TR-693

Logic-oriented Inferential Framework Extensions LIFE-Ω

by
J. Yamaguchi (Kanagawa Univ.)

September, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191 ~ 5
Telex ICOT J32964

Logic-oriented Inferential Framework Extensions LIFE-Ω

by

Jinsei Yamaguchi

Dept. of Information and Computer Sciences

KANAGAWA UNIVERSITY

2946 Tsuchiya, Hiratsuka

Kanagawa 259-12, JAPAN

Abstract: We give the philosophical background of LIFE-Ω, a new paradigm for AI. We begin our argument by listing some properties LIFE-Ω can provide as a paradigm. Then, we present a Boolean algebraic semantics of pure Prolog as a language scheme and relativize this idea within LIFE-Ω to get a procedural semantics of LIFE- I , a Boolean-valued logic programming language scheme. By generalizing the idea further within LIFE-Ω, we can propose other language schemes for AI as extended versions of LIFE- I .

Key words: logic programming, Boolean-valued model, inference, fuzzyness.

§0. Introduction

There are many concepts AI scientists have been trying to capture within their own frameworks or more specifically by their programming languages. Among them are those symbolized by the terminologies "uncertainty", "negation," "learning", "common sense" and "creativity" etc. For example, concerning the notion of uncertainty, many challenged through a variety of approaches. Some generalized the notion of truth value to many-valuedness and some others borrowed such techniques as probability, modality and fuzzyness etc [3], [6], [10]. Here, we would like to propose a paradigm for AI which, we believe, may contribute not only to the notion of uncertainty but also to those cited above. We call this paradigm "LIFE-Ω", the naming comes from "Logic-oriented Inferential Framework Extensions—infinite series" or "Logic-oriented Intelligence for Future Environment." To constitute the paradigm, we have borrowed and transformed the idea of Boolean-valued universe in set theory. In section 1, we see some properties LIFE-Ω can provide as a paradigm. Then, we present a Boolean algebraic semantics of pure Prolog as a language scheme in section 2 and in section 3, we relativize this semantics to get the procedural semantics of LIFE- I (Logic-oriented Inferential Framework Extension—class I), a Boolean-valued logic programming language scheme which belongs to the paradigm LIFE-Ω. In section 4, by generalizing the idea farther, we propose some other possible categories, on the ground of which we can construct a series of language schemes LIFE-X ($\mathrm{II} \leqq X < \infty$) as extended versions of LIFE- I . The theoretical background of the first three schemes LIFE- I , LIFE-II and LIFE-III is discussed in [11].

The broadness of the topics which LIFE-Ω might cover states the fact that this is rather a meta-scheme than a particular scheme. This fact is partially supported by the idea of Boolean-valued universe itself. The most crucial point concerning Boolean-valued technique is that this is not a mere aim to represent many-valuedness of our belief but rather a means to express the semantical idea

of Forcing methodology and so is different from the other many-valued logics from the viewpoint of possible applications. [5] is a standard textbook for those who are not so familiar with the notions of Boolean-valued model, Forcing etc in set theory.

§1. The Idea of LIFE-Ω

In the field of set theory, there is a notion called "Boolean-valued universe". This notion is the semantical version of Forcing technique and has a resemblance with fuzzy set theory to the extent that both essentially treat many-valuedness of membership relations, though they are different in many important ways. The idea of LIFE-Ω springs from the methodology of Boolean-valued universe. In the following, let's list up typical properties that LIFE-Ω can serve when it is applied to the world of logic programming. For those who are familiar with Boolean-valued model in set theory, the philosophical and methodological similarity must be obvious.

1. Employ a complete Boolean algebra B and generalize the notion of truth-value from 2 to B. By doing so, contribute to the notion of negation.
2. Using B, extend the world of logic programming to the world of Boolean-valued logic programming within the paradigm of LIFE-Ω.
   The extended framework in the most simple case may be illustrated as the following figure [1].

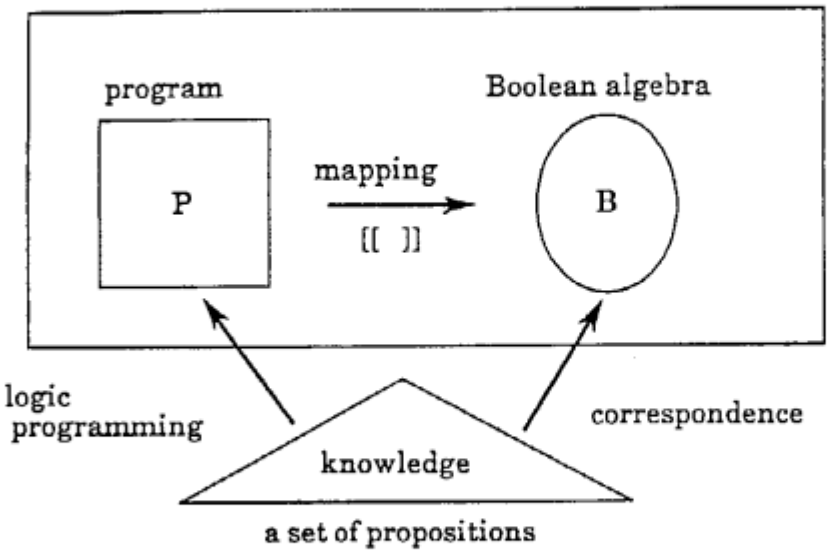
program
Boolean algebra
P
mapping
[[ ]]
B
logic
programming
correspondence
knowledge
a set of propositions

Figure [1]

3. Given a logic programming language scheme for AI, we can obtain a Boolean-valued logic programming language scheme LIFE-X by applying the Boolean-valued technique considered in the paradigm LIFE-Ω.

4. To define procedural semantics of LIFE-X ($I \leqq X < \infty$), essentially follow the original construction method used in a (2-valued) logic programming language scheme except generalizing the technique used at each derivation step.

5. By choosing $B \neq B'$, we can construct totally different $(P, [\![\ \ ]\!]_B)$ and $(P, [\![\ \ ]\!]_{B'})$. The choice of B depends on which knowledge we want to formalize. The crucial point is that, firstly choose the concrete knowledge K we want to formalize, then construct a suitable Boolean algebra B which is hoped to reflect the essence of K. Next, construct $(P, [\![\ \ ]\!]_B)$ using this B and check whether this $(P, [\![\ \ ]\!]_B)$ can serve to represent the wanted knowledge K. If $(P, [\![\ \ ]\!]_B)$ can represent K usefully, then o.k. Else, construct a different B' to apply as the second candidate.

6. The purpose of our considering Boolean-valued logic programming (language) scheme LIFE-X is to manage new environments which usual logic programming language scheme can't attain or (by deciding a concrete language based on it) to formulate elegantly some knowledge which may (or may not) be expressed by the usual (2-valued) language but need an elaboration.

7. Let G be a goal for a logic program P and suppose there is an output λ of P∪{G}. Then, there is a certain kind of output ξ_λ of P∪{G} with respect to $(P, [\![\ \ ]\!])$ with the Boolean value $v(\xi_\lambda) \in B$. The point is that $v(\xi_\lambda)$ is a new output which assert something different from the original answer λ concerning G. (ξ_λ may be or may not be λ.) Since $[\![\ \ ]\!]$ depends on B, there is a possibility that $v(\xi_\lambda)$ has a new important meaning for a certain B. Of course, there should be some connection between G and $v(\xi_\lambda)$. After all, whenever we have an output λ of P∪{G}, by using Boolean-valued method,

we can automatically obtain the output $\upsilon(\xi_\lambda)$ for each different B we choose.

8. There are cases that we can obtain meaningful outputs of P∪{G} w.r.t. (P, [[ ]]) for certain kinds of goal G and map [[ ]], though there is no substantial output of G w.r.t. P.

9. The method can be iterated, and a variety of techniques can be used for the iteration.

In the following, as a simplest example that we can obtain within the paradigm LIFE-Ω, we briefly sketch the procedural semantics of Boolean-valued logic programming (language) scheme LIFE- I , which is a direct generalization of pure Prolog scheme. Before doing so, let's reinterprete the procedural semantics of pure Prolog from a Boolean algebraic viewpoint.

§2. Boolean algebraic semantics of SLD-resolution
——a preparation for LIFE- I ——

Let P be a pure Prolog program and $G = \leftarrow A_1, \cdots, A_k$ be a goal. Suppose there is a SLD-refutation R of P∪{G} with the sequence of mgus $<\theta_1, \cdots, \theta_n>$ and the sequence of input clauses $<C_1, \cdots, C_n>$. Then, we can interpret this refutation procedure from a viewpoint of 2-valued Boolean algebra in the following way.

I . There is a map [[ ]]:$B_P \to 2 = \{1, 0\}$ such that, for any clause C in P,

[[∀C]] = 1 , i.e., $\bigwedge_{i \in I}$ [[$C\rho_i$]] = 1, where $\{\rho_i \mid i \in I\}$ is the set of all ground substitutions for C.

This means, of course,

$$\bigwedge_{i \in I} ([[\, C^+\rho_i \,]] \leftarrow [[\, C^-\rho_i \,]]) = 1$$

i.e.,

$[[\ C^+\rho\ ]] \geqq [[\ C^-\rho]]$ for any ground substitution ρ for C, ……①

where $[[\ C^-\rho]] = [[\ B_1\rho\ ]] \wedge \cdots \wedge [[B_n\rho\ ]]$ if C has the form $C^+ \leftarrow B_1, \cdots, B_n$. So, especially in case of $C^- = \varnothing$, that is , in case that C is an unit clause, ① becomes the form

$[[\ C^+\rho\ ]] = 1$ for any ground substitution ρ for C.

Roughly speaking, this Boolean interpretation of P means that any clause C in P represent either a true assertion or a true rule with respect to [[ ]]:$B_P \rightarrow 2$. In other words, $(U_P, [[\ \ ]])$ becomes a Herbrand model of P where, as usual, U_P is the Herbrand universe of P and the assignment is defined by $(U_P, [[\ ]]) \models \forall C$ iff $[[\forall C\ ]] = 1$.

So, we can say that the mapping [[ ]]:$B_P \rightarrow 2$ should be determined so that the program P becomes a consistent formal theory from a semantical viewpoint.

(Usually, the least Herbrand model of P implicitly plays the role of [[ ]], though the originally intended model may be different.)

II. By refutation, when R ends with the empty clause $\square$, we get the following sequence of (in)equalities.

By definition of unification.

$$[[\ A_1\theta\rho\ ]] \wedge \cdots \wedge [[\ A_k\theta\rho\ ]]$$

② $\rightarrow \geqq$ $[[\ A_1\theta\rho\ ]] \wedge \cdots \wedge [[\ A_{m-1}\theta\rho\ ]] \wedge [[C_1^-\theta\rho]] \wedge [[\ A_{m+1}\theta\rho\ ]] \wedge \cdots \wedge [[\ A_k\theta\rho\ ]]$

:

:

$\geqq 1$,

where $A_m \in G$ is the selected atom for θ_1, $\theta = \theta_1 \cdots \theta_n$ and ρ is a suitable ground substitution. For ②, we use the fact ①. Ending with the empty clause means we must substitute each Boolean element in the right hand side eventually by a

Boolean value of a certain unit clause (groundly instanciated by $\theta\rho$), i.e., 1. So, we get the last inequality "$\geqq 1$".

Now, deducing the empty clause from $P\cup\{\leftarrow A_1\theta\rho, \cdots, A_k\theta\rho\}$ means $P\cup\{\leftarrow A_1\theta\rho, \cdots, A_k\theta\rho\}$ is inconsistent from a viewpoint of Gentzen style proof theory. Translating this fact to Hilbert style, we get the fact that $P\cup\{\neg(A_1\theta\rho\wedge\cdots\wedge A_k\theta\rho)\}$ becomes inconsistent as an axiomatic formal theory. Since we already know that $(\forall C\in P)$ ($[\![\forall C]\!] = 1$) by definition of $[\![\ \ ]\!]$: $Bp\rightarrow 2$, in order to get a contradiction,

$[\![\neg (A_1\theta\rho\wedge\cdots\wedge A_k\theta\rho)]\!]$ should be 0.

So, $[\![A_1\theta\rho]\!]\wedge\cdots\wedge [\![A_k\theta\rho]\!]= 1$.

This is what the above sequence of inequalities asserts. From now on, under this semantics, let's define

pure Prolog = LIFE-0

as a Boolean-valued logic programming language scheme for the sake of the conceptual consistency.

§3. Boolean-valued logic programming language scheme LIFE- I

——a sketch of its procedural semantics——

In this section, as a direct relativization of 2-valued Boolean algebraic semantics of SLD-resolution, we give a brief sketch of the procedural semantics of LIFE- I . (In the literature, this scheme is called "Boolean-valued Prolog (of the first kind)".) The precise definitions and some related topics including both soundness and completeness are discussed in [11].

First of all, we generalize 2 to a complete Boolean algebra B. Then, choose a complete filter F over B. Let P be a Prolog program and $G=\leftarrow A_1, \cdots, A_k$ be a goal. Suppose there is a SLD-refutation R of $P\cup\{G\}$ with the sequence of mgus $<\theta_1,\cdots, \theta_n>$ and the sequence of input clauses $<C_1,\cdots, C_n>$. Then, we can

interpret this refutation procedure in the following way from a viewpoint of B-algebra.

I. There is a map [[ ]] : $B_P \to B$ such that, for any clause C in P,

$$[[\forall C]] \in F, \text{ i.e., } \bigwedge_{\rho:\text{ground}} ([[C^+\rho]] \leftarrow [[C^-\rho]]) \in F. \qquad \cdots ①$$

So, especially in case of $C^- = \varnothing$, ① becomes the form

$$\bigwedge_{\rho:\text{ground}} [[C^+\rho]] \in F.$$

Roughly speaking, this Boolean interpretation means that any clause C in P represents either a true assertion or a true rule to the extent of F over B with respect to [[ ]] : $B_P \to B$. In other words, (U_P, [[ ]]) becomes a B-valued Herbrand model of P modulo F.

II. When R ends with the empty clause $\square$, we naturally get the two Boolean values

$$v(\theta) = \bigwedge_{\rho:\text{ground}} ([[A_1\theta\rho]] \wedge \cdots \wedge [[A_k\theta\rho]]) \qquad \cdots ②$$

and

$$v(R) = (\bigwedge_{\rho:\text{ground}} [[C_1{}^+\theta_1\rho]]) \wedge \cdots \wedge (\bigwedge [[C_n{}^+\theta_n\rho]]) \qquad \cdots ③$$

where $\theta = \theta_1 \cdots \theta_n$.

LIFE- I is the class of all languages having the organization $< (P, [[\ ]]), B, F >$ which use the similar derivation method to pure Prolog and evaluate the resulting values $v(\theta)$ and $v(R)$ at the end of each refutation, or more precisely, calculate

$$\bigwedge_{\rho:\text{ground}} [[C_i{}^+\theta_i\rho]] \qquad \text{for} \quad 1 \leqq i \leqq n$$

at each unification step. Here an observation gives us the fact that

$$v(R) \leqq v(\theta)$$

and

$$v(\theta) \in F \qquad \cdots ④$$

The latter result is a straightforward relativization (modulo F)of the 2-valued case discussed in the previous section. Here, let's call a Boolean-valued program $(P, [\![\ \ ]\!])$ "a F-program" iff $(U_P, [\![\ \ ]\!])$ becomes a B-valued Herbrand model of P modulo F. Then the fact that

$(P, [\![\ \ ]\!])$ is a F-program $\Rightarrow v(\theta) \in F$

shows the relativized soundness property of Boolean-valued derivation. For precise relations between this fact and the original soundness property of SLD-resolution, the reader are recommended to consult [11]. Here a natural question is "Is there any relation between the notion of F-program and $v(R)$?"

The positive answer is also found in [11] as a strong soundness property. Moreover, a variety of Boolean-valued versions of completeness properties are discussed in [11], [12], [15], too.

§4. Generalizations

The Boolean-valued refutation discussed in the previous section depends on the syntactical unification in the usual sense, that is,

θ unifies A and B iff $A\theta = B\theta$.

Let's call this sort of unification with additional computation $\bigwedge_{\rho:\text{ground}} [\![\ A\theta\rho\ ]\!]$ "Boolean-valued unification of the first kind."

Now, by using $[\![\ \ ]\!] : B_P \rightarrow \mathbf{B}$, we can generalize the notion of the 1st kind unification and enter a new derivation world where conventional unification methods have never experienced by themselves. For example, we can define

θ is a higher kind unifier for A and B iff

$[\![\ A\theta\rho\ ]\!] = [\![\ B\theta\rho\ ]\!]$ for any ground substitution ρ for $A\theta \leftrightarrow B\theta$, where $A\theta$ and $B\theta$ satisfy a certain unifying condition concerning a relation $\sim$ over the class of all (not necessarily ground) atoms generated from P. What we really

perform by employing this kind of abstract unification is that we devide Bp into a set of classes using both~ and [[ ]]. In [11], we discuss two simple examples of this kinds of generalized unification and show that the generalized Boolean-valued derivation still preserve both the soundness and the completeness properties in their suitable sense. Of course, the purpose of our employing this kind of generalized derivation is that we would like to obtain the case that, for a suitable program P and a goal G, there is a higher kind refutation of P∪{G} with respect to a suitable *F*-program (P, [[ ]]), though there is no SLD-refutation of P∪{G} at all. (If there is a SLD-refutation of P∪{G}, then there always exists any higher kind of refutation of P∪{G} with respect to any Boolean-valued program (P, [[ ]]).) As a consequence, a higher kind unification can influence CWA, inductive inference, universal unification, qualitative deduction, quantitative deduction or Fuzzy inference, semantical negation, paraunification including non-transitive deduction, etc.

So far, we have been considering only the unification-generalizing direction starting from LIFE- I which depends on the organization < (P, [[ ]]), B, *F*, ~ >, where P is a Prolog-type program, that is, a set of Horn clauses. Another interesting direction is the one which generalize the type of P. Coming soon LIFE-IV belongs to this category. As the basic type of P, we will allow the set of Horn clauses with constraints which is the most simple but rather useful generalizing way. However, there is no reason we should restrict our attension only to those specialization of LIFE-Ω whose object-level program P has the type of Prolog-extension (and so whose derivation kind depends on unification technique).

Farthermore, we can choose the third route to generalize LIFE- I , whose signpost shows the notice stated in 9 in section 1. The shortest distance to reach the destination is to choose finite different Boolean algebras B_1, ···, B_n at the same time for one object-level program P and consider independent maps $[[\ ]]_i$:

$Bp \rightarrow B_i$ $(1 \leqq i \leqq n)$. Any concrete logic programming language belongs to thus obtained scheme $< (P, [[\ \ ]]_i), B_i, F_i, \sim_i >_{1 \leqq i \leqq n}$ will be able to be managed by the meta-level parallel processing method in an essential manner.

The fourth generalization depends on the choice of B. Instead of choosing B as a set of simple structured symbols, objects or propositions, we can make use of even more abstract and complex partially ordered structures as candidates of B. If we employ this sort of complex-structured B, we ought to regard the resulting value $v(\xi_\lambda)$ as a new object to be analized by the subsequent inferential steps or by object-oriented manners.

Of course, there should be many other possible paths which generalize the idea of LIFE- I within the paradigm LIFE-Ω. Moreover, we must be able to exploit totally different frontiers from those cultivated by the tools of logic programming, because the spirit of LIFE-Ω should be as free as human intelligence. Thus, the truely new formulation of knowledge we can't capture today shall be done under the flag of LIFE-Ω in future, we hope.

§5. Conclusion

In this paper, we have presented the philosophical background of LIFE-Ω as a (language) paradigm for AI. LIFE-Ω is the transcendental notation of the series LIFE-X ($1 \leqq X < \infty$). Each LIFE-X is a logic-oriented inferential framework having an abstract organization O with an inferential method D based on O. As discussed in this paper, for some X, O may have the form $<(P, [[\ \ ]]), B, F, \sim, \cdots>$, where P is an object level program, B is a complete Boolean algebra, F is a complete filter over B, $[[\ \ ]] : Bp \rightarrow B$ is a map and $\sim$ is a relation over the set of all atoms generated from P, etc. Though D is a purely logical derivation, each computation concerning $[[\ \ ]]$, F, $\sim$, etc may depend on other programming technique. In this sense, each concrete programming language belonging to a certain LIFE-X will be the fusion of different programming tastes.

This is the reason why we employ the terminology "logic-oriented" instead of "logical". However, at the same time, this terminology suggests that the hero of LIFE-X should be the logical and other approaches are byplayers.

An example of our language scheme LIFE- I is the language "ProBoole" being developed at IBM Tokyo Research Laboratories. (See [9] for an application of ProBoole.)

In addition, many languages so far exist can be reinterpreted by Boolean-valued technique and so belong to our scheme LIFE-X for a certain $I \leqq X < \infty$. Here, we dare say that the generalizing methods we present above have not yet exhausted the possible schemes in LIFE-Ω. From the nature, it is the destiny of the scheme LIFE-X ever to evolve in order to cover new topics in the field of AI and to be combined with related (language) schemes in the ocean of AI frameworks until a completely new methodology which exceeds this paradigm will appear someday in future.

References

[1] K.A. Bowen and R.A. Kowalski, "Amalgamating Language and Meta language in Logic Programming," in : K.L. Clark and S-A. Tärnlund (ed.), Logic Programming (Academic Press 1982), 153-172.

[2] M. Fitting, "A Kripke-Kleene Semantics for Logic Programs", J. Logic Programming, vol. 2 (1985), 295-312.

[3] M.I. Ginsberg, "Multi-valued Logics", Proc. AAAI-86 (1986), 243-247.

[4] J. Jaffar, J-L. Lassez and M.J. Maher, "Some Issues and Trends in the Semantics of Logic Programming", The 3rd International Conference on Logic Programming in: E. Shapiro (ed.) Lecture Notes in Computer Science 225, (Springer Verlag 1986), 223-241.

[5] T.J. Jech, "Set Theory", Academic Press, 1978.

[6] W. Marek, "A Natural Semantics for Modal Logic over Databases and Model-theoretic Forcing I", Non-monotonic Reasoning Workshop by AAAI (1984), 194-240.

[7] C.S. Mellish, "Abstract Interpretation of Prolog Prograṃs", Proc. 3rd International Conference on Logic Programming, in : E. Shapiro (ed.), Lecture Notes in Comptuer Science 225 (Springer Verlag 1986), 463-474.

[8] D.A. Millar and G. Nadathur, "Higher-order Logic Programming", *ibid.*, 448-462.

[9] S. Morishita, M. Numao and S. Hirose, "Symbolical Construction of Truth-value Domain for Logic Program", Proc. of the 4th ICLP (1987), 533-555.

[10] N.J. Nilsson, "Probabilistic Logic", Artificial Intelligence, vol. 28(1986), 71-87.

[11] J. Yamaguchi, "Boolean-valued Logic Programming Language Schemes : LIFE- I , LIFE-II, LIFE-III —Theoretical Background—", NEC, C&C Systems Research Laboratories, LR-5197 .