 9	:= 4
> 2	∈ [-1,8]	:= -1
> 2	= 9	:= 0

Table id: tab_4 - BonusMalus

(?) clientClass	(->) driverDiscount
= -1	:= 260
= 0	:= 160
= 1	:= 100
= 2	:= 90
= 3	:= 80
= 4	:= 70
∈ {5,6}	:= 60
∈ {7,8}	:= 50
= 9	:= 40

Table id: tab_8 - DriverDiscount1

(?) driverAge	(?) driverLicenceAge	(->) driverDiscount
< 25	≥ 3	:= add(driverDiscount,50)
< 25	< 3	:= add(driverDiscount,80)
∈ [25,39]	≥ 3	_
∈ [25,39]	< 3	:= add(driverDiscount,30)
∈ [40,55]	≥ 3	:= sub(driverDiscount,10)
∈ [40,55]	< 3	:= add(driverDiscount,20)
> 55	≥ 3	_
> 55	< 3	:= add(driverDiscount,30)

Table id: tab_3 - DriverDiscount2

(?) carCapacity	(->) baseCharge
< 900	:= 537.00
∈ [901,1300]	:= 753.00
∈ [1301,1600]	:= 1050.00
∈ [1601,2000]	:= 1338.00
> 2000	:= 1536.00

Table id: tab_5 - BaseCharge

(->) payment
:= mul(baseCharge,div(add(driverDiscount,add(carDiscount,otherDiscount)),100.000))

Table id: tab_2 - Table1

(?) carAge	(->) carDiscount
= 0	:= -20
= 1	:= -15
= 2	:= -10
> 2	:= 0

Table id: tab_10 - CarDiscount1

(?) seatsNo	(->) carDiscount
≥ 6	_
< 6	:= add(carDiscount,20)

Table id: tab_9 - CarDiscount2

(?) antiqueCar	(?) technical	(->) carDiscount
= 0	= 1	_
= 0	= 0	:= add(carDiscount,20)
= 1	= 1	:= sub(carDiscount,50)
= 1	= 0	:= sub(carDiscount,30)

Table id: tab_6 - CarDiscount3

(?) installmentNo	(->) otherDiscount
= 1	:= 0
= 2	:= 10

Table id: tab_12 - OtherDicount1

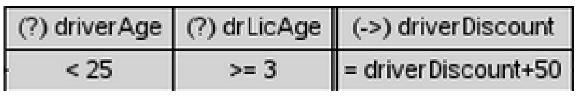
(?) insuranceCont	(?) insuranceCarsNo	(->) otherDiscount
= 1	= 1	:= sub(otherDiscount,10)
= 0	= 1	_
= 1	> 1	:= sub(otherDiscount,20)
= 0	> 1	:= sub(otherDiscount,10)

Table id: tab_11 - OtherDiscount2

(?) insuranceHistory	(?) otherInsurance	(->) otherDiscount
= 1	= 0	_
= 1	= 1	:= sub(otherDiscount,20)
= 0	= 1	:= add(otherDiscount,40)
= 0	= 0	:= add(otherDiscount,60)

Table id: tab_7 - OtherDiscount3

Figure 1. An example of visual representation of XTT2 rule base



(?) driverAge	(?) drLicAge	(->) driverDiscount
< 25	>= 3	= driverDiscount+50

Figure 2. An example of visual representation of XTT2 rule

This rule can be easily expressed with HMR syntax:

```
xrule 'Table1/1':
  [driverAge lt 25,
   drLicAge gte 3]
==>
  [driverDiscount set (driverDiscount+50)]
```

The figure 2 depicts the equivalent visual representation of the provided HMR syntax.

3.3 Rule Semantics

XTT2 is based on the *Attributive Logic with Set of Values over Finite Domains* (ALSV(FD)) logic [11, 18]. ALSV(FD) is a formal framework for attributive logic that provides syntax, semantics and some notes on inference rules for a logical calculus in which attributes can take *set values* (generalized attributes). In comparison with other attributive logics, its expressive power is increased through the introduction of new relational symbols enabling definitions of atomic formulae. This logic provides very strict and rigorous definition of rule semantics allowing for knowledge definition which can be unambiguously interpreted.

A single rule in ALSV(FD) is defined as a set of ALSV(FD) triples. The exemplary rule from Section 3.2 can be expressed in ALSV(FD) in the following way:

$$r_1\colon (driverAge, <, 25) \land (drLicAge, >=, 3) \rightarrow$$
$$\rightarrow (driverDiscount, :=, driverDiscount + 50)$$

The complete formalization of XTT2 can be found in [19].

3.4 Rule Processing

XTT2 provides a dedicated rule processing mechanism. This is an advanced inference algorithm that can work in one of four modes: Fixed Order, Data, Token and Goal Driven (for more details see [13]). The inference mechanism is responsible for evaluating and executing (firing) rules. The rules are processed in predetermined order which is specified by taking the following issues into account: inference mode, links between modules, order of the rules in the XTT2 table.

4 Challenges in the Rule Interoperability between XTT2 and CLIPS

The goal of our work is to develop a common unified rule representation model for efficient rule interoperability between different rules representations. Our starting point is formalization of the XTT2 method which provides the underlying ALSV(FD) logic. Nevertheless, the current form of the method does not use the ALSV(FD) logic effectively (lack of support for complex types) and has several limitations. In comparison to CLIPS, this language constitutes a subset of CLIPS. In this context, the knowledge interchange between XTT2 and CLIPS requires many improvements and changes that must be done in XTT2 in order to provide a better coverage of CLIPS. Some of the limitations stem from the made assumptions and others stem from the visual representation. This section provides a detailed description of the most important challenges in the context of extending XTT2 and efficient rule interoperability with CLIPS. The section is divided into subsections according to aspects introduced in Section 3.

4.1 Rule Base

Rule base modification CLIPS provides a mechanism allowing for modifying a Knowledge Base (KB) by asserting, retracting and modifying facts. This can be done using the following commands: `assert`, `retract` and `modify`. Thanks to this, when the system is running, the number of facts in the KB can be changed. In contrast to CLIPS, the XTT2-based knowledge base defines a system with a constant number of facts described by attributes. During the execution, attributes are neither created nor removed from the knowledge base. Only the current value of the attribute can be changed. Nevertheless, the process of asserting and retracting facts in XTT2 can be achieved using generalized attributes (see paragraph *Multivalued attributes* in Section 4.2).

Modularization of Knowledge Base Both, XTT2 or CLIPS provide mechanism for creating modularized knowledge base. In XTT2, set of rules that work together are grouped into so called contexts. Each context corresponds to a single XTT2 table and contains rules which have the same attributes in their conditional and conclusion parts. In turn, CLIPS provide modules which can be defined using `defmodule` construct. In contrast to XTT2, CLIPS modules do not provide any policy determining which construct can be placed in a module. In particular any rule (or other construct) can be placed in any module. In fact, this is also possible in XTT2 and can be achieved by extending rules LHS and RHS by all attributes that appears in the other rules in this context. However, this can lead to formation of large tables in which majority of cells contain always true comparisons (an example of such table is depicted in Figure 3).

CLIPS modules allow a set of constructs to be grouped together such that explicit control can be maintained over restricting the access of the constructs by other modules. This type of control is similar to global and local scoping used in languages such as C. The default behavior in CLIPS restricts constructs in one module to be accessible in another. However, this can be modified and selected elements can be permitted to be visible from other modules. In turn, the XTT2 rules placed in one context are not accessible from another.

In both CLIPS or XTT2, modules are used by rules to provide execution control. In CLIPS, each module has its own pattern-matching network [13], and thus only rules from the active module can be activated and executed. Similarly, in XTT2 only rules from the active context are evaluated and can be executed.

Variables CLIPS provides two elements which allow for storing information: facts and variables. Nevertheless, the semantics of these two elements is different. Facts are knowledge-based elements which defines what is currently known. Any change made in a set of facts invokes the pattern-matching process. In contrast to facts, variables are used for defining non knowledge-based values e.g. values of some factors, constant values, etc. Variables can be used as a part of pattern-matching process, however changes of their values do not invoke pattern-matching. CLIPS variables can be defined using `defglobal` construct e.g.:

```
(defglobal ?*high-priority-factor* = 100)
```

In turn, XTT2 does not provide any concept having the same semantics as CLIPS variables. The system designed with XTT2 consists of attributes. According to ALSV(FD), the state of the XTT2-based system is defined as a set of current values of all attributes specified within the knowledge base. From logical the point of view, the state of the system is represented as a logical formula (1). According to this definition, all XTT2 attributes are considered to be knowledge-based elements.

It is important to notice that the inference mechanism from XTT2 works in different way than in CLIPS. It evaluates rules in predetermined order and changes in attribute values do not affect it.

4.2 Rule syntax

Complex types The first and most important limitation of the XTT2 is related to complex types. A complex type is a data type that provides its own structure and aggregates a fixed set of labelled fields, possibly of different types, into a single type. An example of such type is a structure that is known from C programming language. The ALSV(FD) logic provides support for complex types and objects throughout attribute function which denotes a property of an object and allows for accessing its value using property name. However, currently XTT2 uses only atomic types for defining all attributes in the knowledge base and assumes that only one object (in this case it is the system being described) with a specific property name exists. In turn, CLIPS provides `deftemplate` element that allows for defining complex facts consisting of number of typed properties (called slots in CLIPS-based vocabulary):

```
(deftemplate person
  (slot name (type SYMBOL))
  (slot surname (type SYMBOL))
  (slot gender (type SYMBOL))
  (slot height (type INTEGER))
  (slot age (type INTEGER))
)
```

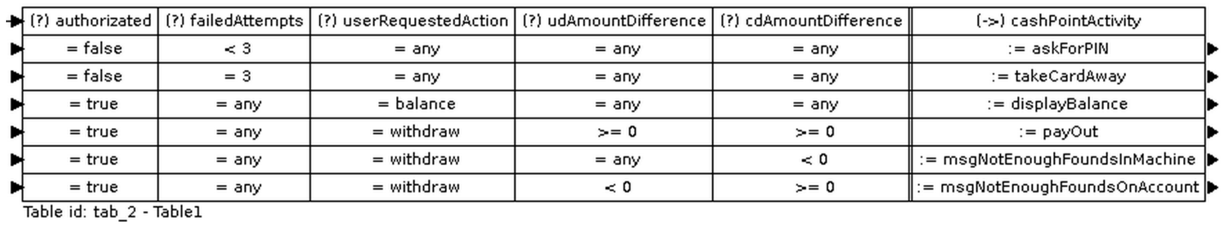
This example defines a template of person which allows for creating complex facts consisting of five typed properties: `name`, `surname`, `gender`, `height` and `age`.

Multivalued attributes ALSV(FD) provides a generalized attribute that can take more than one value at any point of time. This is very important and useful feature of ALSV(FD), however it is hard to assess to which element of the CLIPS language it corresponds. There are two obvious possibilities:

- facts list of the same type – a generalized attribute can be used for aggregation of values having the same type. A value of generalized attribute is defined as set. Such sets can be modified using set theory operators. In particular union of sets or difference of sets can correspond to CLIPS operations of asserting or retracting facts to/from knowledge base.
- multivalued slots – the `deftemplate` construct in CLIPS allows for defining multivalued slots which can take more that one value:

```
(deftemplate person
  (slot name (type SYMBOL))
  (slot surname (type SYMBOL))
  (slot gender (type SYMBOL))
  (slot age (type INTEGER))
  (slot height (type INTEGER))
  (multislot friends (type SYMBOL))
)
(assert (person (name Tom) (surname Joe)
  (gender M) (age 18) (height 180)
  (friends John Alex Emma))
)
```

  This defines a man (`M`) `Tom Joe` that is `180` cm tall and `18` years old and has three friends: `John`, `Alex` and `Emma`. It is important to notice that the list of friends is not treated as one string containing spaces, only as the list of three separate values.
  Usually multislot contains values with the same semantics (informally described by a slot name). Apart from the support for complex types, the generalized attribute in XTT2 can be used in the same context as the multivalued slots in CLIPS.



(?) authorizated	(?) failedAttempts	(?) userRequestedAction	(?) udAmountDifference	(?) cdAmountDifference	(->) cashPointActivity
= false	< 3	= any	= any	= any	:= askForPIN
= false	= 3	= any	= any	= any	:= takeCardAway
= true	= any	= balance	= any	= any	:= displayBalance
= true	= any	= withdraw	>= 0	>= 0	:= payOut
= true	= any	= withdraw	= any	< 0	:= msgNotEnoughFoundsInMachine
= true	= any	= withdraw	< 0	>= 0	:= msgNotEnoughFoundsOnAccount

Table id: tab_2 - Table1

Figure 3. An example of a large XTT2 table

Expressions in LHS The XTT2 method provides mechanisms for logical quality analysis called HalVA [14]. It allows for discovering logical anomalies such as inconsistency, redundancy, contradictions etc. In order to assure higher efficiency of HalVA, the LHS of the rule can contain only a simple attribute-to-value comparisons e.g.:

```
A = 12    B > 23    C in {1,2,3}
```

Such comparisons test a specific attribute against its value. Thus, an attribute is always on the left hand side of a comparison and constant value or set of constant values on the right hand side. Neither attributes nor expressions are allowed on the right hand side e.g.:

```
A = 11+1   B < A-3
```

In turn, the *Right Hand Side* (RHS) of a rule can contain complex expressions and attribute references:

```
A := A+1   B := 4*3
```

In contrast to XTT2, CLIPS allows for any complex expressions in conditional part of the rules. This limitation of XTT2 can be omitted by creating an additional decision table having required expression in its RHS. The figure 4 depicts the equivalent construction in CLIPS and XTT2. The rules comes from the Cashpoint example [21] and are intended to check if a user has entered a correct PIN. This is done by comparing `enteredPIN` and `correctPin` attributes. The equality of this two attributes is a condition that must satisfied in order to authorize a user. In CLIPS this condition can be placed directly in LHS of a rule, while XTT2 required an additional table (`Table3`) and attribute (`pinDifference`).

Constraints ALSV(FD) provides a concept of attribute domain. A domain is a finite set of admissible values that attribute can take. Each domain is based on one of two primitive types *symbolic* or *numeric*. In XTT2, for each attribute a domain must be specified. The domain concept plays important role because it is used by verification mechanism for discovering logical anomalies in knowledge base. The example below shows a definition of types (in HMR language) restricting values of the attributes describing a person. We assume that:

name is not restricted and can contain any list of characters,
gender can take only two values: `M` for male and `F` for female,
height can take a value from the interval $[0, 300]$,
age can take a value from the interval $[0, 120]$.

```
xtype [name: name,
       base: symbolic].
xtype [name: gender,
       base: symbolic,
       domain: [M,F]].
xtype [name: height,
       base: numeric,
       domain: [0 to 300]].
xtype [name: age,
       base: numeric,
       domain: [0 to 120]].
```

In CLIPS, a value of a slot can be restricted using similar concepts: primitive types, list of values, ranges. However, CLIPS provides more primitive types than XTT2: `SYMBOL`, `STRING`, `LEXEME`, `INTEGER`, `FLOAT`, `NUMBER`, `INSTANCE-NAME`, `INSTANCE-ADDRESS`, `INSTANCE`, `EXTERNAL-ADDRESS`, and `FACT-ADDRESS`. Moreover, CLIPS allows for restricting a number of elements in multivalued slots.

The equivalent CLIPS-based definition of slot constraints describing person may look like this:

```
(deftemplate person
  (slot name (type SYMBOL) )
  (slot surname (type SYMBOL))
  (slot gender (type SYMBOL)
    (allowed-symbols M F))
  (slot height (type INTEGER) (range 0 300))
  (slot age (type INTEGER) (range 0 120))
  (multislot friends (type SYMBOL))
)
```

The one advantage of XTT2 in comparison with CLIPS is that the XTT2 allows for defining symbolic ordered domains. Such concept is similar to `enum` construct from C programming language. Thanks to ordering, the symbolic values can be treated as ordinary integer values e.g.:

```
xtype [
  name: weekdaytype,
  base: symbolic,
  domain: [mon/1,tue/2,wed/3,thu/4,
    fri/5,sat/6,sun/7],
  ordered: yes].
```

In this example a type describing weekdays is defined. Each day has assigned an equivalent numeric value. Thanks to that one can write:

```
mon > tue    A = tue+wed
```

The results of this expressions are equal to results of corresponding expressions where symbolic values were replaced with numeric.

Values binding In some cases, it is very hard or even impossible to define LHS of a rule by using only logical and relational operators. Let us consider the following example: The knowledge base contains information about a number of people described by properties defined in paragraph *Complex types*:

```
xrule 'Table3'/1:
    [enteredPin eq any,
      correctPin eq any]
  ==>
    [pinDifference set
      (correctPin-enteredPin)]
    :'Table2'.

xrule 'Table2'/1:
    [pinDifference neq 0]
  ==>
    [authorizated set false,
      failedAttempts set (failedAttempts+1)]
    :'Table1'.
xrule 'Table2'/2:
    [pinDifference eq 0]
  ==>
    [authorizated set true,
      failedAttempts set failedAttempts]
    :'Table1'.
```

```
(defrule rule-1
    ?a <- (atm (enteredPin ?e)
      (correctPin ?c)
      (failedAtempts ?f))
      (test (neq ?e ?c))
  =>
    (modify ?a (authorizated false)
      (failedAtempts (+ ?f 1))))

(defrule rule-2
    ?a <- (atm
      (enteredPin ?e)
      (correctPin ?c))
    (test (eq ?e ?c))
  =>
    (modify ?a (authorizated true)))
```

Figure 4. The equivalent construction in XTT2 (on the left) and CLIPS (on the right)

```
(person (name Tom) (surname Joe)
  (gender M) (age 18) (height 180)
  (friends John Alex Emma))

(person (name Emma) (surname Johnson)
  (gender F) (age 19) (height 180)
  (friends Tom Julia))

(person (name Alex) (surname Johnson)
  (gender M) (age 17) (height 170)
  (friends Tom Emma Julia))
```

Our task is to define a rule selecting all allowed pairs of persons which can dance together. Two person can dance together when satisfy the following conditions: 1) They have different gender and 2) they have the same growth. Such a rule can be easily written using mechanism allowing for value binding. This mechanism allows for retrieving desired value during inference and storing it in a user-defined variable. Then, this variable can be used in further conditions as well as conclusion part. The rule for our task can look like this:

```
(defrule rule-1 "Our solution"
  (person (name ?n1) (surname ?s1)
    (gender M) (height ?h))
  (person (name ?n2) (surname ?s2)
    (gender F) (height ?h))
=>
  (printout t ?n1 " and " ?n2 crlf)
)
```

The LHS of the rule contains two conditions that refers to `person` template. Thanks to this, the inference algorithm would try to match all possible pairs of `person` facts. When a single match is performed, then the variables (which names start with question mark) are bound to the current value of the matched fact. Binding is made only one time during a single match and the variable stores bounded value until this particular match is finished. Thus, usage of bounded variable in further conditions restricts the set of elements that can be matched because matching algorithm must take its value into account. So, the variable binding can be used for defining restrictions across several objects. In our example, the `?h` variable is bound in the first condition and then its value is used in the second condition. This restricts the set of possible facts that can be matched to the second condition, because apart from the value `F` of the `gender` slot, a matched fact must have the same value of the `height` slot as the fact matched in the first condition.

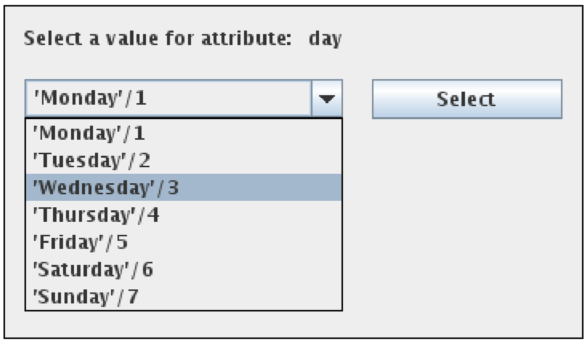
Variable bindings is currently not supported in XTT2. Thus, definitions of equivalent rule is currently not possible.

Functions CLIPS allows for defining functions. It is possible to define a user-defined external functions that can be written in an external language e.g. C and then linked with CLIPS during recompilation. Such functions can be later executed directly in CLIPS in ordinary manner. Moreover, CLIPS provides a second mechanism allowing for defining function directly in CLIPS by using CLIPS-based syntax. This can be done with the help of the `deffunction` construct. The CLIPS-based functions have all features that an ordinary function can have i.e.: unique name, list of parameters, sequence of actions, returned value, recursion. An example function that calculates the factorial of an argument can be written as follows:

```
(deffunction factorial (?a)
  (if (or (not (integerp ?a)) (< ?a 0)) then
    (printout t "Factorial Error!" crlf)
   else
     (if (= ?a 0) then
       1
     else
       (* ?a (factorial (- ?a 1))))))
```

It is important that each function can be invoked from any part of a rule and can modify a knowledge base.

XTT2 provides a similar mechanism to CLIPS user-defined external functions through callbacks. Callback function is an external function written in Prolog or Java language. Then, such function is invoked by Prolog interpreter directly or by using JPL plugin for callbacks written in Java.


Select a value for attribute: day

'Monday'/1 | Select

'Monday'/1
'Tuesday'/2
'Wednesday'/3
'Thursday'/4
'Friday'/5
'Saturday'/6
'Sunday'/7

Figure 5. An example of dialog invoked by callback function

Callbacks in XTT2 are strictly related to attributes. Each attribute can have two callback functions assigned: *input* callback and/or *output* callback. The *input* callbacks are used for retrieving attribute value from outside system when value of an attribute is not defined. Thus, this type of callback function can modify a knowledge base. In contrast to *input* callbacks, the *output* callbacks cannot modify state of the system and are used only for presentation purposes. The order and time when a callback is invoked is determined by inference algorithm and cannot be redefined. The example below depicts the definition of *input* callback and attribute to which is assigned:

```
xcall ask_console_symbolic: [AttName] >>> (
  alsv_domain(AttName,Domain,symbolic),
  write('availible answers are '),
  write(Domain), nl,
  write(AttName), write(': '), read(Answer),
  (member(Answer,Domain) ->

xattr [name: weekday,
  abbrev: weekday,
  class: simple,
  type: weekdaytype,
  comm: in,
  callback: [ask_console_symbolic,[day]]
```

This callback function invokes dialog allowing user to provide value of an attribute. The example of such dialog is depicted in Figure 5. The list of possible values is created according to attribute type. The definition of the attribute `weekday` type can be found in Section 4.2 in paragraph *Constraints*.

4.3 Rule semantics

Ordered structures CLIPS facts defined by using `deffact` construct are also called non-ordered facts. This is because the fact structure consists of fields that are referred by named slots. Additionally, CLIPS provides an ordered facts which encode information positionally. To access the information, a user must know not only what data is stored in a fact but which field contains the data. The first field of an ordered fact specifies a *relation* that applied to the remaining fields in the ordered fact e.g.:

```
(father-of jack bill)
```

This fact defines that `bill` is the `father of jack`.

The current XTT2 method does not provide any concept with similar semantics. ALSV(FD) provides support only for complex types, where properties of object are referred by attribute function.

Rules properties The way, a rule is processed by CLIPS can be modified by changing rule properties. CLIPS provides support for two properties `auto-focus` and `salience`.

The `auto-focus` property allows an automatic focus command to be executed whenever a rule becomes activated. If the `auto-focus` property for a rule is `true`, then a focus command on the module in which the rule is defined is automatically executed whenever the rule is activated. This property can be used for defining rules responsible for values validation:

```
(defrule VIOLATIONS::bad-age
  (declare (auto-focus TRUE))
  (person (name ?name) (age ?x&:(< ?x 0)))
=>
  (printout t ?name " has a bad age value."))
```

The above rule is activated whenever the `VIOLATIONS` module receives focus and checks if all the person facts accessible in that module have correct value of the slot `age`.

The `salience` property allows for assigning a priority to a rule. This property is a part of conflict resolution mechanism, which uses a `salience` value for determining order of rules to be fired. Rules with higher value have precedence to be executed.

XTT2 does not provide any rules properties directly. However, the ALSV(FD) logic defines the decision component (table) as follows:

$$t = (r_1, r_2, \ldots, r_n)$$

This means that rules placed in an XTT2 table are ordered. The inference engine uses this order for determining precedence of rules evaluation and execution. This precedence can be changed by moving rules in the table.

This behavior corresponds to CLIPS `salience` rule property. However, XTT2 forces the different values of rules priority in contrast to CLIPS that allows for rules with the same priority. This is why, the XTT2 method do not provide conflict resolution strategies.

4.4 Rule processing

Facts maintenance Any modification of KB in CLIPS that is done by using commands like `assert`, `retract` and `modify`, executes a pattern-matching algorithm which attempts to match rules to the current state of the system (as represented by the fact-list and instance-list). Each rule that has satisfied their conditional part (LHS – *Left Hand Side*) with respect to the modification is activated for execution. CLIPS allows non monotonic inference because each rule firing may again modify the KB. This inference process continues while KB is being modified. During this time, any rule can be activated and executed many times.

As it was mentioned, the XTT2 knowledge base contains a constant number of attributes (facts). The only modification that can be made is changing of the attribute value. However, in contrast to CLIPS, such modifications of the KB do not execute pattern-matching algorithm in order to find the rules that have satisfied their conditional parts against to a new system state.

This behavior is deliberate and follows from the method assumptions. According to this assumptions, the user defines the functional dependencies between attributes (links between tables). Thus, if a rule should be checked for execution when a value of an attribute is changed, then a user must define an appropriate dependency. This allows for optimized rule activating and more advanced inference control in comparison with CLIPS.

5 Summary

The main focus of this paper is to compare XTT2 with the CLIPS language. The scope of the provided comparison covers only the basic CLIPS language elements. In fact, the CLIPS language provides fully object oriented syntax called *CLIPS Object Oriented Language* (COOL). However, in the context of this paper the COOL syntax has not been taken into account. This paper tries to identify differences between these two languages in terms of the following aspects:

- Rule Base — differences related to knowledge maintenance and representation,
- Rule syntax — comparison of the languages expressiveness,
- Rule semantics — differences in knowledge interpretation,
- Rule processing — issues related to different knowledge evaluation and processing.

As it can be concluded from this paper, expressiveness of the XTT2 language (in comparison with CLIPS) is limited in each of the considered aspect. What is more, this paper shows that the ALSV(FD) logic, on which XTT2 is based, has also several limitations. On the other hand, in contrast to CLIPS, the XTT2 language provides strong underlying formalism playing a key role in rule interoperability. Due to the fact that CLIPS language is only a programming solution, a definition of an efficient CLIPS-based knowledge interchanging cannot be done. This is why, the extension of both the ALSV(FD) logic and XTT2 is a must in order to define an unified and formalized knowledge interoperability method. This extended formalism will allow for preserving rule semantics during interchanging. What is more, this method is intended to be supported by tools.

We selected the CLIPS language because it is considered to be successful in the Artificial Intelligence research community and have been used for many AI software projects. What is more, similarly to CLIPS, the XTT2 language is intended to be rule-based systems modeling method in their classic form. On the other side, the current application of rules (Business Rules) differs from the classic systems. One of the most important difference lies in different rule types. The classic systems usually provide only one rule type called *production rule*, while the BR-based languages provide five rule types: *Denotic Rules*, *Derivation Rules*, *Integrity Rules*, *Reaction Rules* and *Transformation Rules*. This rule classification is based on the specific rule properties (e.g. monotonicity of KB modification) and purposes (e.g. reaction on events). We do not discuss the differences between these types in details, because this is out of scope of this paper. These five types of Business Rules slightly extend the semantics of the *production rules*. Nevertheless, each type of Business Rule can be represented in classic rule-based systems using the *production rules*. This is why, despite the classic nature of CLIPS or XTT2, these languages can also be used for BR modeling.

The mentioned in Section 2. methods for rule interoperability (e.g. RIF) try to take rule properties and purpose into account. This is why, such a language is divided into so called dialects. RIF provides two standard dialects for rule representation: BLD (Basic Logic Dialect) and PRD (Production Rules Dialect). In general, the BLD and PRD dialects divide rules into two types: allowing for monotonic and non-monotonic changes in the Knowledge Base. In terms of BR types, usually the *Derivation*, *Denotic* and *Transformation rules* can be expressed in the BLD dialect while remaining in the PRD dialect.

Working on extension of XTT2 and ALSV(FD), the different types of rules will be taken into account and different formalisms will be provided. We assume, the unified rule representation model will be based on the Attributive Logic. However, this issue will be elaborated in details in the future work.

REFERENCES

[1] Weronika T. Adrian, Szymon Bobek, Grzegorz J. Nalepa, Krzysztof Kaczor, and Krzysztof Kluza, 'How to reason by HeaRT in a semantic knowledge-based wiki', in *Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2011*, pp. 438–441, Boca Raton, Florida, USA, (November 2011).

[2] Grigoris Antoniou and Frank van Harmelen, *A Semantic Web Primer*, The MIT Press, 2008.

[3] Harold Boley, Said Tabet, and Gerd Wagner, 'Design rationale for ruleml: A markup language for semantic web rules', in *SWWS*, eds., Isabel F. Cruz, Stefan Decker, Jérôme Euzenat, and Deborah L. McGuinness, pp. 381–401, (2001).

[4] Paul Browne, *JBoss Drools Business Rules*, Packt Publishing, 2009.

[5] Tim Denvir, Jose Oliveira, and Nico Plat, 'The Cash-Point (ATM) 'Problem'', *Formal Aspects of Computing*, **12**(4), 211–215, (Dec 2000).

[6] Joseph Giarratano and Gary Riley, *Expert Systems. Principles and Programming*, Thomson Course Technology, Boston, MA, United States, 4th edn., 2005. ISBN 0-534-38447-1.

[7] Adrain A. Hopgood, *Intelligent Systems for Engineers and Scientists*, CRC Press, Boca Raton London New York Washington, D.C., 2001.

[8] Michael Kifer and Harold Boley, 'RIF overview', W3C working draft, W3C, (October 2009). http://www.w3.org/TR/rif-overview.

[9] *The Handbook of Applied Expert Systems*, ed., Jay Liebowitz, CRC Press, Boca Raton, 1998.

[10] Antoni Ligęza, 'Expert systems approach to decision support', *European Journal of Operational Research*, **37**(1), 100–110, (1988).

[11] Antoni Ligęza, *Logical Foundations for Rule-Based Systems*, Springer-Verlag, Berli, Heidelberg, 2006.

[12] David Luckham, 'Complex event processing (CEP)', *Software Engineering Notes*, **25**(1), 99–100, (2000).

[13] Grzegorz Nalepa, Szymon Bobek, Antoni Ligęza, and Krzysztof Kaczor, 'Algorithms for rule inference in modularized rule bases', in *Rule-Based Reasoning, Programming, and Applications*, eds., Nick Bassiliades, Guido Governatori, and Adrian Paschke, volume 6826 of *Lecture Notes in Computer Science*, pp. 305–312. Springer, (2011).

[14] Grzegorz Nalepa, Szymon Bobek, Antoni Ligęza, and Krzysztof Kaczor, 'HalVA - rule analysis framework for XTT2 rules', in *Rule-Based Reasoning, Programming, and Applications*, eds., Nick Bassiliades, Guido Governatori, and Adrian Paschke, volume 6826 of *Lecture Notes in Computer Science*, pp. 337–344. Springer, (2011).

[15] Grzegorz J. Nalepa, 'Architecture of the HeaRT hybrid rule engine', in *Artificial Intelligence and Soft Computing: 10th International Conference, ICAISC 2010: Zakopane, Poland, June 13–17, 2010, Pt. II*, eds., Leszek Rutkowski and [et al.], volume 6114 of *Lecture Notes in Artificial Intelligence*, pp. 598–605. Springer, (2010).

[16] Grzegorz J. Nalepa, *Semantic Knowledge Engineering. A Rule-Based Approach*, Wydawnictwa AGH, Kraków, 2011.

[17] Grzegorz J. Nalepa and Antoni Ligęza, 'A graphical tabular model for rule-based logic programming and verification', *Systems Science*, **31**(2), 89–95, (2005).

[18] Grzegorz J. Nalepa and Antoni Ligęza, 'HeKatE methodology, hybrid engineering of intelligent systems', *International Journal of Applied Mathematics and Computer Science*, **20**(1), 35–53, (March 2010).

[19] Grzegorz J. Nalepa, Antoni Ligęza, and Krzysztof Kaczor, 'Formalization and modeling of rules using the XTT2 method', *International Journal on Artificial Intelligence Tools*, **20**(6), 1107–1125, (2011).

[20] Grzegorz J. Nalepa, Antoni Ligęza, Krzysztof Kaczor, and Weronika T. Furmańska, 'HeKatE rule runtime and design framework', in *Proceedings of the 3rd East European Workshop on Rule-Based Applications (RuleApps 2009) Cottbus, Germany, September 21, 2009*, eds., Adrian Giurca, Grzegorz J. Nalepa, and Gerd Wagner, pp. 21–30, Cottbus, Germany, (2009).

[21] Pascal Poizat and Jean-Claude Royer, 'Kadl specification of the cash point case study', Technical report, IBISC, FRE 2873 CNRS - Universite d'Evry Val d'Essonne, France, Genopole Tour Evry 2, 523 place des terrasses de l'Agora 91000 Evry Cedex, (January 2007).

[22] Barbara von Halle, *Business Rules Applied: Building Better Systems Using the Business Rules Approach*, Wiley, 2001.

[23] G. Wagner, A.Giurca, and S. Lukichev, 'R2ml: A general approach for marking up rules', in *Principles and Practices of Semantic Web Reasoning, Dagstuhl Seminar Proceedings 05371*, eds., F. Bry, F. Fages, M. Marchiori, and H. Ohlbach, (2005).