FELIS Programming Language for Rule Systems

D.S. Sfiris
Department of Civil Engineering
Democritus University of Thrace
Xanthi, Greece
Email: dmsfiris {at} otenet.gr

Abstract—**The gaining popularity of production rule systems had as a result the recent Rule Interchange Format (RIF) extensibility framework. This framework, developed by the W3C Consortium, is for modeling production rules in forward chaining rule engines. Our research, based on this RIF extensibility framework, sets the foundational work for a flexible new rule language to facilitate knowledge representation with syntax that incorporates features from both logic and functional programming. The benefits are grammatical freedom, absolute clarity and rigor. Formalizations of the new language and examples are provided.**

Keywords-computer languages; F-expressions; higher-order logic programming; production rule systems; rule dialects

I. INTRODUCTION

In this paper, we set the foundational work of the abstract core of a new production rule language. We call it FELIS (Flexible Expert Language and Inference System), designed for general use in Artificial Intelligence (AI) and the semantic web. The popularity of production systems resulted in the Rule Interchange Format (RIF) extensibility framework [1] to provide a standard dialect for production rules. RIF is compatible with the previously proposed standard of the Object Management Group (OMG) for modeling production rules in forward-chaining rule engines [2].

Based on the W3C Rule Interchange Format (RIF) extensibility framework as starting point, we designed an extension language with flexible syntax that incorporates features from both logic and functional programming. Early research has shown that functional and logic languages can be amalgamated without loosing the efficiency of either functional or logic language implementations [3].

A formal description of production systems resolves a lot of ambiguities and incompatibilities between different implementations. A recent formalization and refinement for a general model of production system is presented in [4]. We introduce predicate-free data structures, which we call F-expressions, to express knowledge. Similar to predicates of logic programming, F-expressions can formulate sentences describing properties of objects and/or relations between objects. They can also be used as functional terms to represent complex data. The terms of these expressions do not rely on any particular pattern such as that the first element must always be a symbol which represents a relation or an operation. Their syntax is rather free. Also, there is the ability to quantify not only over objects but also over relations and functions. F-expressions can be nested within each other to any depth, in order to create complex expressions of object-oriented properties and relations.

There is a long history of efforts to implement higher-order logic initially into logic programming [5]. As it is the case with programming languages such as HiLog [6], Lambda Prolog [7], Mercury [8] and others which mix features from logic programming and functional programming, we allow higher-order expressions and thus we deviate from the standard first-order logic paradigm. The definition of our core language syntax is thus predicate free and function free without any loss of the expressive power of first-order logic. This is because our F-expressions are used to represent both predicates and functions depending on the context.

Although the proposed language does not yet support the full features of RIF-PRD [9] (e.g. subclass and frame-style properties), it does have externally defined functions as well as syntactic constructs for describing actions of production rule systems. The final syntax turns out to be quite close to a natural language as it can be seen in the examples provided.

This paper is structured as follows: In Section II we discuss the production systems basics and relevant terminology, while in Section III the W3C RIF recommendation rule dialects are discussed. The proposed language is presented in Section IV. The benefits of the language and examples follow.

II. CONCEPTS AND TERMINOLOGY

A. Production Systems

A production system is a forward-chaining reasoning system which consists of production rules and a working memory for the facts and relationships about a problem of a specific domain. Its inference mechanism will search for a solution to a problem by combining the facts and rules in its knowledge base to infer new conclusions. The working memory represents the short-term memory of the system as the set of facts may change during the operation. The production rules (or simply productions) constitute the long-term memory of the system in the form of "IF conditions THEN actions". A simple condition is a statement either propositional or predicate and when predicate, expressions with variables or wildcards satisfied by any value are allowed. The action side is a set of conclusions sometimes called results

or consequents that consist of action expressions which assert or retract facts that modify the working memory. Inference engine is the reasoning procedure, a control mechanism where usually a cycle of three steps repeats until no more rules are applicable to the working memory [10]:

- Recognition: finds which rules are applicable, that is, those rules whose antecedent conditions are satisfied by the current working memory;
- Conflict Resolution: among the rules found in the first step (called a conflict set), chooses which of the rules should "fire", that is, get a chance to execute;
- Action: changes the working memory by performing the consequent actions of all the rules selected in the second step.

If facts from the working memory match all the expressions in the condition side of a rule, the rule is called "to be satisfied" at the current matching phase. The process of matching is called unification [11]. Knowledge representation techniques based on Petri nets [12] can provide a visualization of the dynamic behavior of rule-based reasoning to designers for development and to users for validation.

B. S- Expressions

Before discussing our predicate free F-expressions, we refer to lisp S-expressions [13] not only for their historical significance and their influence to AI languages but also for their role to the theory of programming languages in general [14]. This is a suitable basis for our theoretical framework of language extensions. An S-expression is defined recursively:

- An atom is an S-expression;
- If s1, s2,...,sn are S-expressions, then so is the list (s1, s2,...,sn).

C. Predicate free Expressions (F-Expressions)

An F-expression is an ordered sequence of elements of the form $(e_1 \ldots e_n)$ separated by white-space and enclosed in parentheses, where the elements $e_1 \ldots e_n$ may be either atomic symbols or F-expressions, thus resulting in a free syntax.

F-expressions are used to represent any terms, variables, constants, functions, operations, or relations. When regarded as functions, they can have arbitrary number of arguments. F-expressions can represent complex arithmetic expressions by means of nesting. Also F-expressions can be well used as predicates, i.e. functions that always return boolean values, allowing us to write facts, rules and formulas as symbolic expressions. Third and most importantly, there is no restriction in what arguments (functions, operations or data) are passed, which makes F-expressions a vehicle for expressing higher-order abstractions.

III. Recommended Rule Dialects and Extentions

The Rule Interchange Format (RIF) Working Group [15] initially aiming to create a standard for the exchange of rules between rule systems, in particular among web rule engines, managed to provide more than just a format for rule interchange. Specifically, it has also proposed an extensibility framework which consists of RIF dialects that generally fall into two categories: logic based dialects RIF-BLD [16] and production based dialects RIF-PRD [9].

The framework for these dialects, the result of four years effort, includes a great deal of commonly used syntactic and semantic apparatus. However it purposely leaves certain parameters unspecified for designers of logic programming languages.

The Core dialect of the Rule Interchange Format, RIF-Core [17], is a subset of RIF-BLD and of RIF-PRD. It is a language of definite Horn rules without function symbols. The RIF-Core presentation syntax and semantics are specified by restriction in two different ways. First, RIF-Core is specified by restricting the syntax and semantics of RIF-BLD, and second, by restricting RIF-PRD.

IV. FELIS- A New Flexible Programming Language

We now present our contribution language, observing the RIF-Core recommendation.

A. Alphabet and Terms

The underlying syntactic theory comprises of a first-order signature without the logical predicate (and function) symbols on top of which the alphabet is defined.

The alphabet of our sample language L is defined by a signature $\Sigma = (C, V, F, Q, A, RS, AU, \{=\})$ consisting of eight classes of symbols:

- A countably infinite set of constant symbols, $C = \{c_i \mid i \in N\}$;
- A countably infinite set of variables, $V = \{x_i \mid i \in N\}$;
- A countably infinite set of externally defined function symbols, $F = \{f_i \mid i \in N\}$;
- The set of connectives $Q = \{or, and, not, then\}$;
- The set of action symbols $A = \{assert, retract, execute\}$;
- The set of reserved symbols $AR = \{if, end\}$;
- The set of auxiliary symbols $AU = \{(,), ", ?\}$;
- The equality symbol $\{=\}$.

Constants, $c \in C$, are arbitrary symbols representing objects or values in a domain, while variables, $x \in V$, are symbols (prefixed by ?) used to refer to a range of possible objects or values. There no logical predicate and function symbols in the usual first-order sense. Externally defined function symbols $\mathrm{f} \in F$ are supported to interface with *external* (non-logical) operations. Each $\mathrm{f} \in F$ has an associated arity $ar(\mathrm{f}) \in N^*$.

The language is designed to have a small core without a loss of semantic power. This is achieved by the provision of F-expressions as data structure of higher-order. Let S denote the countably infinite set of F-expressions in L. Terms of L are extensions of the specification of RIF-Core, called F-terms.

An F-term is either a constant $c \in C$, a variable $x \in V$, an F-expression, $s \in S$, of the form $(\mathrm{t}_1 \ldots t_n)$, or a function call of the form $(\mathrm{f}\ t_1 \ldots t_n)$ where $\mathrm{t}_1 \ldots t_n$ is an ordered sequence of F-term arguments with $n \geq 1$.

An F-term is called basic if it is either a constant or a variable. Each $s \in S$ has an associated arity $ar(s) \in N^*$ to indicate the number of its arguments. We define functions in L as a variation of F-expressions for invoking externally defined functions.

An F-function is an F-expression of the form $(\mathrm{f}\ e_1 \ldots e_n)$ where its first element is an externally defined function symbol $\mathrm{f} \in F$ and $n \geq 1$.

An F-term is said to be ground, if no variables occur in it. The notion of Herbrand universe is extended to the F-Herbrand universe.

The F-Herbrand universe, $F-HU$, is the set consisting of all the ground F-terms in the language L.

B. Atomic Formulas

F-expressions provide a unified way to represent both terms and atomic formulas. For this it is necessary to extend the notion of atomic formula to that of F-atomic formula.

Given arbitrary F-terms t, s and t_i where $1 \leq i \leq n$, F-atomic formulas (or F-atoms) are either F-expressions of the form $(\mathrm{t}_1 \ldots t_n)$, equality expressions $(\mathrm{t} = s)$ or action expressions $\alpha(\mathrm{t}_1 \ldots t_n)$ where $\alpha \in A$ from the signature Σ.

An F-atom which is not an equality expression neither an action expression is called basic F-atom. Non-ground F-atoms are thought of as being universally quantified. Action expressions are extra-logical constructs for accommodating actions in production rule systems. The translation of F-atoms to RIF positional atoms is shown in the following definition, by the introduction of a constant predicate symbol p.

An F-expression $(t_1 \ldots t_n)$ can be mapped into a positional RIF atom $p(t_1, \ldots, t_n)$ where p is any predicate symbol.

An F-atom is said to be ground, if each of the t_i 's is ground. A basic ground F-atom represents a fact. The definition of the Herbrand base is also extended to the F-Herbrand base.

The F-Herbrand base, $F-HB$, is the set consisting of all the basic ground F-atoms in the language L, formed using elements of the F-Herbrand universe as arguments.

To avoid obtaining logic programs with a infinite F-Herbrand base, a common practice is to restrict the language to recursively enumerable fragments for which only a limited number of nested F-terms is allowed.

C. Production Rules

An F-production rule (or rule) r in L is a conditional statement of the form r: If T then A end, where the if-part is a conditional formula and the then-part is an action block consisting of action expressions. Each action expression is an F-expression prefixed by an action symbol $\alpha \in A$ from the signature Σ.

Condition formulas are inductively defined from basic F-atomic formulas, conjunction $(\phi_1 \ldots and \ldots \phi_n)$ and disjunction $(\phi_1 \ldots or \ldots \phi_n)$ of conditional formulas or negation $(not(\phi))$, where $\phi, \phi_1, \ldots \phi_n, n > 1$ are conditional formulas.

An F-program is a pair $\Pi = \langle G, R \rangle$ where G is a finite set of ground basic F-atoms representing the set of facts, often called the fact base, and R is a set of F-production rules.

D. Operational Semantics

As in RIF-PRD, the semantics of our language are operational. A framework that gives both an operational and model-theoretic semantics to production rules was studied by [18].

A condition formula ϕ of a rule r, either atomic or compound, is evaluated with respect to the state of the fact base. The evaluation is performed in terms of matching substitutions and if successful (ϕ is satisfied), the rule is called instantiated. In the following we give the notions of substitution, unification and matching substitutions for the language L.

Let T be the universe of F-terms of the language L. A substitution is a finitely non-identical assignment of F-terms $\mathrm{t}_i \in T$, to variables $x_i \in V$, written as $\sigma = \{\mathrm{t}_i / x_i\}_{i=1\ldots n}$ where $Dom(\sigma) = \{x_i\}_{i=1\ldots n}$ and $\sigma(x_i) = \mathrm{t}_i, i = 1 \ldots n$.

A substitution σ is called ground when it assigns only ground terms to variables in $Dom(\sigma)$. The unification two F-terms is recursive. If the terms in both F-terms are basic (constants or variables), the unification of each pair of terms from the two F-terms consists just of matching the basic terms. If terms are themselves nested F-terms then the procedure for unifying F-terms is re-entered. The recursion is terminated when the terms of both F-terms are all basic.

The unification of two F-terms $t_A, t_B \in T$ is a most general substitution σ such that one of the following is true:

Case 1. If $t_A, t_B \in C \cup V$ then $\sigma(t_A) = t_B$ if $t_A \in V$ and $\sigma(t_A) = t_A$ otherwise;

Case 2. If $t_A, t_B \in S$ then goto Case 1.

It may happen that a term is a variable and the term in the second F-terms is a nested F-expression. In this case, the F-expression is get instantiated on the variable in the corresponding term of the first F-term. Next, we define the matching substitution of F-atomic formulas.

Let ϕ be a condition formula or a rule r; let Var be a function that maps a condition formula to the set of its free variables; let σ be a ground substitution such that $Var(\phi) \subseteq Dom(\sigma)$; and let G be a set of ground F-atomic formulas. We say that the ground substitution σ matches ϕ to G if and only if one of the following is true:

- ϕ is a basic F-atomic formula and $\sigma(\phi) \in G$;
- ϕ is an equality formula $(t_1 = t_2)$ and the ground terms $\sigma(t_1)$ and $\sigma(t_2)$ have the same value;
- ϕ is $not(\psi)$ and σ does not match the condition formula ψ to G;
- ϕ is a conjunction $(\psi_1 \ldots and \ldots \psi_n), n > 0$ and $\forall i, 1 \le i \le n$, σ matches ψ_i to G;
- ϕ is a disjunction $(\psi_1 \ldots or \ldots \psi_n), n > 0$ and $\exists i, 1 \le i \le n$, such that σ matches ψ_i to G.

E. Conflict Resolution

If the conditions of more than one rule are satisfied at the same time, the pattern matching of production rules leads to a set instantiated rules , called the conflict set. The conflict set is formally defined from the set of production rules R and the state of the fact base (or working memory).

We call working memory state, denoted by w_G, a set of ground F-atomic formulas G; i.e. a subset of the F-Herbrand universe defined on the signature Σ.

For a given working memory state w_G and a set of production rules R, the set $\{r_i \in R \mid \exists \Phi_i \Rightarrow A_i\}_{i=1\ldots r} \subset R$ which is w_G -fireable is called w_G -conflict set and is denoted by $w_G CS_R = \{f_1, \ldots, f_k\}$.

When several rules are found to be instantiated at the same time, a conflict resolution strategy is used to select which rule to apply.

A resolution strategy is a computable function that given a w_G -conflict set of rules R, returns a unique element of the $w_G CS_R$ set.

A specification of the conflict resolution strategy is currently missing from RIF-PRD. A formalization is provided in [19] showing how to specify conflict resolution strategies in Answer-Set Programming (ASP) and illustrate a precise encoding of the RIF-PRD strategy.

F. Atomic Atoms

The language defines several atomic actions for updating the fact base or execute code:

- Assert fact: If ϕ is an F-atom, then Assert(ϕ) is an action expression;
- Retract fact: If ϕ is an F-atom, then Retract(ϕ) is an action expression;
- Execute: if ϕ is an F-atom, then Execute(ϕ) is an action expression.

G. Production Rule System

A production rule system can be semantically defined as a labeled terminal transition system.

A production rule system is a tuple $\langle SS, A, \rightarrow_{TR}, FS \rangle$, where SS is a set of working memory states (or system states), A is a set of transition labels, where each transition label is a sequence of ground action expressions, a set of transition relations $\rightarrow_{TR} \subseteq SS \times A \times FS$ such that $(ss, \alpha, ss') \in \rightarrow_{TR}$ if and only if there is a transition labeled α from the state ss to the state ss', $FS \subset SS$ is a set of final system states where $FS = ss \in SS \mid \forall \alpha \in L, \forall ss' \in SS, (ss, \alpha, ss') \notin \rightarrow\}$.

V. EXAMPLES

Various examples follow to clarify the capabilities of our proposed language.

A. F-Expressions as a Variable

(Aristotle was a (famous Greek natural philosopher))
(Asclepius was a (doctor of medicine))
(Socrates was a philosopher)
(Parmenion was a (Macedonian general))
(Aristotle was willing to teach)
(Asclepius was willing to teach)
(Socrates was willing to teach)
(Alcibiades was willing to teach)
if (?X was a ?Y) and (?X was willing to teach)
then (?X has been a good teacher because he was ?Y)
Output:
(Aristotle has been a good teacher because he was (famous Greek natural philosopher))
(Asclepius has been a good teacher because he was (doctor of medicine))
(Socrates has been a good teacher because he was philosopher)

B. Variables inside an F-Expression

(Aristotle was a (Greek philosopher))
(Spinoza was a (Dutch philosopher))
(Asclepius was a (Greek doctor))
if (?X was a (?Y ?Z)) and (?Y = Greek)
then (?X lived in Greece)
Output:
(Aristotle lived in Greece)
(Asclepius lived in Greece)

C. Nested F-Expressions as Facts

(The book ("Dr No" of 210 pages) is written by (author (Fleming British journalist)))
(The book ("Moby Dick" of 600 pages) is written by (author (Melville American novelist)))

D. Various Fact Types

("Bob Dylan" plays (the guitar))
("Bob Dylan" plays (the red guitar))
("Bob Dylan" plays (the red guitar made in 1969))

E. Free Format

("Da Vinci" painted (the portrait of "Mona Lisa") in 1506)
(Picasso painted (the Guernica) in 1937)
("Van Gogh" painted (the self portrait) in 1888)
IF (?anypainter painted ?anything in ?anyyear) and (?anyyear 1900)
THEN (?anything of ?anypainter has historical value)
Output:
((the portrait of "Mona Lisa") of "Da Vinci" has historical value)
((the self portrait) of "Van Gogh" has historical value)

F. Higher-Order Logic

(Socrates was wise)
(Socrates had (great fame))
(Plato wrote dialogues)
(Socrates respected law)
if (?A ?Z ?B) and (?Z <> respected) then (?Z (?A ?B))
Output:
(Socrates (was wise))
(Socrates(had (great fame)))
(Plato (wrote dialogues))

G. Extension to Fuzzy Domain

An Uncertainty Rule Dialect (RIF-URD) based on RIF-BLD was proposed in [20]. Below is an example of how to declare fuzzy linguistic variables, fuzzy facts and finally how to syntax fuzzy rules, using FELIS language. The deails of the fuzzy extension module will be given in a future publication.
##Fuzzy Linguistic Variables
(probability < less certain, certain, more certain >)
(sunshine < less sunshine, sunshine, more sunshine >)
##Fuzzy Facts
(there is < sunshine 73 percent >)
##Fuzzy Rules
IF (there is < more sunshine >) then (it is < more certain > that we will have a barbeque)
IF (there is < sunshine >) then (it is < certain > that we will have a barbeque)
IF (there is < less sunshine >) then (it is < less certain > that we will have a barbeque)
etc.

VI. Discussion of Research

We see that predicates are removed from the signature of the language and F-expressions become the vehicle to represent either a function, a predicate, a data list and not only, considering the extensibility achieved. We eliminate restrictions of existing rule languages (OPS5 [21], CLIPS [22], Jess [23], RIF-PRD [9]). OPS5, an early production system developed by Charles Forgy, represents facts as a list of attribute-value pairs. On the other hand, CLIPS or its extension Jess adopt a lisp-like parenthesized syntax. They possess two kinds of facts: ordered facts and defined templates. CLIPS does not support nested lists. Indeed, each nested expression is not an arbitrary sublist but an attribute/value pair. Finally, the very recent RIF-PRD dialect has a presentation syntax where a fact is represented by a positional atomic formula. The syntax of RID-PRD for atomic formulas resembles PROLOG's clausal form which is known to lack higher-order capability, i.e. quantification over predicate and function symbols is not allowed.

Our current work on production rule systems is a characterization of general production systems by proposing a rule language that enjoys the following features:

- Facts are expressed in free format as unstructured ordered lists (Example E);
- F-expressions can be assigned to variables (Examples A,D);
- Variables may appear inside an F-expression (Example B);
- Arbitrary depth nesting (Example C);

- Quantification over predicate and function variables is possible, leading to higher-order unification (Example F);
- Extensibility to fuzzy domain (Example G).

A substantial interest exists in integrating aspects of functional languages into logic programming **Error! Reference source not found.**. Despite of the wide research available in these multi-paradigm languages, to our knowledge, the production rule languages widely used in expert systems have not followed up this trend. One reason is perhaps that production systems have not been considered to be a fully declarative logic paradigm, due to additional "impure" features usually added in their implementations. We have seen that F-expressions as a single general kind of expression, compared to other knowledge representations such as horn clauses and attribute-value pairs, are much flexible and yet constitute a richer domain.

VII. Conclusion and Future Research

We have set the foundational work for building a flexible production rule language which is very close to a natural language, as seen in the provided examples. The main vehicle introduced is F-expressions which are permitted to appear as terms of atomic formulas or assigned to variables resulting in simple representation of complex structures. This framework is extensible to higher-order programming, permitting quantification over predicate and function variables. The proposed knowledge representation scheme gives to the community guidelines for the next generation of programming.

Our prototype production system is part of a broad research project "Applications of fuzzy information systems in engineering and management" that involves a great number of aspects besides formal language definition including pattern matching, conflict resolution, inference mechanism and interoperability with other rule systems.

References

[1] M. Kifer, "Rule interchange format: The framework," Web Reasoning and Rule Systems, vol. 19(1-3), pp. 583-628, 2008.

[2] Documents associated with Production Rule Representation (PRR) Version 1.0 Release date: December 2009. http://www.omg.org/spec/PRR/1.0/

[3] M. Hanus, "The integration of functions into logic programming: From theory to practice," The Journal of Logic Programming, vol. 19(1-3), pp. 583-628, 1994.

[4] H. Cirstea, C. Kirchner, M. Moossen, and P. E. Moreau, "Production systems and rete algorithm formalisation," The Journal of Logic Programming, vol. 19(1-3), pp. 583-628, 2004.

[5] L. Naish, "Higher-order logic programming in Prolog," Proc. Workshop on Multi-Paradigm Logic Programming, JICSLP, vol. 96, pp. 1-23, 1996.

[6] W. Chen, M. Kifer, M., and D. S. Warren, "HiLog: A foundation for higher-order logic programming," The Journal of Logic Programming, vol. 15(3), pp. 187-230, 2004.

[7] G. Nadathur, "The metalanguage λ Prolog and its implementation," Functional and Logic Programming, vol. 15(3), pp. 187-230, 2001.

[8] Z. Somogyi, F. Henderson, and T. Conway, "The execution algorithm of Mercury, an efficient purely declarative logic programming language," The Journal of Logic Programming, vol. 29(1-3), pp. 17-64, 1996.

[9] RIF Production Rule Dialect. W3C Recommendation (June 22, 2010), http://www.w3.org/TR/2010/REC-rif-prd-20100622/

[10] R. J. Brachman, H. J. Levesque, Knowledge representation and reasoning, Morgan Kaufmann Pub, 2004, pp.119-120.

[11] J. A Robinson, "Computational logic: The unification computation," In Machine Intelligence, vol. 6, B. Meltzer and D. Michie (Eds.). Edinburgh Univ. Press, Edinburgh, Scotland, 1971, pp. 63-72.

[12] M. Tavana, "Knowledge-Based Expert System Development and Validation with Petri Nets," Journal of Information and Knowledge, vol. 19(1-3), pp. 161-170, 2008.

[13] R. Rivest, S-Expressions. Network Working Group Internet Draft. http://people.csail.mit.edu/rivest/Sexp.txt, 1997.

[14] H. Barendregt, "The impact of the lambda calculus in logic and computer science," Bulletin of Symbolic Logic, vol. 29(1-3), pp. 181-215, 1997.

[15] RIF W3C Working Group, Wiki (last modified on 14 October 2010) http://www.w3.org/2005/rules/wiki/RIF_Working_Group

[16] RIF Basic Logic Dialect. W3C Recommendation (June 22, 2010), http://www.w3.org/TR/2010/REC-rif-bld-20100622/

[17] RIF Core Dialect, W3C Recommendation (22, June 2010), http://www.w3.org/TR/2010/REC-rif-core-20100622/

[18] R. Kowalski, F. Sadri, "Integrating Logic Programming and Production Systems in Abductive Logic Programming Agents," Web Reasoning and Rule Systems, pp. 1-23, 2009.

[19] C. V. Damasio, J. J. Alferes, and J. Leite, "Declarative semantics for the rule interchange format production rule dialect," ISWC 2010, Part I, LNCS vol. 6496, pp. 798-813, 2010.

[20] J. Zhao, H. Boley, "Uncertainty treatment in the rule interchange format: From encoding to extension," Proceedings of the 4th International Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2008) vol. 29, pp. 17-64, 2008.

[21] C. L. Forgy, OPS5 user's manual. Technical Report CMU-CS-81-135, Carnegie Mellon University, 1981.

[22] Artificial Intelligence Section, CLIPS Reference Guide Vol. I and II, Lyndon B. Johnson Space Center, June 2 1993.

[23] E. Friedman-Hill, Jess in action: rule-based systems in java, MANNING, 2003.