ICOT Technical Report: TR-639

TR-639

Boolean-valued Logic Programming Language Scheme LIFE-III

by
J. Yamaguchi (Kanagawa Univ.)

April, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

Boolean-valued Logic Programming Language Scheme LIFE-Ⅲ —A Theoretical Background—

by

Jinsei Yamaguchi

Department of Information and computer Sciences
KANAGAWA UNIVERSITY

Abstract: We define the notion of logic-oriented inferential framework extension LIFE-Ⅲ as an AI language scheme and prove the theoretical background. Especially, the concept of Boolean-valued relativized completeness, which will play an important role in many fields of AI, is proposed and it is systematically studied that what kinds of relativized completeness are satisfied by LIFE-Ⅲ.

Key Words: AI, Logic programming, Boolean-valued semantics, Boolean-valued unification, Boolean-valued relativized completeness

Chapter 0
Preface

§0-0. 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. 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 ($II \leqq X < \infty$) as extended versions of LIFE- I . The theoretical background of the third scheme LIFE-III is the main topic in this paper.

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. [18] is a standard textbook for those who are not so familiar with the notions of Boolean-valued model, Forcing etc in set theory.

§0-1. 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].

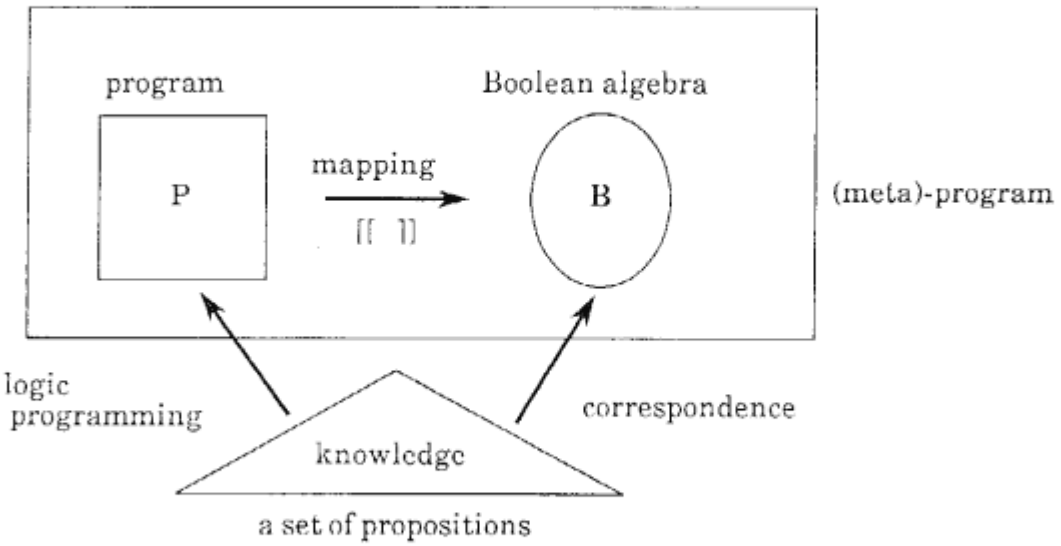

program
Boolean algebra
P
mapping
[[ ]]
B
(meta)-program
logic
programming
correspondence
knowledge
a set of propositions

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 $P \cup \{G\}$. Then, there is a certain kind of output ξ_λ of $P \cup \{G\}$ 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 $P \cup \{G\}$, by using Boolean-valued method,

we can automatically obtain the output $v(\xi_\lambda)$ 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 in the 2-valued sence.

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.

§0-2. 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 [[ ]]:Bp$\rightarrow$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, $(U_P, [[\quad]])$ becomes a Herbrand model of P where, as usual, U_P is the Herbrand universe of P and the assignment is defined by $(U_P, [[\ ]]) \vdash \forall C$ 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 $[[\quad]]$: Bp—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 Boolean-valued logic programming language scheme for the sake of conceptual consistency.

§0-3. 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 [33].

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

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

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

$v(\theta) \in F$ ···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 chapter I. 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 chapter I as a strong soundness property. Moreover, a variety of Boolean-valued versions of completeness properties are discussed later, too.

§0-4. 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 $[\![\ \ ]\!] : B_P \to \mathbf{B}$, 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 set

of all (not necessarily ground) atoms generated from P. What we really perform by employing this kind of abstract unification is that we devide B_P into a set of classes using both~ and [[ ]]. In this paper, we discuss one simple example 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 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. A few examples are discussed later.

So far, we have been considering only the unification-generalizing direction starting from LIFE- I which depends on the organization $< (P, [[\ ]]), B, F, \sim >$, 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 will belong 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 attention 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 $B_1, \cdots, B_n$ at the same time for one object-level program P and consider independent maps [[ ]]:

$Bp \to B_i$ $(1 \leqq i \leqq n)$. Any concrete logic programming language which 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 abstract data types or 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.

§0-5. Concluding Remarks

In this chapter, 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 ($I \leqq X < \infty$). Each LIFE-X is a logic-oriented inferential framework having an abstract organization $\boldsymbol{O}$ with an inferential method D based on $\boldsymbol{O}$. As discussed in this chapter, for some X, $\boldsymbol{O}$ 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, $[[\ \ ]] : Bp \to B$ 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 taste.

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.

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.

Chapter I.
Definition of LIFE-III and Its Theoretical Background

§1-0. Introduction

Nowadays, there are many logic programming languages which are considered to be extensions or generalized versions of Prolog. In this situation, the appearance of [13] etc powerfully suggests the trend that logic programming theorists gradually begin to build a language scheme which governs many concrete logic-oriented (Prolog-type) programming languages instead of constructing an individual programming language. The project of our proposing paradigm LIFE-Ω (Logic-oriented Inferential Framework Extensions-infinite series) is planned to be one rank higher, that is, to synthesize these tones of schemes with the harmony of Boolean-valued semantics and thus to compose an abstract (or rather transcendental) level of logic-oriented programming symphony. This means we not only reorganize logic programming language schemes so far exist but also create new categories from a viewpoint of Boolean-valued universe. In other words, by abstractly determining the characters of Boolean algebra B, map $[\![\ \ ]\!]:Bp\to B$, filter F over B, relation $\sim$ over the set of all atoms and a "kind" of unification in addition to the type of a (object level) program P in LIFE-Ω, we can obtain a language scheme. (The novelty and the philosophical background of LIFE-Ω is discussed in the previous chapter.) Thus obtained scheme is and will be named LIFE-X (Logic-oriented Inferential Framework Extension-class X) where variable X ranges from 0 to ∞. As the starting point, let's define LIFE-0=pure Prolog as a language scheme. The main purpose of this paper is to define LIFE-III as a direct generalization of LIFE-0 and to give the theoretical background. (For definitions of LIFE- I and LIFE-II, see [33].) What is the advantage of our investigating this sort of highly abstract schemes? There are two merits we dare point out. The first concerns with the theoretical aspect and the second with the practical facet.

Take a logic programming language L. We should check both the soundness and the completeness of the procedural semantics of L in its suitable sense, if L is regarded as a logic programming language at all. However, for an AI language programmer in general, verifying the soundness and the completeness for each L is rather tedious. Instead, if we prove the soundness and the completeness of LIFE-X as a scheme, then we can assure the theoretical background of each individual programming language belonging to this category. Of course, in order to assure the theoretical aspects of L, we ought to check whether $L \in$ LIFE-X or not for a cetain $1 \leqq X < \infty$. Pragmatically, the check amounts to (try to) give a Boolean algebraic semantics of L by determining B, [[ ]], F, $\sim$, unifier kind, the type of P in the framework of LIFE-Ω. So, the essence of the first merit becomes the following. If one can directly and rather easily prove the soundness and the completeness of L, then there is no problem. However,there sometimes happens the case that it is difficult to prove the theoretical background of an extended logic programming language. In this case, Boolean-valued technique reveals its real power. Thus, the first merit is of a bottom-up nature. Its necessity arises out of each already existing (or planned) programming language and the stuff (theoretical background) is lifted to the dining room LIFE-X, where it is cooked following a recipe of Boolean-valued semantics.

On the other hand, the second merit is of a top-down nature. By properly choosing P, B, [[ ]], F, $\sim$, unification kind etc, we can define a logic programming language scheme LIFE-X as an example of LIFE-Ω, so that no logic programming extension of pure Prolog so far exists belongs to this category. The point is, first we choose a property Ξ whose color can't be obtained by the simple mixture of the conventional logic programming paint but is expected to be represented by a logic-oriented programming language. Then, decide the above Boolean-character in LIFE-Ω in order that the resulting compound LIFE-X can express or at least approximate the color of Ξ. As a

consequence, every concrete programming language L as a projection of LIFE-X equips this desired property Ξ and we need not worry about the logical background of L.

We begin section 1-1 by defining the declarative semantics of Boolean-valued model. In section 1-2, we define the procedural semantics of Boolean-valued unification. The main target of section 1-3 is to prove the soundness of Boolean-valued derivation of LIFE-III. Section 1-4 deals with the fixed point semantics and finally in section 1-5, we prove the completeness of Boolean-valued derivation of LIFE-III in a certain form. Since there are many definitions which are necessary to state new concepts, we have devided the study of the theoretical background of LIFE-III into a few parts. The successive chapter II handles Boolean-valued logical completeness in relativized situations. Expected instanciations of LIFE-III as an AI language are discussed in chapter IV and chapter V.

Except the new notions defined, we have employed conventional terminology used in [21]. For example, a program P is a set of Horn clauses and

(1) Σ_P and Π_P mean the set of all function symbols (including constants) and the set of predicate symbols used in P respectively and U_P, B_P mean the set of all ground terms (Herbrand universe) and ground atoms (Herbrand base) generated from Σ_P and Π_P respectively. More generally, let $T(\Sigma_p, V)$ ($A(\Pi_p, V)$) be the set of all terms (atoms) generated from $\Sigma_p(\Pi_p)$ whose variables are picked up from V. Then, obviously, $U_P \subset T(\Sigma_p, V)$ and $B_P \subset A(\Pi_p, V)$.

(2) Let C be a program clause of the form

$$A \leftarrow B_1, \cdots\cdots, B_q.$$

Then, $C^+ = A$ and $C^- = \{B_1, \cdots\cdots, B_q\}$.

So, for any substitution θ, $C^+\theta$ means $A\theta$ and $C^-\theta$ means $B_1\theta, \cdots\cdots, B_q\theta$.

In addition to the above, let's employ the following abbreviation throughout this paper for the sake of convenience.

Definition 1-0-1.

For any atom A in P, $A^{\#} \in \Pi_P$ is such that A is an instance of $A^{\#}$. ⊣

§1-1. Declarative semantics of Boolean-valued model

Definition 1-1-1. Let B be a complete Boolean algebra.

(1) (U, f) is a B-valued interpretation with the universe U iff

i) term assignment is the same as (usual) two-valued case

ii) for any (ground) atom $P(t_1, \cdots, t_n)$, $f(P(t_1, \cdots, t_n)) \in B$.

iii) for an arbitrary formula ψ, $f(\psi) \in B$ is defined by the usual induction on logical symbols,

eq. $f(\forall x p(x)) = \bigwedge_{a \in U} f(p(a))$, $f(\neg \psi) = \neg f(\psi)$, etc.

(2) Let S be a set of closed formulas in L and (U,f) be a B-valued interpretation. Then, (U, f) is a model of S with respect to a complete filter F over B iff $(\forall \sigma \in S)(f(\sigma) \in F)$.

In the following, we call this sort of model "F-model". Especially, when we take $F = \{1\}$, F-model is called "B-valued model". ⊣

Note: A filter F over B is "complete" iff $(\forall X \subseteq F)(\wedge X \in F)$. If B is finite, then, of course, every filter becomes complete. Moreover, for any B, every principal filter obviously becomes a complete filter and vice versa. In the following, we exclusively use symbols F and B to denote "complete filter" and "complete Boolean algebra" respectively. ⊣

Definition 1-1-2. Let S be a set of closed formulas.

(1) S is F-satisfiable iff there is a F-model of S.

(2) S is F-unsatisfiable iff S is not F-satisfiable.

(3) Let σ be a closed formula. Then,

σ is a F-logical consequence of S iff

$(\forall M)$(M is a F-model of S$\rightarrow$M is a F-model of σ). $\dashv$

Note: This is the starting point where generalized B-valued notion becomes essentially different from (usual) 2-valued notion.

For example, let M be a F-model of S. Then, there may be a closed formula $\sigma \notin S$ such that neither (M is a F-model of σ) nor (M is a F-model of $\neg\sigma$) holds. This is simply because $B-(F \cup I)$ may not be empty in a general situation, where I is an ideal dual to F. $\dashv$

However, the next relation, which is a Boolean-valued version of a well-known result in usual 2-valued case, holds.

Proposition 1-1-3. Let $S \cup \{\sigma\}$ be a set of closed formulas. Then, σ is a F-logical consequence of S iff $S \cup \{\neg\sigma\}$ is F-unsatisfiable.

Proof: ($\Rightarrow$) Let (U, f) be a F-model of S. Then, (U, f) is a F-model of σ. This means $f(\neg\sigma) = \neg f(\sigma) \notin F$. So, from the arbitrariness of (U, f), there is no F-model of $S \cup \{\neg\sigma\}$.

($\Leftarrow$): Suppose σ is not a F-logical consequence of S.

This means $(\exists(U, f))$ ((U, f) is a F-model of S and $f(\sigma) \notin F$).

Let (U, f) be as above.

i) The case of $f(\sigma) \in I$, where I is the dual idea of F.

Then, $f(\neg\sigma) = \neg f(\sigma) \in F$. So, (U, f) becomes a witness of $SU\{\neg\sigma\}$'s F-satisfiability.

ii) The case of $f(\sigma) \notin I$, i.e., $f(\neg\sigma) \notin F$.

In this case, since $f(\sigma) \notin F \wedge f(\neg\sigma) \notin F$, neither σ nor $\neg\sigma$ is a F-logical consequence of S.

Now consider a complete ultrafilter ultrafilter F' including $F \cup \{f(\neg\sigma)\}$.

Let $2' = B/F'$ be the quotient algebra of B and

h: $B \to 2'$ be the corresponding complete homomorphism.

Then, $(U, h \circ f)$ becomes a 2'-model of $S \cup \{\neg\sigma\}$ because,

$(\forall\tau \in S)\,(h \circ f(\tau) = 1') \wedge h \circ f(\neg\sigma) = 1'$.

Here, without loss of genrality, we can consider 2' is a subalgebra of B, since $2' \simeq 2 \subset B$. Let g: $2' \to B$ be the natural embedding. Then, $(U, g \circ h \circ f)$ becomes a B-model of $S \cup \{\neg\sigma\}$ such that $(\forall\tau \in S)\ (g \circ h \circ f(\tau) = 1) \wedge g \circ h \circ f(\neg\sigma) = 1$. So, of course, $(U, g \circ h \circ f)$ is a F-model of $S \cup \{\neg\sigma\}$.

$\therefore$ $S \cup \{\neg\sigma\}$ is F-satisfiable. □

Definition 1-1-4. We can define both Herbrand universe and Herbrand base just the same as usual. In this situation,

(1) (U, f) is a B-valued Herbrand interpretation iff (U, f) is a B-valued interpretation and U is a Herbrand universe and the term assignment is the same as usual Herbrand case.

(2) $M = (U, f)$ is a Herbrand F-model of S iff M is a B-valued Herbrand interpretation such that M is a F-model of S. ⊣

Note: For B-valued Herbrand interpretation (U, f), the term assignment is fixed (and each logical operation is interpreted as a Boolean operation). So, (U, f) is completely determined by $f \restriction B_L : B_L \to B$, where B_L is the Herbrand base. An interesting fact is that we can weaken the condition "completeness" of F to "$\aleph_1$-completeness" in case of Herbrand F-model, because $|L| \leqq \aleph_0$ and so $|U_L| \leqq \aleph_0$, $|B_L| \leqq \aleph_0$. In this case, there may be a $\aleph_1$-complete filter which is not principal if $|B| \geqq \aleph_1$. ⊣

Proposition 1-1-5. Let S be a set of clauses. If S has a *F*-model, then S has a Herbrand *F*-model.

Proof: Let (U, f) be a *F*-model of S. Let M=(U', g) be a B-valued Herbrand interpretation such that $(\forall\sigma\in B_L)(g(\sigma)=f(\sigma))$, where B_L is the Herbrand base. Since (U, f) is a *F*-model of S, M is also a *F*-model of S.

(For example, $f(\forall x p(x)) \leq g(\forall x p(x))$.) □

Corollary 1-1-6. Let S be a set of clauses. Then, S is *F*-unsatisfiable iff S has no Herbrand *F*-model.

Proof: (⇒): Obvious.

(⇐): If S is *F*-satisfiable, then S has a Herbrand *F*-model by the above Proposition 1-1-5. □

Proposition 1-1-7. (*F*-model intersection property).

Let P be a program and $\{(U_P, f_i) \mid i\in I\}$ be a non-empty set of Herbrand *F*-models of P. Then, M=(Up,f) is also a Herbrand *F*-model of P, where f is defined by $(\forall\sigma\in B_P)(f(\sigma)=\wedge f_i(\sigma))$.

Proof: Let C∈P. Then, $f(\forall C)=\wedge fi(\forall C)\in F$, because *F* is complete.

∴ (Up, f) is a Herbrand *F*-model of P. □

Definition 1-1-8. Let P be a program.

(1) (U_P, μ) is the least Herbrand *F*-model of P iff $(\forall\sigma\in B_P)(\mu(\sigma)=\wedge f_i(\sigma))$, where $\{(U_P, f_i) \mid i\in I\}$ is the set of all Herbrand *F*-models of P.

(2) For any $f: Bp\rightarrow B$, $B_f[F]=\{\sigma\in Bp \mid f(\sigma)\in F\}$. ⊣

Theorem 1-1-9. Let P be a program. Then,

$B\mu[F] = \{\sigma \in Bp \mid \sigma$ is a F-logical consequence of P$\}$, where (U_P, μ) is the least Herbrand F-model.

Proof: Let $\sigma \in Bp$. Then, σ is a F-logical consequence of P

$\Rightarrow$ $(\forall(Up, f_i))$ ((Up, f_i) is a Herbrand F-model of P$\rightarrow f_i(\sigma) \in F$) $\cdots$ (1)

$\Leftrightarrow$ $\bigwedge_{i \in I} f_i(\sigma) \in F$, where $\{(Up, f_i) \mid i \in I\}$ is the set of all Herbrand F-models

$\Leftrightarrow$ $\mu(\sigma) \in F$, where (Up, μ) is the least Herbrand F-model

$\Leftrightarrow$ $\sigma \in B\mu[F]$.

To see the converse of (1), suppose $(\forall(Up, f_i))$ ((Up, f_i) is a Herbrand F-model of P$\rightarrow f_i(\sigma) \in F$). To show that σ is a F-logical consequence of P, assuming σ is not a F-logical consequence of P, we will get a contradiction. Now, σ is not a F-logical consequence of P

$\Rightarrow$ $(\exists(U, g))$ ((U, g) is a F-model of P but (U, g) is not a F-model of σ)

Here, define a Herbrand interpretation (Up, h) by $(\forall \sigma \in Bp)$ $(h(\sigma) = g(\sigma))$.

Then, by Proposition 1-1-5, (Up, h) is a F-model of P. However, by definition of h, $h(\sigma) = g(\sigma) \notin F$. This contradicts the assumption. □

Definition 1-1-10. Let P be a program, G be a goal $\leftarrow A_1, \cdots, A_k$ and θ be a substitution for G. Then, θ is a F-correct answer substitution for $P \cup \{G\}$ iff $\forall((A_1 \wedge \cdots \wedge A_k)\theta)$ is a F-logical consequence of P. ⊣

Theorem 1-1-11. Let P be a program, G be a goal $\leftarrow A_1, \cdots, A_k$ and θ be a ground substitution for G. Then, the following are equivalent.

(a) θ is a F-correct answer substitution for $P \cup \{G\}$.

(b) $(f(A_1\theta) \wedge \cdots \wedge f(A_k\theta)) \in F$ for any Herbrand F-model (U_P, f) of P.

(c) $(\mu(A_1\theta) \wedge \cdots \wedge \mu(A_k\theta)) \in F$ w.r.t. the least Herbrand F-model (U_P, μ) of P.

Proof:

(a)→(b). Suppose θ is F-correct. Let (Up, f) be a Herbrand F-model of P. Since θ is ground, (Up, f) is a F-model of $(A_1 \wedge \cdots \wedge A_k)\theta$.

$\therefore$ $(f(A_1\theta) \wedge \cdots \wedge f(A_k\theta)) \in F$.

(b)→(c). Obvious.

(c)→(a). Let (Up, μ) be the least Herbrand F-model of P. Suppose $\mu(A_1\theta) \wedge \cdots \wedge \mu(A_k\theta) \in F$. Then, by Theorem 1-1-9,

$A_i\theta$ is a F-logical consequence of P for $1 \leqq i < k$

$\Rightarrow$ $A_1\theta \wedge \cdots \wedge A_k\theta$ is a F-logical consequence of P

$\Rightarrow$ $(A_1 \wedge \cdots \wedge A_k)\theta$ is a F-logical consequence of P

$\Rightarrow$ θ is a F-correct answer substitution for $P \cup \{G\}$. □

§1-2. Procedural semantics of Boolean-valued unification

In this section, we define the procedural semantics of Boolean-valued unification.

Definition 1-2-1 Let $[[\ ]]: Bp \to B$ and $A, B \in A(\Pi p, V)$.

Let $\sim$ be a reflexive, symmetric relation over $A(\Pi p, V)$.

(1) $\sim$ is "substitution transitive" iff

for any atom A, B and any substitution ρ,

$A \sim B \Rightarrow A\rho \sim B\rho$.

(2) A substitution θ is called "a B-valued unifier for A and B with respect to [[ ]] and a substitution transitive relation $\sim$"

iff $A\theta \sim B\theta$ and

$$\bigwedge_{p:\text{ground}} ([[\ A\theta p\ ]] \leftrightarrow [[\ B\theta p\ ]]) = 1 \quad \cdots \quad (1)$$

where $1 \in B$ is the greatest element.

Here, we should notice that the condition (1) is equal to

$[\![A\theta\rho]\!] = [\![B\theta\rho]\!]$ for any ground substitution ρ for $A\theta \leftrightarrow B\theta$. ⊣

Throughout this chapter, we assume that the substitution transitive relation ~ needed for B-valued unification is arbitrary but fixed. Of course, when we decide a concrete language, we should define ~ according to the aim.

Note: If we use B-valued unification, we can unify, say, ground atoms A(a) and B(b) such that

$$A^{\#} \neq B^{\#} \quad \text{but} \quad [\![A(a)]\!] = [\![B(b)]\!] \text{ and } A(a) \sim B(b).$$

To tell the truth, B-valued unification is a formal definition of the unification scheme including the notion of both universal unification and semantic unification in its general sense. ⊣

Once we obtain the notion of B-valued unifier, we can define the notion of "B-most (or maximally) general unifier (B-mgu in short)". First of all, given A and B, the notion of B-mgu for A and B may not be uniquely determined even modulo renaming substitution.

Example 1-2-2 Let $A(X_1, X_2)$ and $B(Y_1, Y_2)$ be such that

$\neg(\forall s,t$: ground terms) ($[\![A(a,s)]\!] = [\![B(a,t)]\!]$)

$\wedge$ $\neg(\forall s, t$: ground terms) ($[\![A(s,b)]\!] = [\![B(t,b)]\!]$)

$\wedge$ $\neg(\forall s,t$: ground terms) ($[\![A(a,t)]\!] = [\![B(s,t)]\!]$)

$\wedge$ $\neg(\forall s,t$: ground terms) ($[\![A(s,t)]\!] = [\![B(a,t)]\!]$)

$\wedge$ $\neg(\forall s,t$: ground terms) ($[\![A(s,t)]\!] = [\![B(s,b)]\!]$)

$\wedge$ $\neg(\forall s,t$: ground terms) ($[\![A(s,b)]\!] = [\![B(s,t)]\!]$)

$\wedge$ $\neg(\forall s,t$: ground terms) ($[\![A(s,t)]\!] = [\![B(s,t)]\!]$)

$\wedge$ $(\forall t$: ground term) ($[\![A(a,t)]\!] = [\![B(a,t)]\!]$) $\wedge$ $(A(a,Y) \sim B(a,Y))$

$\wedge$ $(\forall t$: ground term) ($[\![A(t,b)]\!] = [\![B(t,b)]\!]$) $\wedge$ $(A(X,b) \sim B(X,b))$

where $a,b \in U_P$.

Then, both $\theta=\{X_1/a,\ Y_1/a,\ Y_2/X_2\}$ and $\theta'=\{X_2/b,\ Y_2/b,\ Y_1/X_1\}$ become possible candidates of B-mgu for $A(X_1, X_2)$ and $B(Y_1,\ Y_2)$. However, θ is not a renaming substitution of θ'. ⊣

The crucial property concerning (usual) mgu is that "For any syntactical unifier θ for A and B, there is a mgu θ' for A and B , and a substitution γ such that $\theta=\theta'\gamma$," because this is used to prove mgu lemma (and so lifting lemma) which is used to show the completeness property of syntactical unification. With this fact in mind, we ought to expect that B-mgus also have the same propety. To see this, we need a little preparation.

Let K_P be the set of all substitutions based on U_P. Let's define relations $\leqq_S$, $\equiv_S$, $<_S$ on K_P in the following manner.

For any $\theta, \theta' \in K_P$,

$\theta' \leqq_S \theta$ iff $(\exists \gamma \in K_P)(\theta=\theta'\gamma)$

$\theta' \equiv_S \theta$ iff $\theta' \leqq_S \theta \ \wedge\ \theta \leqq_S \theta'$

$\theta' <_S \theta$ iff $\theta' \leqq_S \theta \ \wedge\ \theta' \not\equiv_S \theta$.

Consider $\boldsymbol{K}_P=(K_P/\equiv_S)$ and define a partial order $\lesssim_S$ on $\boldsymbol{K}_P$ so that, for any $[\theta]$, $[\theta'] \in \boldsymbol{K}_P$,

$$[\theta] \lesssim_S [\theta'] \text{ iff } \theta \leqq_S \theta'$$

for suitable representatives θ, θ' for $[\theta]$, $[\theta']$ respectively. (Obviously, the choice of a representative θ, θ' for $[\theta]$, $[\theta']$ does not affect the definition of $\lesssim_S$. So, this is well-defined.)

Using this notation,

Definition 1-2-3. A B-valued unifier θ for A and B is a B-mgu iff

$\neg(\exists\theta')(\theta'$ is a B-valued unifier for A and B $\wedge\ [\theta'] \lessdot_s [\theta])$, where $[\theta'] \lessdot_s [\theta]$ means, as usual, $[\theta'] \neq [\theta] \wedge [\theta'] \leqq_s [\theta]$. ⊣

By this definition, we get;

Lemma 1-2-4. For any B-valued unifier θ for A and B, there is a B-mgu θ' for A and B, and a substitution γ such that $\theta = \theta'\gamma$.

Proof: Let θ be a B-valued unifier for A and B. Consider

$I(\theta) = \{[\theta'] \in Kp \mid [\theta'] \leqq s\ [\theta]\}$.

A crucial property concerning $I(\theta)$ is that $I(\theta)$ has the minimum element $[\varepsilon]$ w.r.t. $\leqq s$, where ε is the empty substitution. From this fact, we at once notice that $I(\theta)$ is a finite set. This is because each term $t_i (1 \leqq i \leqq n)$ is built by a finite step through a finite (possibly different) paths modolo $\equiv s$ by definition of terms, where $\theta = \{X_1/t_1, \cdots, X_n/t_n\}$. (For example, without loss of generality, we can assume dome $(\theta') \subseteq$ dome (θ) for a suitable representative θ' of $[\theta']$ for each $[\theta'] \in I(\theta)$.) So, for any subset $Y \subset I(\theta)$ s.t. $Y \neq \varnothing$, we can always find a minimal element of Y w.r.t. $\leqq s$.

By especially taking

$$Y = \{[\theta'] \in I(\theta) \mid \theta' \text{ is a B-valued unifier for A and B}\} \neq \varnothing,$$

we get the required result. □

Note: Compared with the universal unification based on an equational theory E, the result of lemma 1-2-4 seems to be remarkable from a viewpoint that B-valued unification is more general than universal unification, because it declares that B-mgu always exists in the above sense. The reason is that the definition of $\leqq_s$ is independent of [[ ]] and $\sim$, and so essentially different from the similar relation $\leqq_E$ based on E. ⊣

Definition 1-2-5. A program P is B-valued

iff $\exists [[\ ]] : B_P \to \mathbf{B}$. ⊣

In the following, when we write "(P, [[ ]])", (P, [[ ]]) always means a B-valued program P with respect to $[[\ ]] \in \mathbf{B}^{B_P}$.

Definition 1-2-6. Let (P, [[ ]]) be a B-valued program.

(1) Let G be a goal $\leftarrow A_1, \cdots, A_k$, C be an input clause where C is a variant of a clause in P. Then, we say G' is B-derived from G and C using B-valued unifier θ with the valuc $b \in \mathbf{B}$, if the following conditions hold.

(i) $A_m \in \{ A_1, \cdots, A_k \}$ is the selected atom.

(ii) θ is a B-mgu for A_m and C^+.

(iii) G' is the goal

$$\leftarrow (A_1, \cdots, A_{m-1}, C^-, A_{m+1}, \cdots, A_k)\theta.$$

iv) $b = \bigwedge_{p: \text{ground}} [[A_m \theta p]]$.

(2) Let G be a goal. Then, a B-derivation of $P \cup \{G\}$ consists of

a sequence of goals

$$G = G_0, G_1, \cdots,$$

a sequence of input clauses

$$C_1, C_2, \cdots,$$

a sequence of B-valued unifiers

$$\theta_1, \theta_2, \cdots,$$

and a sequence of B-values

$$b_1, b_2, \cdots,$$

such that each G_{i+1} is B-derived from G_i and C_{i+1} using θ_{i+1} as a B-mgu with the value b_{i+1}. ⊣

Definition 1-2-7. Let (P, [[ ]]) be a B-valued program and G be a goal $\leftarrow A_1, \cdots, A_k$.

(1) A B-refutation R of P∪{G} is a finite B-derivation of P∪{G} which has the empty clause $\square$ as the last goal G_n and the resulting B-value

$$v(R) = b_1 \wedge \cdots \wedge b_n .$$

In this case, we say R has length n and $v(R)$ is called "a refutation value (of G w.r.t. (P, [[ ]]))."

(2) A B-answer substitution θ for P∪{G} is the substitution $\theta_1 \cdots \theta_n$ (obtained by restricting to the variables in G), where $<\theta_1, \cdots, \theta_n>$ is the sequence of B-valued unifiers appeared in a B-refutation R of P∪{G}.

In this case, the B-value

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

is called "an answer value (of G w.r.t. (P, [[ ]]))." ⊣

As for a relation between $v(R)$ and $v(\theta)$, we get

Lemma 1-2-8 Let θ be a B-answer substitution of a B refutation R of P∪{G}. Then,

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

Proof: Let θ be a B-answer substitution of a B-refutation R of $P \cup \{\leftarrow A_1, \cdots, A_k\}$, where $<\theta_1, \cdots, \theta_n>$ and $<C_1, \cdots, C_n>$ are B-valued unifiers and input clauses used in R.

Let $\{\theta_{j_1}, \cdots, \theta_{j_k}\} \subset \{\theta_1, \cdots, \theta_n\}$ and

$\{C_{j_1}, \cdots, C_{j_k}\} \subset \{C_1, \cdots, C_n\}$ be such that

(θ_{j_i}, C_{j_i}) is used to eliminate A_i for $1 \leqq i \leqq k$, that is,

θ_{j_i} is a B-valued unifier for $C_{j_i}^{+}$ and $\Lambda_i\theta_1 \cdots \theta_{j_i-1}$ for $1 \leqq i \leqq k$.

Then, for $1 \leqq i \leqq k$,

$$\begin{aligned} \bigwedge_{\rho:\text{ground}} [[A_i\theta\rho]] &= \bigwedge_{\rho:\text{ground}} [[A_i\theta_1 \cdots \theta_n\rho]] \\ &\qquad (\leftarrow\theta_1 \cdots \theta_{j_i} \text{ is an initial segment of } \theta.) \\ &\geq \bigwedge_{\rho:\text{ground}} [[A_i\theta_1 \cdots \theta_{j_i}\rho]] \\ &= b_{j_i} \end{aligned}$$

$$\therefore\ v(\theta) = \bigwedge_{\rho:\text{ground}} ([[A_1\theta\rho]] \wedge \cdots \wedge [[A_k\theta\rho]]) \quad \geqq \quad b_{j_1} \wedge \cdots \wedge b_{j_k}$$
$$\geqq \quad b_1 \wedge \cdots \wedge b_n = v(R).$$

□

Definition 1-2-9

(1) An unrestricted B-refutation R_u is a B-refutation except that we weaken the restriction of B-mgu to a B-valued unifier at each B-derivation step.

(2) A special (ground) B-refutation R_s is an unrestricted B-refutation such that we specify B-valued unifier to a ground B-valued unifier at each B-derivation step. ⊣

As stated before, mgu lemma is essential to prove the completeness of syntactical unification. This lemma also holds for B-valued case.

Lemma 1-2-10. (B-mgu lemma).

Let (P, [[ ]]) be a B-valued program and G be a goal. If P∪{G} has an unrestricted B-refutation with B-valued unifiers $<\theta_1, \cdots, \theta_n>$, then P∪{G} has a B-refutation of the same length with B-mgus $<\theta_1', \cdots, \theta_n'>$ such that

$(\exists\gamma : \text{substitution})(\theta_1 \cdots \theta_n = \theta_1' \cdots \theta_n'\gamma)$

Proof: Using B-mgu and B-valued unifier instead of usual mgu and unifier, the proof goes through just the same as the 2-valued case. (In this proof, we need the result of Lemma 1-2-4.) □

Contrastingly, the following specializing direction holds, too.

Lemma 1-2-11. (Specialization lemma for B-refutation).

Let (P, [[ ]]) be a B-valued program and G be a goal. If P∪{G} has a B-refutation with B-mgus $< \theta_1, \cdots, \theta_n >$, then P∪{G} has a special B-refutation of the same

length with the same input clauses and ground B-valued unifiers $< \theta_1', \cdots, \theta_n' >$ such that

$$(\exists\gamma : \text{substitution})(\theta_1 \cdots \theta_n\gamma = \theta_1' \cdots \theta_n').$$

Proof: Let $(P, [[\ ]])$ be a B-valued program and G be a goal. Suppose $P \cup \{G\}$ has a B-refutation R with B-mgus $<\theta_1, \cdots, \theta_n>$ and input clauses $<C_1, \cdots, C_n>$ and the resulting goals $<G_1, \cdots, G_n = \square>$. We can assume that C_i has no variable common with $dom(\theta_1 \cdots \theta_{i-1})$. As a result, we can assert that

$$C_i^- \theta_i = C_i^- \theta_1 \cdots \theta_{i-1}\theta_i \qquad \cdots (1)$$

for $1 \leqq i \leqq n$.

Let W be the set of all variables appeared in G, $C_1, \cdots, C_n$ and let γ be a ground subsitution such that $W \subset dom(\gamma)$ and $\theta_1 \cdots \theta_n\gamma$ becomes ground.

Now, let $A_m \in G$ be the selected atom for θ_1. Since θ_1 is a B-valued unifier for C_1^- and A_m, of course, $\theta_1 \cdots \theta_n\gamma$ becomes a ground B-valued unifier for C_1^+ and A_m. Let

$$G_1' = \leftarrow (A_1, \cdots, A_{m-1}, C_1^-, A_{m+1}, \cdots, A_k)\,\theta_1 \cdots \theta_n\gamma = G_1\theta_2 \cdots \theta_n\gamma.$$

Let $A_n\theta_1 \in G_1$ be the selected atom for θ_2.. Since θ_2 B-valued unifies $A_n\theta_1$ and C_2^+, $\theta_2 \cdots \theta_n\gamma$ B-valued unifies $A_n\theta_1$ and C_2^+. So, $\theta_2 \cdots \theta_n\gamma$ B-valued unifies $A_n\theta_1\theta_2 \cdots \theta_n\gamma$ and C_2^+. Here, by (1), $C_2^-\ \theta_2 \cdots \theta_n\gamma = C_2^-\theta_1\theta_2 \cdots \theta_n\gamma$. So, the resulting goal G_2' from G_1' with the ground B-valued unifier $\theta_2 \cdots \theta_n\gamma$ for the selected atom $A_n\theta_1\theta_2 \cdots \theta_n\gamma$ and C_2^+ becomes

$$\begin{aligned} G_2' &= \leftarrow (A_1\theta_1 \cdots \theta_n\gamma, \cdots, A_{n-1}\theta_1 \cdots \theta_n\gamma, C_2^-\theta_1 \cdots \theta_n\gamma, A_{n+1}\theta_1 \cdots \theta_n\gamma, \cdots, \\ &\qquad A_k\theta_1 \cdots \theta_n\gamma) \\ &= \leftarrow (A_1, \cdots, A_{n-1}, C_2^-, A_{n+1}, \cdots, A_k)\theta_1 \cdots \theta_n\gamma = G_2\theta_3 \cdots \theta_n\gamma. \end{aligned}$$

Repeating this process, we get the sequence of goals

$$G_1', G_2', \cdots, G_n'$$

such that

$$G_i' = G_i\theta_{i+1} \cdots \theta_n\gamma \quad \text{for } 1 \leqq i \leqq n-1$$

and

$G_n' = G_n\gamma = \square$.

Here, for each $1 \leqq i \leqq n$, the input clause is C_i and the B-valued unifier is $\theta_i \cdots \theta_n\gamma$, which should be ground by definition. Finally, with this special B-refutation R_s for $P \cup \{G\}$, we notice that the B-answer substitution become $(\theta_1 \cdots \theta_n\gamma)$ $(\theta_2 \cdots \theta_n\gamma) \cdots (\theta_n\gamma) = \theta_1 \cdots \theta_n\gamma$. (In this proof, we essentially use the substitution transitivity of $\sim$ for the first time.) $\square$

In the following, we shall use this kind of derivation. This means, we construct a kind of B-valued logic programming language scheme. Let P be a program. A logic programming language scheme is called "LIFE-III" iff it is the class of Boolean-valued logic programming languages $<(P, [\![\ \ ]\!]), B, F, \sim>$ based on B -derivation.

§1-3. Soundness of LIFE-III

So far, we have been defining general notations used in a B-refutation. The next notions are those that are used mainly to maintain soundness and completeness of LIFE- III.

Definition 1-3-1

(1) $(P, [\![\ \ ]\!])$ is a F-program iff $(U_p, [\![\ \ ]\!])$ becomes a F-model of P.

(2) A (unrestricted) B-refutation R is a (unrestricted) F-B-refutation iff $v(R) \in F$.

(3) A B-answer substitution θ is a F-B-answer substitution iff $v(\theta) \in F$.

(4) A substitution θ for G is an Herbrand F-correct answer substitution for P∪{G} iff for any Herbrand F-model (U_P, f) of P, (U_P, f) is a F-model of ∀(Gθ). ⊣

Note: The above notion of Herbrand F-correct answer substitution is a little weaker than that of F-correct answer substitution defined in Definition 1-10, because we restrict our attension only to Herbrand F-models of P instead of (general) F-models of P. ⊣

Now, as a direct consequence of Lemma 1-2-8, we get;

Proposition 1-3-2. Let (P, [[ ]]) be a B-valued program and G be a goal. If P∪{G} has a F-B-refutation R with the B-answer substitution θ, then θ is a F-B-answer substitution.

Proof: By Lemma 1-2-8, we already know $v(R) \leq v(\theta)$. Since $v(R) \in F$, so $v(\theta) \in F$. □

Next, the relation between F-program and F-B-answer substitution gives the soundness theorem for LIFE-III.

Theorem 1-3-3. (Soundness theorem of LIFE-III.)
Let (P, [[ ]]) be a F-program and G be a goal. Then, every B-answer substitution θ for P∪{G} is a F-B-answer substitution.

Proof: Let (P, [[ ]]) be a F-program and G be a goal $\leftarrow A_1, \cdots, A_k$ and $<\theta_1, \cdots, \theta_n>$ be the sequence of B-valued unifiers used in a B-refutation of P∪{G}. We have to show

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

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

The result is proved by induction on the length n of the refutation.

i) The case of $n=1$.

This means that G has a form $\leftarrow A_1$ and $\exists(A\leftarrow)\in P$ such that θ_1 B-valued unifies A and A_1. Here, since $(A\leftarrow)\in P$ and $(P, [[\ ]])$ is a F-program,

$$(Up, [[\ ]])\models\forall(A\theta_1),$$

i.e.,

$$\bigwedge_{\rho:\text{ground}}([[A\theta_1\rho]])\in F. \quad \cdots (1)$$

$$\therefore\ v(\theta)=\bigwedge_{\rho:\text{ground}}([[A_1\theta_1\rho]]\in F.$$

「Remark:

Let ρ be such that $A_1\theta_1\rho$ is ground. Extend this ρ, if necessary, to ρ' such that ρ' is a ground substitution for $A_1\theta_1\leftrightarrow A\theta_1$. Then, by definition of B-valued unifier, $[[A_1\theta_1\rho']]=[[A\theta_1\rho']]$. Moreover, since $A\theta_1\rho'$ is ground, $[[A\theta_1\rho']]\in F$ by (1).

So, $[[A_1\theta_1\rho']]=[[A_1\theta_1\rho]]\in F$. 」

ii) General case

Suppose that the result holds for the case of $n-1$. Let $<\theta_1, \cdots, \theta_n>$ be the sequence of B-mgus used in a B-refutation of $P\cup\{G\}$ with length n. Let C_1 be the first input clause and A_m be the selected atom in G. Since

$$\leftarrow(A_1, \cdots, A_{m-1}, C_1^-, A_{m+1}, \cdots, A_k)\theta_1$$

has a B-refutation of length $n-1$ with B-mgus $<\theta_2, \cdots, \theta_n>$, by induction hypothesis,

$$\bigwedge_{\rho:\text{ground}}([[A_1\theta_1\theta_2\cdots\theta_n\rho]]\wedge\cdots\wedge[[A_{m-1}\theta_1\theta_2\cdots\theta_n\rho]]\wedge[[C_1^-\theta_1\theta_2\cdots\theta_n\rho]]\wedge([[A_{m+1}\theta_1\theta_2\cdots\theta_n\rho]]\wedge\cdots\wedge[[A_k\theta_1\theta_2\cdots\theta_n\rho]])\in F$$

So, $\bigwedge_{\rho:\text{ground}}([[C_1^-\theta\rho]]\in F \quad \cdots (2)$

and

$$\bigwedge_{\rho:\text{ground}}([[A_1\theta\rho]]\wedge\cdots\wedge[[A_{m-1}\theta\rho]]\wedge[[A_{m+1}\theta\rho]]\wedge\cdots\wedge[[A_k\theta\rho]])\in F. \quad \cdots (3)$$

Here, since $C_1\in P$ and $(P, [[\ ]])$ is a F-program,

$$(Up, [[\ ]])\models\forall(C_1\theta)$$

i.e.,

$$\bigwedge_{\rho:\text{ground}} [[C_1\theta\rho]] = \bigwedge_{\rho:\text{ground}} ([[C_1{}^+\theta\rho]] \leftarrow [[C_1{}^-\theta\rho]]) \in F. \quad \cdots (4)$$

(2)+(4) gives

$$\bigwedge_{\rho:\text{ground}} [[C_1{}^+\theta\rho]] \in F. \quad \cdots (5)$$

(Here, we tacitly use the similar reasoning as in the above Remark.)

Since θ_1 is a B-valued unifier of A_m and $C_1{}^+$,

$[[A_m\theta_1\rho]] = [[C_1{}^+\theta_1\rho]]$ for any ground ρ

$\Rightarrow$ $[[A_m\theta_1\theta_2\cdots\theta_n\rho]] = [[C_1{}^+\theta_1\theta_2\cdots\theta_n\rho]]$ for any ground ρ

$\Rightarrow$ $[[A_m\theta\rho]] = [[C_1{}^+\theta\rho]]$ for any ground ρ

$\Rightarrow$ $\bigwedge_{\rho:\text{ground}} ([[A_m\theta\rho]] \in F$ by (5). $\cdots (6)$

$\therefore$ (3)+(6) gives

$$\left(\bigwedge_{\rho:\text{ground}} ([[A_1\theta\rho]] \wedge \cdots \wedge [[A_{m-1}\theta\rho]] \wedge [[A_{m+1}\theta\rho]] \wedge \cdots \wedge [[A_k\theta\rho]])\right)$$

$$\wedge \quad \left(\bigwedge_{\rho:\text{ground}} ([[A_m\theta\rho]])\right) \in F.$$

So, $v(\theta) = \bigwedge_{\rho:\text{ground}} ([[A_1\theta\rho]] \wedge \cdots \wedge [[A_k\theta\rho]]) \in F.$ □

Note: From this fact, we notice that, if θ is a B-answer substitution for $P\cup\{G\}$ w.r.t. a F-program $(P, [[\ ]])$, then

$$v(\theta) = \bigwedge_{\rho:\text{ground}} [[G\theta\rho]]$$

never becomes **0** in B. ⊣

Definitions 1-3-4. Let $(P, [[\ ]])$ be a B-valued program. Then,

i) A subset Sp of Bp is the success set of P, that is,

$Sp = \{v \in Bp \mid P\cup\{\leftarrow\sigma\}$ has a refutation in the usual sense $\}$.

ii) A subset $Sp([[\ ]], \sim)$ of Bp is a B-success set of $(P, [[\ ]])$ iff

$Sp([[\ ]], \sim) = \{\sigma \in Bp \mid P\cup\{\leftarrow\sigma\}$ has a B-refutation w.r.t. $[[\ ]]$ and $\sim\}$.

By definition, $Sp \subseteq Sp([[\ ]], \sim)$ for any $[[\ ]]$ and $\sim$. ⊣

Using this notion, as a direct consequence of Theorem 1-3-3, we at once notice that

Proposition 1-3-5. (Soundness theorem of B-derivation – the ground form –)
Let $(U_P, [[\ ]])$ be a Herbrand F-model of P. Then,

$$S_P([[\ ]], \sim) \subseteq B_{[[\ ]]}[F]$$

Proof: By Theorem 1-3-3,

$$\bigwedge_{\rho:\text{ground}} [[\sigma\theta\rho]] \in F.$$

Since σ is ground, this means

$$[[\sigma]] \in F.$$ □

Corollary 1-3-6. (Soundness theorem – the least Herbrand F-model form –)Let (U_P, μ) be the least Herbrand F-model of P. Then,
$S_P(\mu, \sim) \subseteq B_\mu[F]$.

Proof: Direct consequence of Proposition 1-3-5. □

By the way, by taking $\sim$ such that $(\forall A, B \in A(\Pi_P, V))\ (A \sim B \text{ iff } A = B)$, B-valued unification becomes the syntactical unification as a special case except additional B-value evaluations. From this fact, we obtain

Corollary 1-3-7. (Soundness of usual derivation)
Let $(P, [[\ ]])$ be a F-program and G be a goal. Then, every answer substitution θ for $P \cup \{G\}$ w.r.t. $(P, [[\ ]])$ is a Herbrand F-correct answer substitution.

Proof: Let $(P, [[\ ]])$ be a F-program and G be a goal. Let θ be an answer subsitution for $P \cup \{G\}$. Then, for any F-grogram $(P, [[\ ]]')$, θ become an answer substitution for $P \cup \{G\}$, because the construction of θ does not depend on $[[\ ]]$,

[[ ]]' $\in B^{B_P}$. In other words, for any Herbrand F-model (Up, f) of P, θ becomes a B-answer substitution for P$\cup${G} w.r.t. f and the above special $\sim$. By theorem 1-3-3, this means

$\bigwedge_{\rho:\text{ground}} f(G\theta\rho) \in F$ for every Herbrand F-model (Up, f) of P, where $f(G\theta\rho)$ is an abbreviation for "$f(A_1\theta\rho) \wedge \cdots \wedge f(A_k\theta\rho)$" if $G = (\leftarrow A_1, \cdots, A_k)$.

$\Rightarrow$ $f(\forall G\theta) \in F$ for every Herbrand F-model (Up, f) of P.

$\Rightarrow$ θ is a Herbrand F-correct answer substitution. □

Here, we can ask the following stronger question.

"Let (P, [[ ]]) be a F-program and G be a goal. Then, every answer substitution θ for P$\cup${G} w.r.t. (P, [[ ]]) is a F-correct answer substitution." ···(1)

To prove this extension form of Corollary 1-3-7, we need a little wider notion of B-valued program.

Definition 1-3-8. Let (D, f) be a F-model of a program P. Then, (P, (D, f)) is called F-program in a wider sense. ⊣

Using this wider notion, we can define
B-valued unifier, B-refutation, B-answer substitution etc
in a wider sense. For example,

θ is a B-valued unifier for A and B in a wider sense
iff $f(\forall (A\theta \leftrightarrow B\theta)) = 1$, where $A\theta \sim B\theta$.
θ is a F-B-answer substitution in a wider sense
iff $v(\theta) = f(\forall(G\theta)) \in F$

etc.

With these wider notions, in order to see (1), it is enough to check,

Theorem 1-3-9. (Soundness theorem of LIFE-III in a wider sense)
Let (P, (D, f)) be a F-program in a wider sense and G be a goal. Then, every B-answer substitution θ for $P \cup \{G\}$ w.r.t. (P, (D,f)) is a F-B-answer substitution in a wider sense.

Proof: Using wider notions, the proof is similar to the proof of Theorem 1-3-3. □

Then, we can obtain the proof of (1) through the similar argument to the proof of Corollary 1-3-7, using wider notions.

Note: The above situation typically shows that, considering B-valued concepts, we can always transform a result to the corresponding result in a wider sense. For this reason, we omit the investigation of B-valued concepts in wider senses. To tell the truth, as you know, in almost every case, the notion of Herbrand model is sufficient to say something positively about logic programming. (See, for example, Theorem 1-1-9.) ⊣

Now, by Proposition 1-3-2 and Theorem 1-3-3, we already know the relations

F-B-refutation $\Rightarrow$ F-B-answer substitution

and

F-program $\Rightarrow$ F-B-answer substitution

Here, a natural question is "Is there any relation between the notions of F-program and F-B-refutation?" The following lemma, combined with specialization lemma, answers this question.

Lemma 1-3-10. Let (P, [[ ]]) be a F-program and G be a goal. If P∪{G} has a special B-refutation R_s, then R_s becomes a F-B-refutation.

Proof: Let (P, [[ ]]) be a F-program and $G=(\leftarrow A_1, \cdots, A_k)$ be a goal and $<\theta_1, \cdots, \theta_n>$, $<C_1, \cdots, C_n>$ be the sequences of ground B-valued unifiers and input clauses used in a special B-refutation R_s of P∪{G}. We have to show

$$v(R_s) = (\bigwedge_{\rho:\text{ground}} [[C_1{}^+\theta_1\rho]] \wedge \cdots \wedge (\bigwedge_{\rho:\text{ground}} [[C_n{}^+\theta_n\rho]])$$
$$= [[C_1{}^+\theta_1]] \wedge \cdots \wedge [[C_n{}^+\theta_n]] \in F.$$

As usual, the result is proved by induction on the length n of the B-refutation.

i) The case of $n=1$

In this case, since $v(R_s)=[[C_1{}^+\theta_1]]=v(\theta)$, the proof is the same as the case of $n=1$ in the proof of Theorem 1-3-3.

ii) General case

Suppose the result holds for the case of $n-1$. Let A_m be the selected atom of G. Since

$$G_1 = \leftarrow(A_1, \cdots, A_{m-1}, C_1{}^-, A_{m+1}, \cdots, A_k)\,\theta_1$$

has a special B-refutation R_s' of length $n-1$ with the input clauses $<C_2, \cdots, C_n>$, by induction hypothesis,

$$v(R'_s) = [[C_2{}^+\theta_2]] \wedge \cdots \wedge [[C_n{}^+\theta_n]] \in F. \quad \cdots (1)$$

By Lemma 1-2-8, $v(R_s') \leqq v(\theta_2 \cdots \theta_n)$. So (1) gives $v(\theta_2 \cdots \theta_n) \in F$. This means

$$\bigwedge_{\rho:\text{ground}} ([[A_1\theta_1\theta_2 \cdots \theta_n\rho]] \wedge \cdots \wedge [[A_{m-1}\theta_1\theta_2 \cdots \theta_n\rho]] \wedge [[C_1{}^-\theta_1\theta_2 \cdots \theta_n\rho]] \wedge$$
$$[[A_{m+1}\theta_1\theta_2 \cdots \theta_n\rho]] \wedge \cdots \wedge [[A_k\theta_1\theta_2 \cdots \theta_n\rho]]\ ) \in F.$$

$\therefore$ Putting $\theta=\theta_1 \cdots \theta_n$,

$$\bigwedge_{\rho:\text{ground}} ([[C_1{}^-\theta\rho]] \in F. \quad \cdots (2)$$

Now, since $C_1 \in P$ and (P, [[ ]]) is a F-program,

$$(Up, [[\ ]]) \models V(C_1\theta),$$

i.e.,

$$\bigwedge_{\rho:\text{ground}} ([[C_1{}^+\theta\rho]] \leftarrow [[C_1{}^-\theta\rho]]\ ) \in F. \quad \cdots (3)$$

(2) + (3) gives

$\bigwedge_{\rho:\text{ground}} [[C_1{}^+\theta\rho]] \in F.$

Here, since θ_1 is a ground B-valued unifier for A_m and $C_1{}'$, $C_1{}^+\theta\rho = C_1{}^+\theta_1 \cdots \theta_n\rho = C_1{}^+\theta_1$ for any ground substitution ρ

$\therefore \quad [[C_1{}^+\theta_1]] \in F. \quad \cdots (4)$

So, (1)+(4) gives

$$v(R_s) = ([[C_1{}^+\theta_1]] \wedge \cdots \wedge [[C_n{}'\theta_1]]) \in F. \qquad \square$$

Note: Roughly speaking, Lemma 1-3-10 asserts the following.

Let (P, [[ ]]) be a F-program. Then, for any clause $C \subset P$ and for any ground substitution θ for C,

$$(U_P, [[ ]]) \models C\theta \text{ , i.e., } [[C\theta]] \in F.$$

However, this fact does not necessarily assert that

$$(U_P, [[ ]]) \models C^+\theta \text{ , i.e., } [[C^+\theta]] \in F.$$

But, whenever θ occurs as a ground B-valued unifier for the input clause C during a special B-refutation of $P \cup \{G\}$ for a suitable goal G, if possible, then

$$[[C^+\theta]] \in F. \qquad \dashv$$

As a corollary of this lemma, we get the following strong soundness theorem for B-valued derivation, which embodies the fruitfulness of B-valued concepts.

Theorem 1-3-11. (Strong soundness theorem of LIFE-III)

Let (P, [[ ]]) be a F-program and G be a goal. Suppose $P \cup \{G\}$ has a B-refutation R with B-answer substitution θ, then $P \cup \{G\}$ has a special F-B-refutation R_s with B-answer substitution $\theta\gamma$ for a suitable γ.

Proof: Suppose $P \cup \{G\}$ has a B-refutation R with B-answer substitution θ. By specialization lemma (Lemma 1-2-11), $P \cup \{G\}$ has a special B-refutation R_s with B-answer substitution $\theta\gamma$ for a suitable γ. Then, using the above Lemma 1-3-10, we notice that R_s is a F-B-refutation. $\square$

§1-4. Fixed point semantics of LIFE-III

In this section, we give a fixed point characterization of the least Herbrand F-model for the preparation of a proof of the completeness theorem of LIFE-III. Fist of all, for the sake of convenience, let's define;

Definition 1-4-1. Let P be a program.
For any $\sigma \in B_P$, let $C(\sigma) = \{C \in P \mid (\exists \theta$: ground substitution for $C)(\sigma = C\theta)\}$. ⊣

Now, since we are going to define B-valued fixed point semantics, we ought to begin to consider B^{B_P} instead of the power set $p(B_P)$ of B_P, i.e., 2^{B_P}.

Definition 1-4-2. Let P be a program.
Define relations $\leqq_F$ and $\equiv_F$ over B^{B_P} by, for any $f, g \in B^{B_P}$,

$$f \leqq_F g \quad \text{iff} \quad \bigwedge_{\sigma \in B_P} (f(\sigma) \rightarrow g(\sigma)) \in F$$

and

$$f \equiv_F g \quad \text{iff} \quad f \leqq_F g \ \wedge \ g \leqq_F f.$$

Let $P_P = (B^{B_P} / \equiv_F)$. Define a partial order $\leqslant_F$ over P_P by, for any $[f], [g] \in P_P$,

$[f] \leqslant_F [g]$ iff $f \leqq_F g$ for suitable representatives f, g for [f], [g] respectively. ⊣

Obviously, the above definition of $\leqslant$ does not depend on the choice of representatives f, g for [f], [g] and so is well-defined.

With this notation, we get;

Proposition 1-4-3. Let P be a program. Then, $(P_P, \leqslant_F)$ is a complete lattice.

Proof: It is obvious that $(P_P, \leqslant_F)$ is partially ordered. Let X be a subset of P_P. Define

$$\bigvee X = [g] \quad \text{by} \quad (\forall \sigma \in B_P)\ (g(\sigma) = \bigvee_{[f] \in X} f(\sigma))$$

and

$$\wedge X = [h] \qquad \text{by} \quad (\forall \sigma \in Bp)\ (h(\sigma) = \bigwedge_{[f] \in X} f(\sigma))\ .$$

where f is a suitable representative of [f]. Then, it is easy to check that $\vee X$ is the l.u.b. of X and $\wedge X$ is the g.l.b. of X. (Again, the definition of $\vee$ and $\wedge$ does not depend on the choice of each representative f of [f].) □

Note: The top element of $(P_P, \leqslant_P)$ is [T] such that

$$(\forall \sigma \in B_P)(\ T(\sigma) = 1\ )$$

and the bottom element is $[\perp]$ such that

$$(\forall \sigma \in B_P)(\ \perp(\sigma) = 0\ )$$ ⊣

Definition 1-4-4. Let P be a program. The mapping

$$T_P : P_P \to P_P$$

is defined as follows. For any $[f] \in P_P$, $T_P([f]) = [g]$ such that

$$g(\sigma) = \begin{cases} 1\ , & \text{if there is an unit clause in } C(\sigma) \\ \bigvee_{C_i \in C(\sigma)} f(C_i^- \theta_i)\ , & \text{if } C(\sigma) \neq \varnothing \text{ and there is no unit clause in } C(\sigma) \\ 0\ , & \text{otherwise,} \end{cases}$$

where $f(C_i^- \theta_i) = f(A_1 \theta_i) \wedge \cdots \wedge f(A_k \theta_i)$, when $C_i^- = \{A_1, \cdots, A_k\}$ and $C_i^+ \theta_i = \sigma$.

⊣

There is another characterization of $T_P : P_P \to P_P$.

Lemma 1-4-5. Let P be a program and $T_P : P_P \to P_P$ be as above. Then,

$$T_P([f]) = \wedge \{[g] \mid (\forall C \in P)(\forall \theta\text{: ground substitution for } C)$$
$$((g(C^+\theta) \leftarrow f(C^-\theta)) \in F)\}.$$

Proof: Let $T_P : P_P \to P_P$ be as above. Let $[f] \in P_P$, $T_P([f]) = [G]$,

$X = \{[g] \mid (\forall C \in P)\ (\forall \theta\text{: ground substitution for } C)\ ((g(C^+\theta) \leftarrow f(C^-\theta)) \in F)\}$, and

$[h] = \wedge X$

We want to show $[G]=[h]$.

To show $[h] \leqslant_F [G]$, it is enough to check $[G] \in X$.

Let $C \in P$ and θ be a ground substitution for C. Let $C^+\theta=\sigma$.

By definition, since $C(\sigma) \neq \varnothing$,

$$G(C^+\theta)=G(\sigma)= \begin{cases} 1, \text{ if there is an unit clause in } C(\sigma) \\ \bigvee_{C_i \in C(\sigma)} f(C_i^-\theta_i) \geqq f(C^-\theta), \text{ otherwise.} \end{cases}$$

So, $(G(C^+\theta) \leftarrow f(C^-\theta))=1$.

$\therefore$ $[G] \in X$.

Next, to show $[G] \leqslant_F [h]$, we devide the argument into two cases. Let $\sigma \in Bp$ be arbitrary.

I. The case that $C(\sigma) \neq \varnothing$

i) The case that there is an unit clause in $C(\sigma)$.

Let $[g] \in X$ be arbitrary. Since

$(\forall C \in P)$ $(\forall \theta$: ground substitution) $((g(C^+\theta) \leftarrow f(C^-\theta)) \in F)$,

especially taking $C=\{A\leftarrow\}$ and θ such that $A\theta=\sigma$, we notice that

$$g(\sigma)=g(A\theta) \in F.$$

$\therefore$ $h(\sigma)=\bigwedge_{[g] \in X} g(\sigma) \in F$, because F is complete.

So, $(h(\sigma) \leftarrow G(\sigma)) \in F$.

ii) Otherwise

Let $[g] \in X$ be arbitrary. Since

$(\forall C \in P)$ $(\forall \theta$: ground substitution) $((g(C^+\theta) \leftarrow f(C^-\theta)) \in F)$,

$(g(\sigma) \leftarrow f(C_i^-\theta_i)) \in F$ for any $C_i \in C(\sigma)$ and θ_i such that $\sigma=C_i^+\theta_i$.

$\Rightarrow$ $(g(\sigma) \leftarrow \bigvee_{C_i \in C(\sigma)} f(C_i^-\theta_i)) \in F$.

$\Rightarrow$ $(g(\sigma) \leftarrow G(\sigma)) \in F$.

So, $((\bigwedge_{[g] \in X} g(\sigma)) \leftarrow G(\sigma)) \in F$, i.e., $(h(\sigma) \leftarrow G(\sigma)) \in F$. ⊣

II. The other case, i.e., $C(\sigma)=\varnothing$

By definition, $G(\sigma) = \mathbf{0}$.

$\therefore$ $(h(\sigma) \leftarrow G(\sigma)) \in F$. $\dashv$

So, I + II gives $[G] \leqslant_F [h]$. □

In the following, we will use these two definitions of T_P interchangeably.

Lemma 1-4-6. Let P be a program. Then,

$T_P : P_P \rightarrow P_P$ is monotonic.

Proof: Let $[f], [g] \in P_P$ be such that $[f] \leqslant_F [g]$. We want to show $T_P([f]) \leqslant_F T_P([g])$, that is, $\bigwedge Y_f \leqslant_F \bigwedge Y_g$, where

$$Y_f = \{[h] \mid (\forall C \in P)(\forall \theta\text{: ground substitution})((h(C^+\theta) \leftarrow f(C^-\theta)) \in F)\}$$

and

$$Y_g = \{[h] \mid (\forall C \in P)(\forall \theta\text{: ground substitution})((h(C^+\theta) \leftarrow g(C^-\theta)) \in F)\}.$$

To show this, it is enough to check $Y_f \supset Y_g$.

Now, by definition,

$(\forall \sigma \in B_P)((f(\sigma) \rightarrow g(\sigma)) \in F)$.

$\therefore$ $(\forall C \in P)(\forall \theta\text{: ground substitution})((f(C^-\theta) \rightarrow g(C^-\theta)) \in F)$. … (1)

Let $[h] \in Y_g$. Then, for any $C \in P$ and any ground substitution θ,

$(h(C^+\theta) \leftarrow g(C^-\theta)) \in F$

$\Rightarrow$ $(h(C^+\theta) \leftarrow f(C^-\theta)) \in F$ by (1).

So, $[h] \in Y_f$. $\therefore$ $Y_f \supset Y_g$. □

Proposition 1-4-7. Let P be a program and (Up, f) be a B-valued Herbrand interpretation of P. Then, (Up, f) is a F-model of P iff $T_P([f]) \leqslant_F [f]$.

Proof; ($\Rightarrow$): Let (Up, f) be a F-model of P. Then,

$(\forall C \in P)(\forall \theta$: ground substitution for C$)((f(C^+\theta) \leftarrow f(C^-\theta)) \in F)$.

$\Rightarrow$ $[f] \in X$, where

$X = \{[g] \mid (\forall C \in P)(\forall \theta$: ground substitution for C) $((g(C^+\theta) \leftarrow f(C^-\theta)) \in F)\}$.

$\Rightarrow \quad Tp([f]) \leqslant_F [f]$.

($\Leftarrow$): Suppose (Up, f) is not a F-model of P. This means

$(\exists C \in P)(\exists \theta$: ground substitution) $((f(C^+\theta) \leftarrow f(C^-\theta)) \notin F)$. ... (1)

Let $X = \{[g] \mid (\forall C \in P)(\forall \theta$: ground substitution) $((g(C^+\theta) \leftarrow f(C^-\theta)) \in F)\}$.

Then, $Tp([f]) = \wedge X$. Let $\wedge X = [h]$.

Here, using C and θ in (1), we get

$(\forall [g] \in X)((g(C^+\theta) \leftarrow f(C^-\theta)) \in F)$.

$\Rightarrow \quad (\bigwedge_{[g] \in X} g(C^+\theta) \leftarrow f(C^-\theta)) \in F$.

$\Rightarrow \quad (h(C^+\theta) \leftarrow f(C^-\theta)) \in F$. ... (2)

(1) + (2) gives $(f(C^+\theta) \leftarrow h(C^+\theta)) \notin F$.

$\Rightarrow \quad (\bigwedge_{\sigma \in B_P} (f(\sigma) \leftarrow h(\sigma)) \notin F$.

$\Leftrightarrow \quad [h] \not\leqslant_F [f]$.

$\Leftrightarrow \quad Tp([f]) \not\leqslant_F [f]$. □

As a corollary, we get the following fixed point characterization of the least Herbrand F-model.

Theorem 1-4-8. ($\wedge$ fixed point characterization of the least Herbrand F-model) Let P be a program and (Up, μ) be the least Herbrand F-model of P. Let T_P: $P_P \rightarrow P_P$ be as before. Then,

$[\mu]$ = the least fixed point of T_P.

Proof: Let (Up, μ) be the least Herbrand F-model of a program P. Then, by definition,

$(\forall \sigma \in B_P)(\mu(\sigma) = \wedge \{f_i(\sigma) \mid (U_P, f_i)$ is a Herbrand F-model of P$\}$).

Let $X = \{[f_i] \mid (U_P, f_i)$ is a Herbrand F-model of P$\}$. Then,

$[\mu] \quad = \quad \wedge X$

$\quad = \quad \wedge \{[f_i] \mid Tp([f_i]) \leqslant_F [f_i]\}$, by Proposition 1-4-7

$=$ the least fixed point of Tp,

because Pp is a complete lattice and Tp is monotonic. □

Lemma 1-4-9. Tp is continuous.

Proof: Let $[g_0] \leqslant_F [g_1] \leqslant_F \cdots \leqslant_F [g_n] \leqslant_F \cdots$ be an increasing sequence of Pp.

We want show

$Tp(\bigvee_{n<\omega} [g_n]) = \bigvee_{n<\omega} Tp([g_n])$.

Let $[G] = Tp(\bigvee_{n<\omega} [g_n])$ and $[H] = \bigvee_{n<\omega} Tp([g_n])$. Let $\sigma \in Bp$ be arbitrary. It is enough to check $(G(\sigma) \leftrightarrow H(\sigma)) \in F$.

i) The case that $C(\sigma) = \varnothing$.

By definition of Tp, $G(\sigma) = 0 = H(\sigma)$.

ii) The case that there is an unit clause in $C(\sigma)$.

By definition of Tp, $G(\sigma) = 1 = H(\sigma)$.

iii) The other case.

$$G(\sigma) = \bigvee_{C_i \in C(\sigma)} h(C_i^- \theta_i) \text{ , where } [h] = \bigvee_{n<\omega} [g_n].$$

($\Leftarrow$ use the fact that $\cdots \leqslant_F [g_n] \leqslant_F \cdots$ are increasing)

$$= \bigvee_{C_i \in C(\sigma)} (\bigvee_{n<\omega} g_n(C_i^- \theta_i)). \quad \cdots ①$$

$$H(\sigma) = \bigvee_{n<\omega} Tp([g_n])$$

$$= \bigvee_{n<\omega} (\bigvee_{C_i \in C(\sigma)} g_n(C_i^- \theta_i)). \quad \cdots ②$$

Here, the equality ① = ② can be shown easily, using the fact that B is complete.

□

Definition 1-4-10. Let Pp and $Tp : Pp \to Pp$ be as before.

Define $Tp \uparrow 0 = [\bot]$

$Tp \uparrow n+1 = Tp(Tp \uparrow n)$

$$T_P \uparrow \omega = \bigvee_{n<\omega} T_P \uparrow n .$$ ⊣

Lemma 1-4-11.

$(\forall n \leqq \omega) (\forall \sigma \in Bp) (f_n(\sigma) = 1 \vee f_n(\sigma) = 0)$, where f_n is a suitable representative of $Tp \uparrow n$ for $n \leqq \omega$.

Proof: Obvious. (By induction on the construction of $Tp \uparrow n$.) □

Proposition 1-4-12. $(\forall n < \omega) (Tp \uparrow n+1 \geqslant_F Tp \uparrow n)$.

Proof: Since Tp is monotonic, this is obvious. □

As a corollary, we notice

Corollary 1-4-13. $T_P \uparrow \omega$ is the least fixed point of T_P.

Proof: By continuity of Tp, this is obvious. □

§1-5. Completeness of LIFE-III

In this section, we prove the completeness of LIFE-III in its B-valued sense. First of all, using the results of theorem 1-4-8 and corollary 1-4-13, we can obtain the following completeness theorem of the least Herbrand *F*-model version.

Theorem 1-5-1. (Completeness theorem of LIFE-III—the least Herbrand *F*-model form—). Let P be a program. Then,

$$B\mu [F] = S_P(\mu , \sim),$$

where (U_P, μ) is the least Herbrand *F*-model of P and $\sim$ is arbitrary.

Proof: By soundness theorem (Corollary 1-3-6), we have already obtained the direction

$$Sp(\mu, \sim) \subseteq B\mu\,[F]\,. \quad \cdots \text{(a)}$$

Let $\sigma \in B\mu[F]$, i.e., $\mu(\sigma) \in F$. We aleady know that $[\mu]$ = the least fixed point of $Tp = Tp \uparrow \omega$.

$\therefore \quad (\exists n < \omega)\,(f_n(\sigma) \in F)$, where $[f_n] = Tp \uparrow n$.

(Remember $f_n(\sigma) = 1$ or 0).

In the following, we prove by induction on n that

$f_n(\sigma) \in F$, i.e., $f_n(\sigma) = 1 \Rightarrow \sigma \in Sp$, where Sp is in Definition 1-3-4.

First of all, by assumption that $f_n(\sigma) = 1$, we notice that $C(\sigma) \neq \varnothing$ by definition of Tp. In the following, w use this fact without mentioning.

i) The case of $n = 1$

Let $\sigma \in Bp$ be such that $f_1(\sigma) = 1 \in F$.

By definition,

$$f_1(\sigma) = \begin{cases} 1, & \text{if there is an unit clause in } C(\sigma) \\ \bigvee_{C_i \in C(\sigma)} \perp (C_i^- \theta_i), & \text{otherwise.} \end{cases}$$

Since $\perp (C_i^- \theta_i) = 0$ for any $C_i \in C(\sigma)$ and θ_i such that $\sigma = C_i^+ \theta_i$, $C(\sigma)$ has at least one unit clause because $f_1(\sigma) = 1$.

Let $\{A \leftarrow\} \in C(\sigma)$ and θ be such that $\sigma = A\theta$.

Then, $P \cup \{\leftarrow \sigma\}$ has a refutation of length 1. ⊣

ii) General case

Suppose the result holds for $n - 1$. Let $\sigma \in Bp$ be such that $f_n(\sigma) = 1 \in F$.

Claim. $(\exists C \in C(\sigma))\,(\exists \theta$: ground substitution for C$)\,(\sigma = C^+\theta \wedge$

$(C^- = \varnothing \vee f_{n-1}(C^-\theta) = 1))$.

⌈ Suppose $(\forall C \in C(\sigma))(\forall\theta$: ground substitution for C) $((\sigma = C^+\theta) \rightarrow (C^- \neq \varnothing \wedge$

$f_{n-1}(C^-\theta) \neq 1))$.

Then, by definition,

$$f_n(\sigma) = \bigvee_{C_i \in C(\sigma)} f_{n-1}(C_i^-\theta_i) = 0 .$$

This contradicts the assumption $f_n(\sigma) = 1$. ⊣ ⌋

(I) The case of $(\exists(A\leftarrow) \in C(\sigma))(\exists\theta$: ground substitution) $(\sigma = A\theta)$.

In this case, $P \cup \{\leftarrow\sigma\}$ has a refutation of length 1 with the input clause $\{A\leftarrow\}$ and the syntactical unifier θ. ⊣

(II) The other cases, that is,

$(\exists C \in C(\sigma))(\exists\theta$: ground substitution) $(\sigma = C'\theta \wedge f_{n-1}(C^-\theta) = 1)$. ⋯ (b)

Let $C^- = \{A_1, \cdots, A_k\}$. By induction hypothesis,

$A_i\theta \in Sp$ for $1 \leq i \leq k$

because $f_{n-1}(A_1\theta) \wedge \cdots \wedge f_{n-1}(A_k\theta) = 1$.

So, $P \cup \{\leftarrow\sigma\}$ has a refutation with the first input clause C and the syntactical unifier θ, where C and θ are as in the above (b). ⊣

(I) + (II) gives $f_n(\sigma) \in F \rightarrow \sigma \subset Sp$.

What we have shown is

$$B\mu[F] \subseteq Sp. \quad \cdots \text{(c)}$$

(a) + (c) gives $B\mu[F] \subseteq Sp \subseteq Sp(\mu, \sim) \subseteq B\mu[F]$.

So, $B\mu[F] = Sp(\mu, \sim)$. □

As a direct consequence, we notice

Corollary 1-5-2. Let P be a program. Then,

$S_P = S_P(\mu, \sim)$,

where (U_P, μ) is the least Herbrand F-model.

Proof: Obvious. □

Now , to prove a completeness theorem for F-program (P, [[ ]]) in a general situation, we need a few lemmas. The first one is a direct generalization of the famous result in pure Prolog.

Lemma 1-5-3. (Lifting lemma for B-derivation)
Let (P, [[ ]]) be a B-valued program, G be a goal and θ be a substitution. If there is a B-refutation of $P \cup \{G\theta\}$ with B-mgus $<\theta_1, \cdots, \theta_n>$, then there exists a B-refutation of $P \cup \{G\}$ with B-mgus $<\theta_1', \cdots, \theta_n'>$, such that
$(\exists\gamma$: substitution) $(\theta\theta_1 \cdots \theta_n = \theta_1' \cdots \theta_n'\gamma)$.

Proof: Using Lemma 1-2-10 (B-mgu Lemma), the proof is just the same as usual case. □

The second also comes from the corresponding result in SLD-resolution.

Lemma 1-5-4. Let (U_P, μ) be the least Herbrand F-model of P. Suppose V(A) is a F-logical consequence of P. Then, there is a (usual) refutation of $P \cup \{\leftarrow A\}$ with the identity substitution as the answer substitution.

Proof: Let V(A) be a F-logical consequence of P. Then, using the same argument as in pure Prolog case, we can assert that there is a refutation of $P \cup \{\leftarrow A\}$ with the identity answer substitution by theorem 1-5-1 and Corollary 1-5-2. □

Now, we are ready to prove

Theorem 1-5-5. (Completeness theorem of LIFE-III)

Let (P, [[ ]]) be a B-valued *F*-program and G be a goal. For every *F*-correct answer substitution θ for P∪{G}, there is an answer substitution θ' for P∪{G} and a substitution γ' such that θ = θ'γ'.

Proof: Using Lemma 1-5-4, the proof is similar to the usual 2-valued case. □

Corollary 1-5-6. Let (P, [[ ]]) be a B-valued *F*-program and G be a goal. For every *F*-correct answer substitution θ for P∪{G}, there is a refutation of P∪{Gθ} with the identity answer substitution.

Proof: This proof is included in the above proof of theorem1- 5-5. □

Corollary 1-5-7. Let (P, [[ ]]) be a B-valued *F*-program and G be a goal. For every *F*-correct answer substitution θ for P∪{G}, there is a B-refutation of P∪{G} with B-answer substitution θ' with respect to [[ ]] and an arbitrary ~ such that

$$(\exists\gamma')\,(\theta = \theta'\gamma')\,.$$

Proof: Since every refutation is also a B-refutation w.r.t. any [[ ]] and ~, this is a direct consequence of the above theorem 1-5-5. □

So far, for any program P, we have obtained the inclusion relations

$$B_\mu[F] = S_P = S_P(\mu, \sim) \subseteq S_P([[\ ]], \sim) \subseteq B_{[[\ ]]}[F]$$

① ② ③ ④

where (U_P, μ) is the least Herbrand *F*-model of P and $(U_P, [[\ ]])$ is an arbitrary Herbrand *F*-model of P and ~ is arbitrary.

Reasons:

① Theorem 1-5-1

② Corollary 1-5-2

③ Obvious

④ Proposition 1-3-5. ⊣

Here, a little observation tells us that equality relation in ③ and ④ does not hold generally. To tell the truth, the possibility of equality in ③ and ④ heavily depends not only ~ and [[ ]]: Bp→B but also the type of a program P itself. Precise arguments concerning the equality at ④ are discussed in the next chapter.

In this chapter, we have defined the notions of Boolean-valued logic programming language scheme LIFE-III as components of LIFE-Ω and proved both the soundness and the completeness in their suitable senses.

Chapter II Relativized Completeness

§2-1. Preliminaries

Definition 2-1-1. Let $Mp(B, F) = \{[f] \in Pp \mid (Up, f)$ is a Herbrand F-model of P$\}$. ⊣

In chapter I, we showed that, for any $[f] \in Mp(B, F)$,

$$B\mu[F] = S_P = Sp(\mu, \sim) \subseteq Sp(f, \sim) \subseteq B_f[F].$$

① ② ③ ④

The above results assert that:

① The least Herbrand F-model is the standard model concerning usual syntactical unification over P.

At the same time,

② The least Herbrand F-model (U_P, μ) is the standard model concerning B-valued unification over P w.r.t. μ and $\sim$, where $\sim$ is an arbitrary substitution transitive relation. That is, for any $\sigma \in B_P$,

$((U_P, \mu) \models \sigma$ iff$)$ $\sigma \in B\mu[F]$ iff $\sigma \in Sp(\mu, \sim)$

(iff $(P, \mu) \vdash \sigma$) ,where "$(P, \mu) \vdash$" is the B-derivation from P based on B-valued unification w.r.t. μ and $\sim$.

③ Let $[f] \in Mp(B, F)$ and $\sim$ be arbitrary. Then, B-valued unification over P w.r.t. f and $\sim$ is richer than B-valued unification over P w.r.t. μ and $\sim$. That is, for any $\sigma \in B_P$,

$(P, \mu) \vdash \sigma \Rightarrow (P, f) \vdash \sigma$.

However, the converse does not always hold.

④ Let [f] $\in$ Mp(B, F) and $\sim$ be arbitrary. Then, for any $\sigma \in B_P$,

$$(P, f) \vdash \sigma \quad \text{iff} \quad \sigma \in S_P(f, \sim)$$
$$\Rightarrow \quad \sigma \in B_{[f]}[F] \quad \text{iff} \quad (U_P, f) \models \sigma \ .$$

① claims the fact that the least Herbrand F-model is essentially the same as the least Herbrand model in the usual 2-valued sense.

② suggests that Boolean-valued unification w.r.t. the least [μ] $\in$ Mp(B, F) and an arbitrary substitution transitive $\sim$ is essentially the same as the usual syntactical unification for any B, F.

③ , on the other hand, asserts the fundamental difference between the usual syntactical unification and Boolean-valued unification w.r.t. arbitrary [f] $\in$ Mp(B, F) and substitution transitive $\sim$. So, ② asserts that [μ] works, in a sense, as an extreme case concerning B-valued unification.

④ finally claims the logical soundness of the inference based on B-valued unification w.r.t. an arbitrary [f] $\in$ Mp(B, F) and an arbitrary substitution transitive $\sim$ for any B, F.

Here, a natural question is:

"Except the above case ②, under which condition over f and $\sim$, can we obtain the equality?"

The main purpose of this chapter is to present reasonable and useful conditions over (f, $\sim$) which become answers to the above question. The advantage of the results should be clear. Theoretically, it offers a class $\mathbf{C}$ of B-

valued program (P, [[ ]]) with certain kinds of properties which can never be characterized by the usual 2-valued philosophy, and the logical soundness and completeness assures the existence of the standard element of C just in the same sense as the least Herebrand model is the standard element of the total class {(P, f) | (Up, f) is a Herbrand model of P in the 2-valued sense}.

For this purpose, borrowing the technique which is used to show that the least element [μ] of Mp(B, F) satisfies Bμ[F] = Sp(μ, ~) for any ~, we employ the following strategy. First of all, define ~, B, F which are expected to be suitable for our aim. Secondly choose a proper subset N ⊆ Pp such that (N, $<_P$) becomes a complete sublattice. Lastly, take the least element [μ_N] of N∩Mp(B, F) and compare $B_{\mu_N}[F]$ and Sp(μ_N, ~).

In the next section, we propose a general and systematic method which can characterize N ⊆ Pp such that

$$S_P(\mu_N, \sim) = B\mu_N[F]$$

for any kind of ~, B, F, where ~ becomes an equivalence relation.

§2-2. J-faithfulness

Definition 2-2-1. Let J : Bp→Bp be an idempotent function,

i.e., $(\forall \sigma \in B_P)(J(J(\sigma)) = J(\sigma))$.

(1) Let ~ be an equivalence relation which is used to define B-valued unification. Then,

J satisfies B-valued unification condition w.r.t.~ iff

$(\forall \sigma \in B_P)(\sigma \sim J(\sigma))$.

(2) $f \in B^{B_P}$ is J-faithful iff

$(\forall \sigma \in B_P)(f(\sigma) = f(J(\sigma)))$.

(3) A B-valued program (P, [[ ]]) is J-faithful iff

[[ ]] is J-faithful.

(4) A Herbrand F-model (U_P, f) is J-faithful iff

f is J-faithful.

(5) A subset $\Gamma_P(J)$ of P_P is defined as

$\Gamma_P(J) = \{ [f] \in P_P \mid f \text{ is J-faithful} \}$.

(6) (U_P, μ_J) is the least J-faithful Herbrand F-model of P

iff $[\mu_J]$ is the least element of $(Mp(B, F) \cap \Gamma_P(J), <_F)$, that is,

$(\forall \sigma \in B_P) (\mu_J(\sigma) = \wedge \{ f(\sigma) \mid (U_P, f)$ is a J-faithful Herbrand

F-model of P $\})$. ⊣

Throughout this section, we always assume that B, F, $\sim$ and J defined as above are arbitrary but fixed and J satisfies B-valued unification condition w.r.t. $\sim$. In addition, we assume that

$(\forall [f] \in \Gamma_P(J))$ (a representative f of [f] is J-faithful).

Proposition 2-2-2. $\Gamma_P(J)$ is closed w.r.t. $\vee$, $\wedge$, that is,

$(\Gamma_P(J), \leqslant_F)$ is a complete lattice.

Proof: Obvious. □

Definition 2-2-3. Let $T_P(J) : P_P \to P_P$ be such that, for any $[f] \in P_P$,

$T_P(J) ([f]) = [G]$, where

$$G(\sigma) = \begin{cases} 1 & \text{, if there is an unit clause in } D(J(\sigma)) \\ \bigvee_{C_i \in D(J(\sigma))} f(C_i^- \theta_i) & \text{, if } D(J(\sigma)) \neq \varnothing \wedge \text{there is no unit clause in} \end{cases}$$

$$\begin{cases} & D(J(\sigma)) \\ 0 & \text{, otherwise} \end{cases}$$

and $D(J(\sigma)) = \{C \in P \mid (\exists\theta\text{: ground substitution})(J(C^+\theta) = J(\sigma) \text{ and } C^+\theta \sim J(\sigma)\}$. ⊣

Proposition 2-2-4. For any $[f] \in P_P$, $T_P(J)([f]) \in \Gamma_P(J)$.

Proof: Let $[f] \in P_P$ and $T_P(J)([f]) = [G]$.

For any $\sigma \in Bp$, $G(\sigma)$ is defined by means of the terminology $D(J(\sigma))$ and $J(\sigma) = J^2(\sigma)$ by idempotentness.

$\therefore$ $G(\sigma) = G(J(\sigma))$ for any $\sigma \in Bp$.

$\therefore$ $[G] \in \Gamma_P(J)$. □

Lemma 2-2-5. Let $T_P(J) : P_P \to P_P$ be as above. Let $[f] \in \Gamma_P(J)$ and $T_P(J)([f]) = [G]$. Moreover, let

$$[h] = \wedge \{[g] \in \Gamma_P(J) \mid (\forall C \in P)(\forall\theta\text{: ground substitution for } C)((g(C^+\theta) \leftarrow f(C^-\theta)) \in F)\}.$$

Then, $[h] = [G]$.

Proof: Let $T_P(J)$, $[G]$, $[h]$ be as in the assumption.

Let $X = \{[g] \in \Gamma_P(J) \mid (\forall C \in P)(\forall\theta\text{: ground substitution for } C)((g(C^+\theta) \leftarrow f(C^-\theta)) \in F)\}$.

To show $[h] \leqslant_F [G]$, it is enough to check $[G] \in X$.

Let $C \in P$ and θ is a ground substitution for C. Let $C^+\theta = \sigma$. By definition, since $C \in D(J(\sigma))$, $D(J(\sigma)) \neq \varnothing$. So,

$$G(C^+\theta) = G(\sigma) = \begin{cases} 1 & \text{, if there is an unit clause in } D(J(\upsilon)) \\ f(C_i^- \theta_i) & \text{, otherwise.} \\ \bigvee_{C_i \in D(J(\upsilon))} & \end{cases}$$

In anyway,

$$(G(C^+\theta) \leftarrow f(C^-\theta)) \in F.$$

$$\therefore \quad [G] \in X.$$

Next, to show $[G] \leqslant_F [h]$, it is enough to see

$$(\forall \sigma \in Bp)((h(\sigma) \leftarrow G(\sigma)) \in F).$$

Let $\sigma \in Bp$ be arbitrary.

i) The case that there is an unit clause in $D(J(\sigma))$.

Let $C \in D(J(\sigma))$ be such that $C = \{A \leftarrow\}$ and θ be a ground substitution such that $J(A\theta) = J(\sigma)$. Then, for any $[g] \in X$, $g(\upsilon) = g(J(\upsilon)) = g(J(A\theta)) = g(A\theta) = g(C^+\theta) \in F$ by definition of X.

So, $\quad h(\sigma) = \bigwedge_{[g] \in X} g(\sigma) \subset F$

$\Rightarrow \quad (h(\sigma) \leftarrow G(\sigma)) \in F.$

ii) The case that $D(J(\sigma)) \neq \varnothing$ but there is no unit clause in $D(J(\sigma))$.

Let $[g] \in X$ be arbitrary. Since $(\forall C \in P)$ $(\forall \theta$: ground substitution)

$((g(C^+\theta) \leftarrow f(C^-\theta)) \in F)$,

$(g(C_i^+\theta_i) \leftarrow f(C_i^-\theta_i)) \in F$ for any $C_i \in D(J(\sigma))$ and θ_i s.t. $J(C_i^+\theta_i) = J(\sigma)$

$\Rightarrow \quad (g(J(C_i^+\theta_i)) \leftarrow f(C_i^-\theta_i)) \in F$ for any $C_i \in D(J(\sigma))$ and θ_i s.t. $J(C_i^+\theta_i) = J(\sigma)$

$\Rightarrow \quad (g(J(\sigma)) \leftarrow \bigvee_{C_i \in D(J(\upsilon))} f(C_i - \theta_i)) \in F$

$\Rightarrow \quad (g(J(\sigma)) \leftarrow G(\upsilon)) \in F$

$\Rightarrow \quad (g(\sigma) \leftarrow G(\sigma)) \in F$

$\therefore \quad (\bigwedge_{g \in ex} g(\sigma) \leftarrow G(\sigma)) \in F$

$\Rightarrow \quad (h(\sigma) \leftarrow G(\sigma)) \in F.$

iii) The other case.

By definition, since $G(\sigma) = 0$,

$(h(\sigma) \leftarrow G(\sigma)) \in F$ is obvious.

i) + ii) + iii) give the required result. □

In the following, we consider only the case of $T_P(J) | \Gamma_P(J)$ without noting "$| \Gamma_P(J)$" explicitly and use the above two definitions interchangeably.

Proposition 2-2-6. $T_P(J)$ is monotonic.

Proof: Similar to the proof of Lemma 1-4-6 in chapter I. □

Proposition 2-2-7. Let (U_P, f) be a B-valued J-faithful Herbrand interpretation of P. Then,

(U_P, f) is a F-model of P iff $T_P(J)([f]) \leqslant_F [f]$.

Proof: Similar to the proof of Proposition 1-4-7 in chapter I. □

As a corollary, we get the following fixed point characterization of the least J-faithful Herbrand *F*-model.

Corollary 2-2-8. Let P be a program and (U_P, μ_J) be the least J-faithful Herbrand *F*-model of P. Let $T_P(J) : \Gamma_P(J) \to \Gamma_P(J)$ be as above.
Then,

$[\mu_J]$ = the least fixed point of $T_P(J)$.

Proof: Similar to the proof of Theorem 1-4-8 in chapter I. □

Lemma 2-2-9. Tp(J) is continuous.

Proof: Similar to the proof of Lemma 1-4-9 in chapter I. □

Definition 2-2-10. Let $\Gamma_P(J)$ and $T_P(J) : \Gamma_P(J) \to \Gamma_P(J)$ be as before.

Define $T_P(J) \uparrow 0 = [\bot]$, where $(\forall \sigma \in B_P)(\bot(\sigma) = 0)$.

$T_P(J) \uparrow n+1 = T_P(J)(T_P(J) \uparrow n)$

$T_P(J) \uparrow \omega = \bigvee_{n<\omega} T_P(J) \uparrow n$ ⊣

By definition of ⊥ and Proposition 2-2-2 + Proposition 2-2-4 + Proposition 2-2-6,

$(\forall n \leq \omega)(T_P(J) \uparrow n \in \Gamma_P(J))$

and

$(\forall n<\omega)(T_P(J) \uparrow n+1 \geqslant_F T_P(J) \uparrow n)$

are obvious. Moreover,

$$(\forall n \leqq \omega)(\forall \sigma \in Bp)(f_n(\sigma) = 1 \text{ or } f_n(\sigma) = 0)$$

where f_n is a suitable representative of $Tp(J) \uparrow n$ for $n \leqq \omega$, is also obvious.

So, combining the above facts with Lemma 2-2-9, we obtain

Proposition 2-2-11. $Tp(J) \uparrow \omega$ is the least fixed point of $Tp(J)$.

Proof: Similar to the proof of Corollary 1-4-13 in chapter I. □

Now, we are ready to prove a completeness theorem for J-faithful programs.

Theorem 2-2-12. Let P be a program and (U_P, μ_J) be the least J-faithful Herbrand *F*-model of P. Then,

$$B_{\mu_J}[F] = Sp(\mu_J, \sim).$$

(Compare Theorem 1-5-1 in chapter I.)

Proof: By the soundness theorem, we already know that

$Sp(\mu_J, \sim) \subseteq B_{\mu_J}[F]$.

So, it is enough to see

$B_{\mu_J}[F] \subset Sp(\mu_J, \sim)$.

As stated before, we tacitly assume that J satisfies B-valued unification condition w.r.t. $\sim$. Let $\sigma \in B\mu_J[F]$, that is, $\mu_J(\sigma) \in F$.

We already know that $[\mu_J] = Tp(J) \uparrow \omega$.

$\therefore \quad (\exists n < \omega)\,(f_n(\sigma) = 1)$, where $[f_n] = Tp(J) \uparrow n$.

In the following, we prove by induction on n that

$f_n(\sigma) = 1 \Rightarrow \sigma \in Sp(\mu_J, \sim)$.

Since $f_0(\sigma) = 0$ for any $\sigma \in Bp$, we can start the induction hypothesis from the case of $n = 1$.

i) The case of $n = 1$

First of all, by assumption that $f_1(\sigma) = 1$, we notice that $D(J(\sigma)) \neq \varnothing$ by definition of Tp(J).

$$\therefore \quad f_1(\sigma) = \begin{cases} 1 & \text{, if there is an unit clause in } D(J(\sigma)) \\ \bigvee_{C_i \in D(J(\sigma))} \perp(C_i^{-}\theta_i) & \text{, otherwise.} \end{cases}$$

Since $\perp(C_i^{-}\theta_i) = 0$ for any $C_i \in D(J(\sigma))$ and θ_i, $D(J(\sigma))$ must have at least one unit clause, because $f_1(\sigma) = 1$.

Let $\{A\leftarrow\} \in D(J(\sigma))$ and θ be such that $J(\sigma) = J(A\theta)$ and $J(\sigma) \sim A\theta$.

Since $[\mu_J] \in \Gamma p(J)$,

$\mu_J(\sigma) = \mu_J(J(\sigma)) = \mu_J(J(A\theta)) = \mu_J(A\theta)$.

Here, both σ and $A\theta$ are ground, and $\sigma \sim J(\sigma) \sim A\theta$.

So, θ B-valued unifies σ and A w.r.t. μ_J and $\sim$.

$\therefore \quad \sigma \in Sp(\mu_J, \sim)$.

ii) General case

Suppose the result holds for $n-1$ and $f_n(\sigma)=1$.

By definition,

$$f_n(\sigma)=\begin{cases} 1 & \text{, if there is an unit clause in } D(J(\sigma)) \\ \bigvee_{C_i \in D(J(\sigma))} f_{n-1}(C_i^- \theta_i) & \text{, if } D(J(\sigma)) \neq \emptyset \wedge \text{ there is no unit clause in } D(J(\sigma)). \end{cases}$$

I. The case that there is an unit clause in $D(J(\sigma))$.

Using the same argument as in i), we notice that

$$\sigma \in Sp(\mu_J, \sim).$$

II. The other cases, that is,

$(\exists C \in D(J(\sigma)))(\exists \theta$: ground substitution$)\,(J(\sigma)=J(C^+\theta) \wedge J(\sigma) \sim C^+\theta \wedge$

$f_{n-1}(C_i^-\theta)=1)$. $\cdots$ (1)

Let $C^-=\{A_1, \cdots, A_k\}$. By induction hypothesis,

$A_i\theta \in Sp(\mu_J, \sim)$ for $1 \leqq i \leqq k$,

because $f_{n-1}(A_1\theta) \wedge \cdots \wedge f_{n-1}(A_k\theta)=1$.

So, using the same argument as in case i), we notice that $P \cup \{\leftarrow \sigma\}$ has a B-refutation with the first input clause C and B-valued unifier θ w.r.t. μ_J and $\sim$, where C and θ are as in (1).

I + II gives $f_n(\sigma)=1 \rightarrow \sigma \in Sp(\mu_J, \sim)$.

So, we get the required result. □

§2-3. (Δ, Λ)-absoluteness

In this section, we propose another class O of complete sublattices of Pp such that

$(\forall\ N \in O)$ (the least element $[\mu_N]$ of $Mp(B, F) \cap N$ satisfies the condition

$Sp(\mu_N, \sim) = B\mu_N[F]$).

For this purpose, we need the following notion.

Definition 2-3-1. Let $\Delta = \{W_1, \cdots, W_n\}$ be a finite family of pairwise disjoint subsets of Bp. Let $\Lambda = \{b_1, \cdots, b_n\}$ be a subset of B. Let $f \in B^{Bp}$. Then,

i) f is called "(Δ, Λ)-fixed" iff

$(\forall\ \sigma \in W_i)\ (f(\sigma) = b_i)$ for $1 \leqq i \leqq n$.

ii) A subset $Ep(\Delta, \Lambda) \subseteq Pp$ is called "(Δ, Λ)-absolute"

iff

$Ep(\Delta, \Lambda) = \{[f] \in Pp \mid f \text{ is } (\Delta, \Lambda)\text{-fixed}\}$. ⊣

Note: In the following, without loss of generality, we implicitly assume that, for any $[f] \in Ep(\Delta, \Lambda)$, a representative f of [f] is (Δ, Λ)-fixed. ⊣

Proposition 2-3-2. For any (Δ, Λ), (Δ, Λ)-absolute subset $Ep(\Delta, \Lambda)$ of Pp is a complete sublattice of Pp.

Proof: Easy □

Note: The top element $[T_{\Delta, \Lambda}]$ and the bottom element $[\perp_{\Delta, \Lambda}]$ of $(Ep(\Delta, \Lambda), \leqq_F)$ are such that, for any $\sigma \in Bp$,

$$T_{\Delta, \Lambda}(\sigma) = \begin{cases} b_i & , \quad \text{if} \quad \sigma \in W_i \\ 1 & , \quad \text{otherwise} \end{cases}$$

and

$$\perp_{\Delta,\Lambda}(\sigma)=\begin{cases} \mathbf{bi} \ , & \text{if} \quad \sigma\in W_i \\ \\ \mathbf{0} \ , & \text{otherwise .} \end{cases} \qquad \dashv$$

Definition 2-3-3. Let $\sim$ be a substitution transitive relation over $A(\Pi_p, V)$ which is used at B-valued unification. Then, Δ satisfies B-valued unification condition w.r.t $\sim$ iff

$(\forall \sigma,\tau \in W_i)(\sigma\sim\tau)$ for $1\leqq i \leqq n$,

where $\Delta=\{W_1, \cdots, W_n\}$. $\dashv$

Throughout this section, we assume that $\sim$ is arbitrary but fixed and we consider only those $\{Ep(\Delta, \Lambda)|\ \Delta$ satisfies B-valued unification condition$\}$. In this sense, each $Ep(\Delta, \Lambda)$ depends on $\sim$.

Definition 2-3-4. Let $Ep(\Delta, \Lambda)$ be as above.

Define $Tp(\Delta, \Lambda) : Ep(\Delta, \Lambda)\rightarrow Ep(\Delta, \Lambda)$ so that, for any $[f] \in Ep(\Delta, \Lambda)$, $Tp(\Delta, \Lambda)([f])=[G]$, where

$$G(\sigma)=\begin{cases} \mathbf{bi} \ , & \text{if} \quad \sigma\in W_i \ \text{ for } \ 1\leqq i\leqq n . \\ \mathbf{1} \ , & \text{if} \quad \sigma \notin \bigcup_{1\leq i\leq n} W_i \text{ and there is an unit clause in } C(\sigma) \\ \bigvee_{C(\in C(\sigma))} f(C_i^{-}\theta_i) \ , & \text{if } \sigma\notin \bigcup_{1\leq i\leq n} W_i \text{ and } C(\sigma)\neq\emptyset \text{ and there is no unit clause in } C(\sigma) \\ \mathbf{0} \ , & \text{otherwise,} \end{cases}$$

and

$$C(\sigma)=\{C\in P \mid (\exists\theta\text{: ground substitution})(C^{+}\theta=\sigma)\}. \qquad \dashv$$

So far, we have not restricted the possible values in $\Lambda=\{b_1, \cdots, b_n\}$. However, as the next simple example typically shows, Λ must satisfy a certain

kind of conditions w.r.t. F in order that the target program $(P, [[\ ]])$ becomes a F-program.

Example 2-3-5. Let

$P=\{\Psi(a)\leftarrow\}$ and $\Delta=\{W_1\}$ where $W_1=\{\Psi(a)\}$

and $\Lambda=\{b_1\}$. Let $B=\{1, 0\}$ and $F=\{1\}$.

Then, for any $[[\ ]]\in Ep(\Delta, \Lambda)$, in order that $(P, [[\ ]])$ becomes a F-program, b_1 should be 1. ⊣

With the above observation in mind, let's say

Definition 2-3-6. For any P, $Ep(\Delta, \Lambda)$ satisfies F-program property

iff $\quad Ep(\Delta, \Lambda)\cap Mp(B, F)\neq\emptyset$. ⊣

What kind of $Ep(\Delta, \Lambda)$ satisfies F-program property? The simplest case is, of course, that $\Lambda\subset F$. A little more interesting case is;

Example 2-3-7. Let $Hp=\{\sigma\in Bp|(\exists C\in P)(\exists\theta$: ground substitution$)(\sigma=C^+\theta)\}$. (The "headquarter" of P.) Let $\Delta=\{W_1, \cdots, W_n\}$ be such that, for some i $(1\leqq i\leqq n)$, $\{W_1, \cdots, W_i\}\subseteq p(Hp)$ and $\{W_{i+1}, \cdots, W_n\}\subseteq p(Bp-Hp)$, where $p(X)$ is the power set of X. Then, if $\{b_1, \cdots, b_l\}\subseteq F$, the corresponding $Ep(\Delta, \Lambda)$ satisfies F-program property. ⊣

A little more complex case is,

Example 2-3-8. Let $\Delta=\{W_1, \cdots, W_n\}$ be such that, for any i s.t. $W_i\cap(Bp-Hp)\neq\emptyset$ and for any $C\in P$ s.t. C^- witnesses an element of $W_i\cap(Bp-Hp)$,

either $[C^+]\subset(Bp-\bigcup_{1\leqq i\leqq n}W_i)$

or $[C^+]\subset W_{j_1}\cup\cdots\cup W_{j_k}$ s.t. $\quad b_{j_l}\geqq b_i$ for $1\leqq l\leqq k$,

where $[C^+]=\{\sigma\in Bp|(\exists\theta$: ground substitution$)(\sigma=C^+\theta)\}$.

Then, $Ep(\Delta, \Lambda)$ satisfies F-program property. ⊣

To present more complex cases, we need a systematic study of the structure of P. However, the following stronger notion is the one that we want in order to continue the argument.

Definition 2-3-9. Let $Ep(\Delta, \Lambda)$ be arbitrary. Then, $Ep(\Delta, \Lambda)$ satisfies $Tp(\Delta, \Lambda)$-consistent property iff, for any $[f] \in Ep(\Delta, \Lambda)$,

$X[f] = \{ [g] \in Ep(\Delta, \Lambda) \mid (\forall C \in P)(\forall \theta$: ground substitution for C)

$((g(C^+\theta) \leftarrow f(C^-\theta)) \in F \} \neq \emptyset$. ⊣

The relation between these two notions is that

$Ep(\Delta, \Lambda)$ satisfies $Tp(\Delta, \Lambda)$-consistent property $\overset{\Rightarrow}{\Leftarrow}$ $Ep(\Delta, \Lambda)$ satisfies F-program property.

The direction $\Leftarrow$ is easy to check. For example,

Example 2-3-10. Let $P = \{\psi(p) \leftarrow \psi(q)\}$ and $\Delta = \{ \{\Psi(p)\} \}$ and $\Lambda = \{0\}$.

Then, by taking $[[\ ]] \in B^{Bp}$ s.t. $[[\psi(q)]] = \mathbf{0}$, we notice that

$[[[\ ]]] \in Ep(\Delta, \Lambda) \cap Mp(B, F)$. So, $Ep(\Delta, \Lambda)$ satisfies F-program property.

However, for a $[f] \in Ep(\Delta, \Lambda)$ s.t. $f(\psi(q)) = 1$,

$X[f] = \emptyset$ because, for any $[g] \in Ep(\Delta, \Lambda)$,

$$(g(\psi(p)) \leftarrow f(\psi(q))) = 0 \leftarrow 1 = 0 \notin F.$$

So, $Ep(\Delta, \Lambda)$ does not satisfy $Tp(\Delta, \Lambda)$-consistent property. ⊣

The converse direction "$\Rightarrow$" will be shown soon.

By the way, the naming of "$Tp(\Delta, \Lambda)$-consistency" comes from the following observation.

Lemma 2-3-11. Let $Tp(\Delta, \Lambda)$: $Ep(\Delta, \Lambda) \to Ep(\Delta, \Lambda)$ be as above. Suppose $Ep(\Delta, \Lambda)$ satisfies $Tp(\Delta, \Lambda)$-consistent property.

Then, for any $[f] \in Ep(\Delta, \Lambda)$, $Tp(\Delta, \Lambda)([f]) = \wedge X[f]$.

Proof: Similar to the proof of Lemma 1-4-5 in chapter I except the case of $\sigma \in \bigcup_{1 \leq i \leq n} W_i$. □

In the following, we exclusively consider the case that $Ep(\Delta, \Lambda)$ satisfies $Tp(\Delta, \Lambda)$-consistent property. Under which condition does $Ep(\Delta, \Lambda)$ satisfy $Tp(\Delta, \Lambda)$-consistent property? By the above relation, we notice that more systematic study is needed to investigate the condition than the case of *F*-program property. However, it is also easy to see that the condition stated in Example 2-3-7 is enough to preserve $Tp(\Delta, \Lambda)$-consistent property. Now,

Definition 2-3-12. Let $Tp(\Delta, \Lambda)$: $Ep(\Delta, \Lambda) \rightarrow Ep(\Delta, \Lambda)$ be as above.

Define

$Tp(\Delta, \Lambda) \uparrow 0 = [\bot_{\Delta, \Lambda}]$

$Tp(\Delta, \Lambda) \uparrow n+1 = Tp(\Delta, \Lambda)(Tp(\Delta, \Lambda) \uparrow n)$

$Tp(\Delta, \Lambda) \uparrow \omega = \bigvee_{n<\omega} Tp(\Delta, \Lambda) \uparrow n$. ⊣

Lemma 2-3-13. $Tp(\Delta, \Lambda)$ is monotonic and continuous.

Proof: Similar to the proof of Lemma 1-4-9 in chapter I. □

As a direct consequence, we obtain, as usual,

Corollary 2-3-14. $Tp(\Delta, \Lambda) \uparrow \omega$ is the least fixed point of $Tp(\Delta, \Lambda)$.

Proof: Similar to the proof of Corollary 1-4-13 in chapter I. □

Proposition 2-3-15. Let $Tp(\Delta, \Lambda)$: $Ep(\Delta, \Lambda) \rightarrow Ep(\Delta, \Lambda)$ be as above. Suppose $[f] \in Ep(\Delta, \Lambda)$. Then,

$[f] \in Mp(B, F)$ iff $Tp(\Delta, \Lambda)([f]) \leqslant_F [f]$.

Proof: Similar to the proof of Proposition 1-4-7 in chapter 1. □

From this proposition, we at once notice that

$Mp(B, F)\cap Ep(\Delta, \Lambda) \ni Tp(\Delta, \Lambda) \uparrow \omega$. Thus we have shown

$Tp(\Delta, \Lambda)$-consistent property $\Rightarrow$ F-program property.

Moreover, we obtain

Corollary 2-3-16. Let $Tp(\Delta, \Lambda)$ be as above and $[\mu_{\Delta, \Lambda}]$ be the least element of $Mp(B, F)\cap Ep(\Delta, \Lambda)$. Then,

$[\mu_{\Delta, \Lambda}]$ = the least fixedpoint of $Tp(\Delta, \Lambda)$.

Proof: Similar to the proof of Theorem 1-4-8 in chapter 1. ⊓

Before stating relativized logical completeness w.r.t. $Ep(\Delta, \Lambda)$, we need two more properties.

Definition 2-3-17. Λ satisfies F-consistent property iff, for any disjunctive B-value $\mathbf{d} = \mathbf{d}_1 \vee \cdots \vee \mathbf{d}_k$ where each $\mathbf{d}_i$ $(1 \leqq i \leqq k)$ has a conjunctive normal form generated from $\Lambda \cup \{0, 1\}$

(, that is $\mathbf{d}_i = \mathbf{b}_{i_1} \wedge \cdots \wedge \mathbf{b}_{i_j}$ s.t. $\mathbf{b}_{i_l} = \mathbf{b}_{i_{l_1}} \vee \cdots \vee \mathbf{b}_{i_{l_m}}$ and $\{\mathbf{b}_{i_{l_1}}, \cdots, \mathbf{b}_{i_{l_m}}\} \subset \Lambda \cup \{0,1\}$ for $1 \leqq l \leqq j$),

if $\mathbf{d} \in F$, then $(1 \leqq \exists i \leqq k)\ (\mathbf{d}_i \in F)$. ⊣

We have already obtained the property of $Tp(\Delta, \Lambda)$-consistent property of $Ep(\Delta, \Lambda)$. These two properties are irrelevant in the following sense.

$Ep(\Delta, \Lambda)$ satisfies $Tp(\Delta, \Lambda)$-consistent property

$\nRightarrow$

$\nLeftarrow$

Λ satisfies F-consistent property.

The check is easy. For example,

Example 2-3-18.

1. Let $P=\begin{cases}\psi(p)\leftarrow\psi(q)\\ \psi(q)\leftarrow\end{cases}$, $\Delta=\{\{\psi(p)\},\{\psi(q)\}\}$, $\Lambda=\{0,1\}$

s.t. $\psi(p)\rightarrow 1$, $\psi(q)\rightarrow 0$.

Then, $Ep(\Delta,\Lambda)$ is not $Tp(\Delta,\Lambda)$-consistent for any F over any **B**. However, Λ satisfies F-consistent property.

2. Let $P=\begin{cases}\psi(p)\leftarrow\psi(q)\\ \psi(p)\leftarrow\psi(r)\end{cases}$, $\Delta=\{\{\psi(q)\},\{\psi(r)\}\}$, $\Lambda=\{\mathbf{a},\mathbf{b}\}$

s.t. $\psi(q)\rightarrow \mathbf{a}$, $\psi(r)\rightarrow \mathbf{b}$, $\mathbf{a}\vee\mathbf{b}\in F$, $\mathbf{a}\notin F$, $\mathbf{b}\notin F$.

(For example, take $B=\{1,\ \mathbf{b},\ \neg\mathbf{b},\ 0\}$, $F=\{1\}$, $\mathbf{a}=\neg\mathbf{b}$.) Then, for any $[f]\in Ep(\Delta,\Lambda)$, by taking $[g]\in Ep(\Delta,\Lambda)$ such that $g(\psi(p))=1$, we notice that $X[f]\neq\emptyset$. So, $Ep(\Delta,\Lambda)$ satisfies $Tp(\Delta,\Lambda)$-consistent property. However, by the choice of **a** and **b**, Λ does not satisfy F-consistent property. ⊣

The second property is;

Definition 2-3-19. $Ep(\Delta,\Lambda)$ claims the existence of eliminators iff, for any $W_i\in\Delta$ such that $\mathbf{b}_i\in F$, there is an atom $\varepsilon_i\in W_i$ such that P contains the assertion $\varepsilon_i\leftarrow$. Each ε_i is called "an eliminator"of W_i. ⊣

By definition, we permit the case that there might be two different eliminators ε_i, ε_i' for the same W_i. That

existence of eliminators $\not\Leftrightarrow$ $Tp(\Delta,\Lambda)$-consistency of $Ep(\Delta,\Lambda)$

and

existence of eliminators $\not\Leftrightarrow$ F-consistency of Λ

is too obvious to check.

Now, we are ready to prove the main theorem in this section.

Theorem 2-3-20. Let $\sim$ be a substitution transitive relation over A(Πp, V). Let B be a complete Boolean algebra and F be a complete filter over B. Let Ep(Δ, Λ) be a complete sublattice of Pp such that

1. Ep(Δ, Λ) satisfies Tp(Δ, Λ)-consistent property and claims the existence of eliminators
2. Δ satisfies B-valued unification condition w.r.t.$\sim$
3. Λ satisfies F-consistent property.

Let $[\mu_{\Delta,\Lambda}]$ be the least element of $\mathrm{Ep}(\Delta, \Lambda) \cap \mathrm{Mp}(B, F)$.

Then,

$\mathrm{Sp}(\mu_{\Delta,\Lambda}, \sim) = B_{\mu_{\Delta,\Lambda}}[F]$, where $\mu_{\Delta,\Lambda}$ is a (Δ, Λ)-fixed representative of $[\mu_{\Delta,\Lambda}]$.

Proof: By soundness theorem

$\mathrm{Sp}(\mu_{\Delta,\Lambda}, \sim) \subseteq B\mu_{\Delta,\Lambda}[F]$ is obvious.

So, we want to show $B\mu_{\Delta,\Lambda}[F] \subseteq \mathrm{Sp}(\mu_{\Delta,\Lambda}, \sim)$.

Let $\sigma \in B\mu_{\Delta,\Lambda}[F]$ be arbitrary. We already know that

$[\mu_{\Delta,\Lambda}] = \mathrm{Tp}(\Delta, \Lambda) \uparrow \omega$.

$\therefore$ $(\exists n < \omega)\ (f_n(\sigma) \in F)$, where $[f_n] = \mathrm{Tp}(\Delta, \Lambda) \uparrow n$.

In the following, we prove by induction on n that

$f_n(\sigma) \in F \Rightarrow \sigma \in \mathrm{Sp}(\mu_{\Delta,\Lambda}, \sim)$.

I) The case of $n = 0$.

By definition of $[f_0] = [\perp_{\Delta,\Lambda}]$, since $f_0(\sigma) \in F$,

$\sigma \in W_i$ for some i such that $b_i \in F$.

So, using an eliminator $\varepsilon_i \leftarrow$, $P \cup \{\leftarrow o\}$ has a B-refutation of length 1 w.r.t. $\mu_{\Delta,\Lambda}$, because

$$\mu_{\Delta,\Lambda}(\sigma) = \mu_{\Delta,\Lambda}(\varepsilon_i) = b_i \text{ and } \sigma \sim \varepsilon_i.$$

II) General case.

Suppose the result holds for $n-1$. Let $f_n(\sigma) \in F$.

i) The case that $\sigma \in W_i$ for some $1 \leqq i \leqq n$.

Then, using the same argument as in I), we notice that P∪{←σ} has a B-refutation of length 1 w.r.t. $\mu_{\Delta,\Lambda}$.

ii) The case that $\sigma \notin \bigcup_{1 \leq i \leq n} W_i$ and C(σ) has an unit clause.

Let A← ∈C(σ) be such that

(∃θ: ground substitution) (Aθ = σ).

Then, it is obvious that P∪{←σ} has a B-refutation (w.r.t. $\mu_{\Delta,\Lambda}$) with the input clause A← and the B-valued unifier θ .

iii) The other cases.

Since $f_n(\sigma) \in F$, we notice that

$$f_n(\sigma) = \bigvee_{C_i \in C(\sigma)} f_{n-1}\ (C_i^- \theta_i).$$

Here, we know that, for each $C_i \in C(\sigma)$, the B-value $f_{n-1}(C_i^- \theta_i)$ has the form

$$f_{n-1}(A_1\theta_i) \wedge \cdots \wedge f_{n-1}\ (A_k\theta_i)$$

where $C_i^- = A_1, \cdots, A_k$.

Now, by the construction of $Tp(\Delta, \Lambda) \uparrow n-1$, we notice that each $f_{n-1}\ (A_j\theta_i)$ $(1 \leqq j \leqq k)$ can be expressed as a conjunctive normal form

$$d_{j1} \wedge \cdots \wedge d_{jl}$$

generated from (a subset of) $\Lambda \cup \{0, 1\}$, by using Boolean algebraic transformation. Here, since Λ satisfies *F*-consistent property,

$(\exists C_i \in C(\sigma))\ (f_{n-1}\ (C_i\ \theta_i) \in F)$ ··· ①

$\Rightarrow$ $f_{n-1}(A_i\theta_i) \wedge \cdots \wedge f_{n-1}(A_k\theta_i) \in F$

$\Rightarrow$ $f_{n-1}(A_j\theta_i) \in F$ for $1 \leqq j \leqq k$.

So, by the induction hypothesis,

$A_j\theta_i \in Sp(\mu_{\Delta,\Lambda}, \sim)$ for $1 \leqq j \leqq k$.

∴ P∪{←σ} has a B-refutation based on $\mu_{\Delta,\Lambda}$ with the first input clause C_i in ① and the first B-valued unifier θ_i .

□

§2-4. More about Logical Completeness

In §2-2, we propose a subclass $\Gamma p(J)$ of $\mathbf{Pp}$ such that the least element $[\mu_J]$ of $\Gamma p(J) \cap Mp(B, F)$ satisfies

$$B\mu_J(F) = Sp(\mu_J, \sim)$$

and in §2-3, we show that another subclass $Ep(\Delta, \Lambda)$ of $\mathbf{Pp}$ also satisfies

$$B\mu_{\Delta,\Lambda}(F) = Sp(\mu_{\Delta,\Lambda}, \sim)$$

where $[\mu_{\Delta,\Lambda}]$ is the least element of $Ep(\Delta, \Lambda) \cap Mp(B, F)$. Here, consider the least element $[\mu_{\Delta,\Lambda,J}]$ of $\Gamma p(J) \cap Ep(\Delta, \Lambda) \cap Mp(B, F)$. Does $[\mu_{\Lambda,\Lambda,J}]$ satisfy the condition $B\mu_{\Delta,\Lambda,J}(F) = Sp(\mu_{\Delta,\Lambda,J}, \sim)$?

The answer is negative for general pair $(\Gamma p(J), Ep(\Delta, \Lambda))$. However, with slight conditions over (Δ, Λ, J), we obtain the expected result.

Definition 2-4-1 Let's $Ep(\Delta, \Lambda)$ be as in §2-3 and $J : Bp \to Bp$ be an idempotent function such that

1. $(\forall \sigma \in Bp)(J(\sigma) \sim \sigma)$ (J satisfies B-valued unification condition)
2. $(\forall W_i \in \Delta)(\exists \upsilon_i \in W_i)(\forall \tau \in W_i)(J(\tau) = \sigma_i)$
3. $(\forall \sigma \in (Bp - \bigcup_{1 \le i \le n} W_i))(J(\sigma) \notin \bigcup_{1 \le i \le n} W_i)$.

Then, J is called to be "consistent with Δ." ⊣

Thus, the effect of J is divided into disjoint two parts $J \lceil \bigcup_{1 \le i \le n} W_i$ and $J \lceil (Bp - \bigcup_{1 \le i \le n} W_i)$. For $J \lceil \bigcup_{1 \le i \le n} W_i$, J works as a choice function of a representative σ_i for each W_i. Here, consider $Ep(\Delta, \Lambda) \cap \Gamma p(J)$, where J is consistent with Δ. Obviously, $Ep(\Delta, \Lambda) \cap \Gamma p(J)$ becomes a complete sublattice of $\mathbf{Pp}$. For $Ep(\Delta, \Lambda) \cap \Gamma p(J)$, we employ the similar argument to the case of $Ep(\Delta, \Lambda)$. The main differences are the following.

Instead of $Tp(\Delta, \Lambda)$, we use

$$Tp(\Delta, \Lambda, J) : (Ep(\Delta, \Lambda) \cap \Gamma p(J)) \to (Ep(\Delta, \Lambda) \cap \Gamma p(J))$$

such that, for any $[f] \in Ep(\Delta, \Lambda) \cap \Gamma p(J)$,

$Tp(\Delta, \Lambda, J)([f]) = [G]$ such that, for any $\sigma \in Bp$,

$$G(\sigma) = \begin{cases} b_i \ , & \text{if } \sigma \in W_i \quad \text{for } 1 \leqq i \leqq n \\ 1 \ , & \text{if } \sigma \notin \bigcup_{1 \leq i \leq n} W_i \text{ and there is an unit clause in } D(J(\sigma)) \\ \bigvee_{C \in D(J(\theta))} f(C_i \theta_i), & \text{if } \sigma \notin \bigcup_{1 \leq i \leq n} W_i \text{ and } D(J(\sigma)) \neq \emptyset \text{ and there is no unit clause in } D(J(\sigma)) \\ 0 \ , & \text{otherwise,} \end{cases}$$

where $D(J(\sigma)) = \{C \in P \mid (\exists\theta\text{: ground})\,(J(C^+\theta) = J(\sigma) \text{ and } C^+\theta \sim J(\sigma)\}$.

Here, we can define the similar notation to $Tp(\Delta, \Lambda)$-consistent property defined in the previous section.

Definition 2-4-2. $Ep(\Delta, \Lambda) \cap \Gamma p(J)$ satisfies $Tp(\Delta, \Lambda, J)$-consistent property iff, for any $[f] \in Ep(\Delta, \Lambda) \cap \Gamma p(J)$,

$\{[g] \in Ep(\Delta, \Lambda) \cap \Gamma p(J) \mid (\forall C \in P)\,(\forall\theta$: ground substitution for $C)$

$((g(C^+\theta) \leftarrow f(C^-\theta)) \in F)\} \neq \emptyset$ ⊣

In this situation, we obtain;

Theorem 2-4-3. Let $\sim$ be a substitution transitive equivalence relation over $A(\Pi p, V)$. Let B be a complete Boolean algebra and F be a complete filter over B. Consider

$Ep(\Delta, \Lambda) \cap \Gamma p(J)$ such that

1. Δ satisfies B-valued unification condition w.r.t. $\sim$.
2. J is consistent with Δ.
3. $Ep(\Delta, \Lambda) \cap \Gamma p(J)$ satisfies $Tp(\Delta, \Lambda, J)$-consistent property.
4. Λ satisfies F-consistent property
5. for any $1 \leq i < n$ such that $b_i \in F$, the representative σ_i of W_i by J becomes the eliminator of W_i.

Then, $Ep(\Delta, \Lambda) \cap \Gamma p(J) \cap Mp(B, F) \neq \emptyset$.

Moreover, let $[\mu_{\Delta, \Lambda, J}]$ be the least element of

$Ep(\Delta, \Lambda) \cap \Gamma p(J) \cap Mp(\mathbf{B}, F)$, then

$Sp(\mu_{\Delta, \Lambda, J}, \sim) = B_{\mu_{\Delta, \Lambda, J}}[F]$, where $\mu_{\Delta, \Lambda, J}$ is a J-faithful and (Δ, Λ)-fixed representative of $[\mu_{\Delta, \Lambda, J}]$.

Proof: Combination of Theorem 2-2-12 and Theorem 2-3-20. □

Chapter III Some Remarks with Examples

$$\text{Let } P=\begin{cases}\psi(p)\leftarrow\\ \psi(q)\leftarrow\psi(r)\end{cases}$$

In the following, for simplicity, we write p, q, r instead of $\psi(p)$, $\psi(q)$, $\psi(r)$ respectively. Let B be an arbitrary complete Boolean algebra whose cardinality is larger than (or at least equal to) 2^4 and F be a complete filter over B such that $\{1, a, b\}\subseteq F$. $(1\neq a\neq b)$

Let $c\in B$ be such that $c\notin F\cup I$, where I is the dual ideal of F. $Bp=\{p, q, r\}$ and the success set Sp of P is $\{p\}$. So, the least Herbrand model is $\{p\}$. Let $\top$, $\bot$, μ, f, g, h, i, j be maps from Bp to B such that

$\top(p)=\top(q)=\top(r)=1$

$\bot(p)=\bot(q)=\bot(r)=0$

$\mu(p)=1, \mu(q)=\mu(r)=0$

$f(p)=f(q)=1, f(r)=0$

$g(p)=1, g(q)=0, g(r)=1$

$h(p)=1, h(q)=c, h(r)=0$

$i(p)=1, i(q)=c, i(r)=c$

$j(p)=j(q)=1, j(r)=c$.

By definition, we at once notice that

$$\{[\top], [\bot], [\mu], [f], [g], [h], [i], [j]\}\subset Pp \qquad \cdots(a)$$

and no two equivalence classes are the same.

Let $\top'\in B^{Bp}$ be such that

$\top'(p)=1, \top'(q)=a, \top'(r)=b.$

Then, by definition, $\top'\in[\top]$.

Similarly, there are many elements in each equivalence class $[[\ \ ]]$ of Pp.

For example, if $\bot'$, $\mu'\in B^{Bp}$ be such that $\bot'(p)=0$, $\bot'(q)=\neg a$, $\bot'(r)=\neg b$, and $\mu'(p)=a\wedge b$, $\mu'(q)=\neg a$, $\mu'(r)=0$, then $\bot'\in[\bot]$ and $\mu'\in[\mu]$, etc.

The importance is that

$(\forall [[\ ]], [[\ ]]' \in [\,[[\ ]]\,])(\,B_{[[\ ]]}[F] = B_{[[\ ]]'}[F])$.

In this sense, $B_{[[\ ]]}[F]$ is uniquely determined for each $[\,[[\ ]]\,]$.

Here, the first fact we had better remark is

Remark 1.

There may be a class $[\,[[\ ]]\,]$ in Pp such that no representative of $[\,[[\ ]]\,]$ can be reduced to "two-valued." ⊣

In this example, [h] plays this role. In this sense, B-valued is essentially different from 2-valued philosophy. That how [h] works will become clear soon. Next, for some people who are not familiar with logic, let's point the fact that

Remark 2.

Though $[\mu] <_F [g]$ and (Up, μ) is a Herbrand F-model, (Up, g) is not a Herbrand F-model. This is simply because $g(q \leftarrow r) = 0 \leftarrow 1 = 0 \notin F$. On the other hand, (Up, h) is surely a Herbrand F-model as you can see easily. ⊣

So, as candidates belonging to $Mp(B, F)$, we can pick up

$$\{[T], [\mu], [f], [h], [i], [j]\} \subset Mp(B, F) \qquad \cdots(\beta)$$

from (α).

Among the list (β), there is the least element of $Mp(B, F)$. [μ] is the one. The crucial fact concerning the least element in general is;

Remark 3.

For any P, B, F, ∼, the least element of $Mp(B, F)$ has a representative who takes {0, 1} as the range. ⊣

In this example, μ as the representative of [μ] works this task. This fact is interesting from the viewpoint of Remark 1 and is already proved in chapter I. More generally,

Remark 4.

For any P, B, F, ∼, the least element $[\mu_J]$ of $\Gamma p(J) \cap Mp(B, F)$ has a J-faithful representative who takes {0, 1} as the range, where ∼ should be an equivalence relation and $\Gamma p(J)$ is as in chapter II, §2-2. ⊣

This is the fact which is obtained as a byproduct of the main results stated in §2-2 of chapter II. The crucial point is the least element $[\mu_J]$ of $\Gamma p(J) \cap Mp(B, F)$ may no more be the least element of original $Mp(B, F)$. ($[\mu_J]$ depends on $\sim$, J. Examples will be discussed soon.)

However, on the other hand,

Remark 5.

There exist P, B, F, $\sim$, Δ, Λ such that the least element $[\mu_{\Delta, \Lambda}]$ of $Ep(\Delta, \Lambda) \cap Mp(B, F)$ has no representative who takes $\{0, 1\}$ as the range, where $Ep(\Delta, \Lambda)$ is as in chapter II, §2-3 satisfying the conditions stated in Theorem 2-3-20.

More generally, there exist P, B, F, $\sim$, J, Δ, Λ such that the least element $[\mu_{\Delta, \Lambda, J}]$ of $\Gamma p(J) \cap Ep(\Delta, \Lambda) \cap Mp(B, F)$ has no representative who takes $\{0, 1\}$ as the range, where $\Gamma p(J) \cap Ep(\Delta, \Lambda)$ satisfies the conditions stated in Theorem 2-4-3. ⊣

So far, we have spotted our attention on the declarative semantics of B-valued model which are classified by F. From now on, let's begin to focus on procedural semantics of B-valued unification. The fundamental fact we want to point is that

Remak 6.

$$Sp([[\ ]], \sim) \subset B_{[[\ ]]}[F] \qquad \cdots (¥)$$

always holds if only (Up, [[ ]]) is a Herbrand F-model. (Soundness of B-valued unification. See chapter I, §1-3.) However, the soundness does not always hold if (Up, [[ ]]) is not a F-model. This is the reason we should start from a B-valued F-model instead of an arbitrary B-valued interpretation of P in order to state something meaningfully. ⊣

Here, since B-valued unification directly depends on both $\sim$ and [[ ]] by definition, it is usual that

$$(\exists [[\ ]], [[\ ]]' \in [[[\ ]]])(Sp([[\ ]], \sim) \neq Sp([[\ ]]', \sim)) \qquad \cdots (A)$$

for many $[\,[\![\ ]\!]\,] \in \mathrm{Mp}(\mathrm{B}, F)$. In this sense, the classification modulo F can't categorize the procedural world except the fact (Υ), which asserts that $\mathrm{B}_{[\![\ ]\!]}[F]$ becomes an upper bound of $\{\mathrm{Sp}([\![\ ]\!], \sim) \mid [\![\ ]\!] \in [\,[\![\ ]\!]\,]\}$, i.e.,

$$\mathrm{B}_{[\![\ ]\!]}[F] \supset \bigcup \{\mathrm{Sp}([\![\ ]\!], \sim) \mid [\![\ ]\!] \in [\,[\![\ ]\!]\,]\}.$$

Nevertheless, there are some elements $[\,[\![\ ]\!]\,], \cdots$ in $\mathrm{Mp}(\mathrm{B}, F)$ which satisfy the relation

$$(\forall [\![\ ]\!], [\![\ ]\!]' \in [\,[\![\ ]\!]\,])(\mathrm{Sp}([\![\ ]\!], \sim) = \mathrm{Sp}([\![\ ]\!]', \sim)). \qquad \cdots (\mathrm{B})$$

As candidates for such $[\,[\![\ ]\!]\,]$ satisfying the relation (B), we have shown that

Remark 7.

For any P, B, F, the least element $[\mu]$ of $\mathrm{Mp}(\mathrm{B}, F)$ satisfies the relation (B). More strongly, $[\mu]$ satisfies

$$(\forall [\![\ ]\!] \in [\mu])(\mathrm{Sp}([\![\ ]\!], \sim) = \mathrm{B}_{\mu}[F]) \qquad \cdots (\mathrm{C})$$

(See chapter I, §1-5)

More generally, for any P, B, F, the least element $[\mu_J]$ of $\Gamma\mathrm{p}(J) \cap \mathrm{Mp}(\mathrm{B}, F)$ satisfies the relation

$$(\forall [\![\ ]\!], [\![\ ]\!]' \in [\mu_J])([\![\ ]\!] \text{ and } [\![\ ]\!]' \text{ are J-faithful} \rightarrow \mathrm{Sp}([\![\ ]\!], \sim) = \mathrm{Sp}([\![\ ]\!]', \sim)) \qquad \cdots (\mathrm{B}')$$

where $\sim$ is an equivalence relation and $\Gamma\mathrm{p}(J)$ is as in chapter II, §2-2.

More strongly, $[\mu_J]$ satisfies

$$(\forall [\![\ ]\!] \in [\mu_J])([\![\ ]\!] \text{ is J-faithful} \rightarrow \mathrm{Sp}([\![\ ]\!], \sim) = \mathrm{B}\mu_J[F] \qquad \cdots (\mathrm{C}')$$

(See chapter II, §2-2)

Similar relations also hold for the least element $[\mu_{\Delta,\Lambda}]$ of $\mathrm{Ep}(\Delta, \Lambda) \cap \mathrm{Mp}(\mathrm{B}, F)$ and the least element $[\mu_{\Delta,\Lambda,J}]$ of $\mathrm{Ep}(\Delta, \Lambda) \cap \Gamma\mathrm{p}(J) \cap \mathrm{Mp}(\mathrm{B}, F)$ where $\mathrm{Ep}(\Delta, \Lambda)$ and $\mathrm{Ep}(\Delta, \Lambda) \cap \Gamma\mathrm{p}(J)$ satisfy suitable conditions stated in chapter II, §2-3 and §2-4 ⊣

That the relations (B) and (C) are not equivalent is easy to see. For example, consider a program $\mathrm{P} = \{\psi(\mathrm{a}) \leftarrow \psi(\mathrm{b})\}$. Let $[\![\ ]\!] : \mathrm{Bp} \rightarrow \mathrm{B}$ be such that $[\![\psi(\mathrm{a})]\!] = [\![\psi(\mathrm{b})]\!] = 1$. Let F be a complete filter over B which has at least two

elements. Then, (Up, [[ ]]) becomes a Herbrand F-model of P and $B_{[[\ ]]}=\{\psi(a), \psi(b)\}$. However, for any relation $\sim$, $(\forall[[\ ]]\in[\,[[\ ]]\,])(\mathrm{Sp}([[\ ]], \sim)=\emptyset)$. So, for this kind of [[ ]], (B) trivially holds but (C) does not hold. Note that, in general, the relation

$$\cup\{\mathrm{Sp}([[\ ]], \sim)\mid [[\ ]]\in[\,[[\ ]]\,]\}=B_{[[\ ]]}[F] \qquad \cdots(D)$$

and (C) is not equivalent (a witness will be given soon), though (C)⇒(D) is obvious. Furthermore, the above example also tells us that (B) ⇏ (D). That (D)⇏ (B) will soon be explained at the following I, ii).

Here, we should notice the fact that the least elements $[\mu_J]$, $[\mu_{\Delta,\Lambda}]$, $[\mu_{\Delta,\Lambda,J}]$ in the above Remark 7 satisfy the condition (D), not the original (C) itself. To be more precise, they satisfy

$$(\exists[[\ ]]\in[\,[[\ ]]\,])(\mathrm{Sp}([[\ ]], \sim)=B_{[[\ ]]}(F)) \qquad \cdots(E)$$

where a witness of $[\mu_J]$ ($[\mu_{\Delta,\Lambda}]$, $[\mu_{\Delta,\Lambda,J}]$) in the sense of (E) is a J-faithful ((Δ, Λ)-fixed, J-faithful and (Δ, Λ)-fixed respectively) element of $[\mu_J]$ ($[\mu_{\Delta,\Lambda}]$, $[\mu_{\Delta,\Lambda,J}]$). Obviously,

(E)→(D) holds. An interesting fact is that

Proposition 8. (D)→(E) hols.

Proof: Suppose (D). It is enough to see that

$$(\exists\,\zeta\in[\,[[\ ]]\,])(\forall[[\ ]]\in[\,[[\ ]]\,])(\mathrm{Sp}([[\ ]], \sim)\subseteq\mathrm{Sp}(\zeta, \sim)), \qquad \cdots(1)$$

because, if so, then we get

$$\cup\{\mathrm{Sp}([[\ ]], \sim)\mid [[\ ]]\in[\,[[\ ]]\,]\}\subseteq\mathrm{Sp}(\zeta, \sim)\subseteq B_\zeta[F].$$

(By (D), this means $\mathrm{Sp}(\zeta, \sim)=B_\zeta[F]$.)

For this purpose, we choose $\zeta\in[\,[[\ ]]\,]$ such that

$(\forall\sigma, \tau\in\mathrm{Bp})(\zeta(\sigma)=\zeta(\tau)$ iff $(\zeta(\sigma)\leftrightarrow\zeta(\tau))\in F)$.

(Consider $\{[[\sigma]]\mid\sigma\in\mathrm{Bp}\}/F$ and let c be a choice function.

Then, for each equivalence class $[\,[[\sigma]]\,]_F$, let's define $\zeta(\sigma)=c([\,[[\sigma]]\,]_F)$. Thus defined ζ obviously satisfies the required condition.)

Then, all we need to see is

$(\forall[[\ ]]\in[\ [[\ \ ]]\ ])\ (\forall\ \sigma\in Bp)$ (P∪{←σ} has a B-refutation based on $[[\ \ ]]$ and $\sim$ → P∪{←σ} has a B-refutation based on ζ and $\sim$).

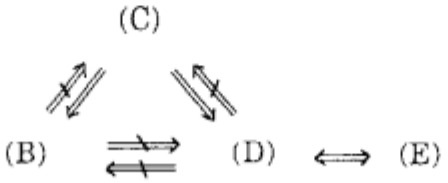
Here, by considering special (ground) refutation defined in chapter I §1-2, we notice that it is enough to check that

$(\forall[[\ ]]\in[\ [[\ \ ]]\ ])\ (\forall\ \sigma,\tau\in Bp)\ (\ [[\sigma]]=[[\tau]]\rightarrow\zeta(\sigma)=\zeta(\tau)\ )$.

Let $[[\ \ ]]\in[\ [[\ \ ]]\ ]$ be arbitrary. Let $\sigma,\ \tau\in Bp$ be such that $[[\sigma]]=[[\tau]](=\mathbf{b}$, say). Then, by combining $(\ [[\sigma]]\leftrightarrow\zeta(\sigma)\ )\in F$ and $(\ [[\tau]]\leftrightarrow\zeta(\tau))\in F$, we notice $(\zeta(\sigma)\leftrightarrow\zeta(\tau)\ )\in F$

$\Rightarrow\ \zeta(\sigma)=\zeta(\tau)$. □

Thus, we obtain a diagram



Now, returning to our concrete example, let's reopen our explanation by checking the above fundamental facts 1~7.

I. For a while, for simplicity, let's assume that p~q~r.

i). For 6, just take $\perp,\perp'\in[\perp]$.

Then, $Sp(\perp,\sim)=\{p,\ q,\ r\}$ and $Sp(\perp',\sim)=\{p\}$. However, $B_{\perp}[F]=B_{\perp'}[F]=\{\sigma\in Bp|\perp(\sigma)\in F\}=\emptyset$. So, $[\perp]$ violates the relation 6. This is simply because $[\perp]\notin Mp(B,F)$.

ii) Next, for (A), just take $\top,\top'\in[\top]$. Then, $Sp(\top,\sim)=\{p,q,r\}\neq Sp(\top',\sim)=\{p\}$.

This $[\top]$ also becomes a witness that (D)⇒(B) and at the same time becomes a witness of inequality of (D) and (C).

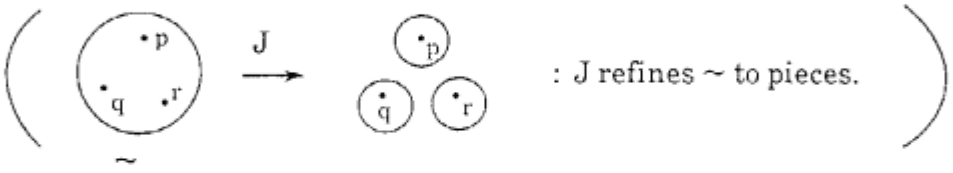
Here, for 6, check the fact that $Sp(\top,\sim)\subset B_{\top}[F]=\{\sigma\in Bp|\top(\sigma)\in F\}=\{p,q,r\}$.

iii). For 7, we already know that $[\mu]$ is the least of $Mp(B,F)$. Take $\mu,\mu'\in[\mu]$. Then, $Sp(\mu,\sim)=\{p\}=Sp(\mu',\sim)$ and $B\mu[F]=\{\sigma\in Bp|\mu(\sigma)\in F\}=\{p\}$. Here, it is

easy to see that taking any other possible element μ" of [μ] will result Sp(μ", ~)={p}. So, we notice that, for any [[ ]] ∈[μ], Sp([[ ]], ~)=$B_{\mu}[F]$.

iv). Let's consider the effects of J.

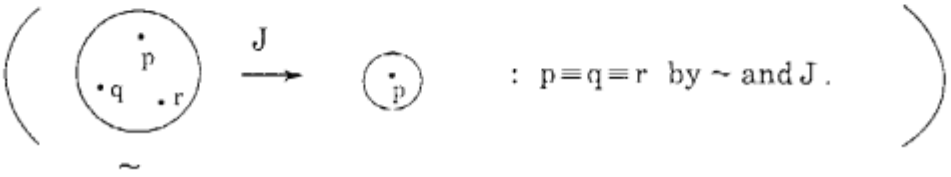
イ)The case that J is defined by J(p)=p, J(q)=q, J(r)=r.



: J refines ~ to pieces.

In this case, by definition, J plays no essential role at the stage of the selection of a subclass of Pp. To be more precise, Pp=Γp(J) and $[\mu_J]=[\mu]$.

ロ)The case that J is defined by J(p)=J(q)=J(r)=p.



: p≡q≡r by ~ and J .

In this case, by definition, {[T], [⊥]}⊂Γp(J) and {[T]}⊂Mp(B, F)∩Γp(J) in the above example. (So, J essentially works.) Here, it is easy to check that [T] becomes the least element of Mp(B, F)∩Γp(J) . Now, as in ii), we obtain

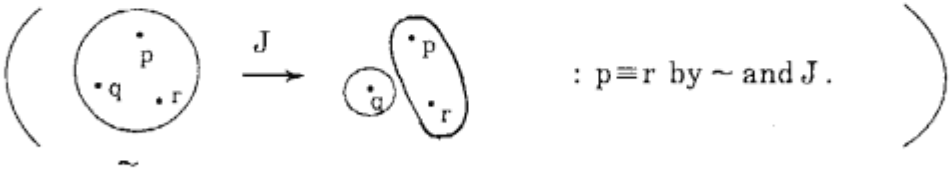
Sp(T, ~)=$B_T[F]$.

However, we can't choose T' as a representative of [T] because T' is not J-faithful. Here, it is easy to see that taking any other possible J-faithful element T" of [T] will result

Sp(T", ~)=$B_{T''}[F]$.

So, we checked 4 and 7 in this simple example.

ハ)The case that J is defined by J(p)=J(r)=p, J(q)=q.



: p≡r by ~ and J .

By ~ and J, equivalence relation over Bp is defined so that $[p]_{\equiv}=\{p, r\}$, $[q]=\{q\}$.

Then, $\{[\top], [g], [\bot]\}\subset\Gamma p(J)$ but $[g]\notin Mp(B, F)$ as is explained earlier in 2.

As above, $[\top]$ becomes the least element of $Mp(B, F)\cap\Gamma p(J)$. Now, consider $\top''\in[\top]$ s.t.

$\top''(p)=\top''(r)=1$, $\top''(q)=a$.

As stated above, $Sp(\top, \sim)=Sp(\top'', \sim)=\{p, q, r\}=B_{\top}[F]$ However, B-valued unification based on $\top$ and B-valued unification based on $\top''$ have different procedural semantics. Just consider a query $\{\leftarrow q\}$. In the usual 2-valued philosophy, this can never happen. In general, from a viewpoint of universal unification, this phenomenon amounts to

<1> $(\sim, J)$ defines static equivalence relation $\equiv$ over Bp so that the least element $[\mu_J]$ of $Mp(B, F)\cap\Gamma p(J)$ provides a witness of the standard model of $(P, =)$. (Under a weak condition on the cardinality of B and F. Precise arguments can be seen in the next chapter.)

and

<2> the choice of a representative of $[\mu_J]$ modifies the procedural semantics of $(P, \equiv)$ while preserving the influence of $\equiv$, i.e. the declarative semantics of $(P, \equiv)$ as a whole. This meta-fuzzyness of inference rules, i.e., the flexibility of the choice of a representative from $[\mu_J]$ is a theoretical advantage of B-valued unification.

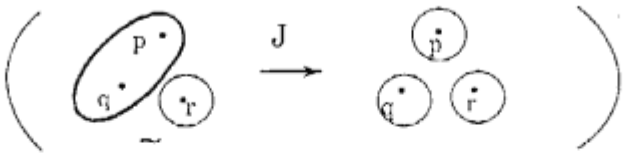
How can we apply this theoretical advantage to a concrete example? One possible answer might be the following. Suppose we are given an algorithm AL which determines $\equiv$ over Bp (or more generally, over $A(\Pi p, V)$). Translate $\equiv$ to B-valued unification to decide $\sim$, J and so $[\mu_J]$ in $Mp(B, F)\cap\Gamma p(J)$. A representative μ_J of $[\mu_J]$ surely decides the procedural semantics of $(P, \equiv)$ which is equal to the one based on the original algorithm AL. Here, we can change AL to another algorithm AL' to the extent that AL' is decided by another representative μ'_J of $[\mu_J]$. This new AL' may be

superior to AL in programming efficiency and/or answer gathering process. (Considering the possible existence of infinite branch(es) of a search tree in the practical phase, the power of the answer enumeration might be greater than you expect.)

v). We present a witness of Remark 5 and Remark 7. Let $\Delta=\{W_1\}$ and $\Lambda=\{c\}$ be such that $W_1=\{r\}$. Then, $Ep(\Delta, \Lambda)$ satisfies the conditions in Theorem 2-3-20. In this case, [i] becomes the least element of $Ep(\Delta, \Lambda)\cap Mp(B, F)$ and i is a (Δ, Λ)-fixed representative of [i]. Here, we notice that no representative (not necessarily (Δ, Λ)-fixed) of [i] has $\{0, 1\}$ as the range. More generally, let J be such that $J(p)=p$, $J(q)=p$, $J(r)=r$. Then, $Ep(\Delta, \Lambda)\cap\Gamma p(J)$ satisfies the conditions in Theorem 2-4-3. In this case, [j] becomes the least element of $\Gamma p(J)\cap Ep(\Delta, \Lambda)\cap Mp(B, F)$ and j is a J-faithful, (Δ, Λ)-fixed representative of [j]. Here again, we notice that no representative of [j] has $\{0, 1\}$ as the range. Moreover, it is easy to check that both [i] and [j] become witnesses of the relations (modifications of (B') and (C')) stated in Remark 7.

II. Here, we assume that $p\sim q$ only.

二) Suppose J is defined by $J(p)=p$, $J(q)=q$, $J(r)=r$.



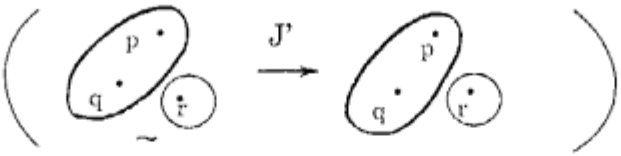
Then, as before, $Pp=\Gamma p(J)$, and $[p_J]=[p]$. Now, to enrich informations obtained positively, there are mainly two method.

One is to enlarge the equivalence relation $\equiv$ based on $\sim$ and J. ($[p]_{\equiv}=\{p\}$, $[q]_{\equiv}=\{q\}$, $[r]_{\equiv}=\{r\}$.) In this case, by changing the definition of J to

$J'(p)=J'(q)=p$, $J'(r)=r$,

the corresponding equivalence relation is transformed from $\equiv$ to $\equiv'$ such that

$[p]_{\equiv'}=\{p, q\}$, $[r]_{\equiv'}=\{r\}$.



Thus, the situation has changed to $\Gamma p(J')=\{[T], \{\perp], [f], [j]\}$ and $Mp(B, F)\cap\Gamma p(J')=\{[T], [f], [j]\}$ and the least element (of $Mp(B, F)\cap\Gamma p(J')$) $[\mu_{J'}]=[f]$. The resulting success set becomes

$Sp(f, \sim)=\{p, q\}$.

This may be restated as [f] has changed its status from a negative world w.r.t. (B, F, $\sim$, J) to a positive world w.r.t. (B, F, $\sim$, J'). In these transformations, as we have already seen before, elements having essentially intermediate value-assignment like [j] and/or [h] can't contribute at all, because each least element $[\mu_J]$ must have a representative whose value range is $\{0, 1\}$.

Another is to loose the restriction on values which are supposed to be "true" in B-valued sense. By employing a complete filter $F'\supset F$ such that $c\in F'$, we can staple two classes $[f]_F$ and $[h]_F$ into one equivalence class $[f]_{F'}$ in the sense of B-valued F'-model (generic extension). What is the merit of this sort of contribution of maps having intermediate values modulo F. A direct contribution is, of course, to widen the flexibility of choosing a representative. However, the essential contribution will be done when we evaluate B-values at each unification step. (The merit of the evaluation is discussed in chapter 5). By CWA, this loosening of restriction on F may be restated as $[h]_F$ has changed its status from a negative world w.r.t. (B, F, $\sim$, J) to a positive world w.r.t. (B, F', $\sim$, J).

Here, the key point is that the above two methods are disjoint to each other in the sense that, owing to their characteristics, the former method never affect the family of classes changed by the latter method and the latter can't influence the family of classes transformed by the former universally unificational technique. And so far, the only used technique which

fundamentally influences CWA has been the former one. In this sense, B-valued technique really propose "profound insight into CWA." Talking about CWA, "the notion of B-valued negation as failure" will be discussed in another paper.

Chapter IV
Universal Unification from a Viewpoint of LIFE-III

§4-0. Introduction

There have been many streams whose aims are to enrich the field of logic programming. One attempt is to combine functional programming technique with logic programming and another is to embed constraint into logic programming and yet another is to generalize the truth-value domain from {1, 0} to a complete lattice. The strategy to employ universal unification based on an equational theory E instead of the usual syntactical unification can be interpreted as a special case in either functional or constraint (or both) stream(s). However, in many practical cases where programs treat general knowledges, it is almost impossible to axiomatize the equivalence relation as a formal neat equational theory which we would like to use for the generalized unification step. This is because a knowledge-based model which determines the equivalence relation over B_P comes first and then comes a possible candidate which formally describe the relation and hopefully gives an algorithm to be managed within a computable program. So, in many cases where we can't take a formal equational theory at the starting point, we ought to employ other methods which can treat and represent the equivalence relation. Considering this fact, we dare claim that there is no necessity that the notion of universal unification is exclusively defined by means of an axiomatic equational theory E, because the purpose of our employing universal unification is to loose the restriction on the possible candidates of unifiers at each unification step. In other words, we can define the notion of universal unification in general based on an equivalence relation $=$ over the set of all atoms used in a logic program P, under the suitable conditions like there is an

effective algorithm to enumerate complete set of universal unifiers w.r.t $\equiv$ for any atoms A, B. (Here, of course, θ is a $\equiv$-unifier for A and B iff $A\theta \equiv B\theta$.)

What is the merit of thus generalizing the notion of universal unification? One interesting advantage is that we might be able to characterize the class of pairs (P, $\equiv$) such that $\equiv$ can't be defined by means of an equational theory method but $\equiv$-unification over P acts importantly in the practical phase. The main purpose of this paper is to abstract a few classes of pairs (P, $\equiv$) such that there are proper effective algorithms to enumerate complete sets of $\equiv$-unifiers under certain simple conditions respectively. In section 4-1, we define the general notion of universal unification, using the concept of "substitution transitive" equivalence relation (the definition will appear soon) and propose an example of general $\equiv$-unification which lies outside the conventional concept of universal unification based on an equational theory. Then, we investigate the notational similarity and difference between the generalized new and the functional old. Section 4-2 is devoted to review the notion of Boolean-valued unification. Though, by definition, we can regard Boolean-valued unification as the marging pond of functional, constraint and many-valued streams, we study the property of it only from an angle of universal unification here in this chapter. By doing so, in §4-3, we propose three classes of pairs $\{(P, \equiv) | \cdots\}$ which satisfies the above mentioned condition with the help of Boolean-valued technology. Lastly, in §4-4, we propose a B-valued standard model of a logic program based on a universal unification, using the result of §2-2 in chapter II.

§4-1. Universal unification

Definition 4-1-1.

Let $\equiv$ be a substitution transitive equivalence relation over $A(\Pi_P, V)$. Then, a substitution θ over $T(\Sigma_P, V)$ is a universal unifier for A and B w.r.t. $\equiv$ iff $A\theta \equiv B\theta$ ⊣

A typical example of universal unification can be given by means of an equational theory over $T(\Sigma_P, V)$.

Example 4-1-2. Let E be a (Horn clause) equational theory over $T(\Sigma_P, V)$. E can define a congruence relation $\equiv$ over $T(\Sigma_P, V)$ by

$(\forall \sigma, \tau \in T(\Sigma_P, V))(\sigma \equiv \tau$ iff $E \vdash \sigma = \tau)$.

This relation can be extended over $A(\Pi_P, V)$ by the usual manner, that is, for any $A(s_1, \cdots, s_m), B(t_1, \cdots, t_n) \in A(\Pi_P, V)$,

$A(s_1, \cdots, s_m) \equiv B(t_1, \cdots, t_n)$ iff $A^{\#} = B^{\#}$ and $m = n$ and $s_i \equiv t_i$ for $1 \leqq i \leqq n$.

Here, from the property of equational theory, since every free variable is (implicitly) universally quantified, thus defined equivalence relation $\equiv$ over $A(\Pi_P, V)$ satisfies the substitution transitivity. ⊣

There are many equivalence relations over $A(\Pi_P, V)$ which can't be defined by an equational theory. For example, let $\Phi_1, \cdots, \Phi_n$ be unary 2nd order predicates over $A(\Pi_P, V)$ which are pairwise disjoint (i.e, $\{ A \in A(\Pi_P, V) \mid \Phi_i(A) \} \cap \{ B \in A(\Pi_P, V) \mid \Phi_j(B) \} = \emptyset$ for $i \neq j$) and satisfy the condition that

$(\forall A \in A(\Pi_P, V))(\forall \theta$: substitution over $T(\Sigma_P, V))(\Phi_i(A) \rightarrow \Phi_i(A\theta))$

for $1 \leqq i \leq n$. Using $\Phi_1, \cdots, \Phi_n$, we can decompose $A(\Pi_P, V)$ into a set of equivalence classes in an obvious manner. Then, the resulting equivalence

relation becomes substitution transitive. (In this case, there may be a lot of equivalence classes which are singletons.) Here, the point is that there is no assurance that we can define an equational theory E which can express thus defined equivalence relation. The next example can be seen as an abstract version of this case.

Example 4-1-3. Let W be an arbitrary set and $f : B_P \to W$ be a map. Define an equivalence relation $\equiv_f$ over $A(\Pi_P, V)$ by, for any $A, B \in A(\Pi_P, V)$,

$A \equiv_f B$ iff $f(A\rho) = f(B\rho)$ for any ground substitution ρ for A and B.

Then, it is easy to check that thus defined relation $\equiv_f$ is really an equivalence relation over $A(\Pi_P, V)$ and satisfies the substitution transitivity. ⊣

Now, we have presented two typical methods which define substitution transitive equivalence relation over $A(\Pi_P, V)$. A natural question is, "Is there any relation between these two methods?" We are going to answer this question in a more general setting. Let $\equiv$ be a substitution transitive equivalence relation over $A(\Pi_P, V)$. Define $f : B_P \to B_P/\equiv$ so that, for any $A \in B_P$,

$$f(A) = [A]_\equiv .$$

Then, by definition, for any $A, B \in B_P$,

$A \equiv B$ iff $f(A) = f(B)$,

that is,

$$\equiv \subseteq B_P \times B_P = \equiv_f \subseteq B_P \times B_P$$

where $\equiv_f$ is the equivalence relation over $A(\Pi_P, V)$ induced by f as in Example 1-3.

Using this fact, we notice that, for any $A, B \in A(\Pi_P, V)$,

A and B are universally unifiable w.r.t. $\equiv$

iff $(\exists\tau : \text{substitution})(A\tau \equiv B\tau)$

$\leftarrow$substitution transitivity

iff $(\exists\rho : \text{ground substitution})(A\rho \equiv B\rho)$

iff $(\exists\theta : \text{substitution})(\forall\rho : \text{ground substitution})(A\theta\rho \equiv B\theta\rho)$

iff $(\exists\theta : \text{substitution})(\forall\rho : \text{ground substitution})(f(A\theta\rho) = f(B\theta\rho))$

iff $(\exists\theta : \text{substitution})(A\theta \equiv_f B\theta)$

iff A and B are universally unifiable w.r.t. $\equiv_f$.

However, this does not necessarily assert that

$$\equiv \; = \; \equiv_f .$$

This is because the universal unifier τ w.r.t. $\equiv$ and the universal unifier θ w.r.t. $\equiv_f$ may be different. Nevertheless, it is easy to see that, without loss of generality, $\equiv \subset \equiv_f$, that is,

$A \equiv B \Rightarrow A \equiv_f B$ for any $A, B \in A(\Pi_P, V)$.

The converse direction does not always hold.

Example 4-1-4. Let P be the program $\begin{cases} A(a) \leftarrow B(X) \\ B(b) \leftarrow \end{cases}$ and $E = \{ a = b \}$,

in addition to usual axioms of equality. Then, $U_P = \{ a, b \}$ and $B_P = \{ A(a), A(b), B(a), B(b) \}$. Let $A(\Pi_P, V) = B_P \cup \{ A(X), A(Y), B(X), B(Y) \}$ by taking $V = \{ X, Y \}$. Then, E defines an equivalence relation $\equiv$ over $A(\Pi_P, V)$ such that $\equiv = \{ (A(a), A(a)), (A(a), A(b)), (A(b), A(a)), (A(b), A(b)), (B(a), B(a)), (B(a), B(b)), (B(b), B(a)), (B(b), B(b)) \}$. However, we can't assert, for example, $A(X) \equiv A(Y)$ because $\neg (E \vdash X = Y)$. On the other hand, by definition, we can say that both $A(X) \equiv_f A(Y)$ and $B(X) \equiv_f B(Y)$. $\dashv$

As for the converse, we need a condition over $\equiv$.

Definition 4-1-5.

1. Let $\equiv$ be a substitution transitive equivalence relation over $A(\Pi_P, V)$. Then, $\equiv$ is substitution complete iff

$(\forall A, B \in A(\Pi_P, V))(A \equiv B$ iff $A\rho \equiv B\rho$ for any ground substitution ρ for A and B).

2. Let E be an equational theory in any form. Then, E is substitution complete iff $\equiv_E$ induced by E is substitution complete. ⊣

It is obvious that, if $\equiv$ is substitution complete, then $\equiv = \equiv_f$, where $\equiv_f$ is defined as above by $f: B_P \rightarrow B_P/\equiv$.

By the way, what is the content of the notion of substitution completeness of an equivalence relation $\equiv$ over $A(\Pi_P, V)$ from a viewpoint of universal unification. The aim of our employing universal unification is, of course, to obtain an enlarged success set compared with the original success set obtained through the syntactical unification. For this purpose, we need an equivalence relation $\equiv_P$ over B_P. Here comes the role of an equational theory E over $T(\Sigma_P, V)$ and/or $f: B_P \rightarrow W$ etc, which possibly determine $\equiv_P$ over B_P.

Remark: As is already touched in introduction, concerning an equational theory E, we should point out the following. Roughly speaking we can categorize equational theories into two classes, that is, the class of a priori E and that of a posteriori E. The latter consists of E which is defined semantically as representing a relation on the domain of discourse. Now, concerning the latter case, the formal theory E is hoped to faithfully reflect the

expected equivalence relation $\equiv_D$ over the domain D, that is, E is considered to satisfy the condition that, for any $\sigma, \tau \in D$,

$$E \vdash \sigma = \tau \qquad \text{iff } \sigma \equiv_D \tau .$$

Here, in the case of logic programming, the intended domain of discourse is the Herbrand universe U_P. However, even if E faithfully represents $\equiv_P$ over U_P, there is no necessity that E is uniquely determined even modulo theories. In other words, there may be equational theories E, E' such that $\equiv_E \neq \equiv_{E'}$ but $\equiv_E | B_P \times B_P = \equiv_{E'} | B_P \times B_P$, where $\equiv_E$ and $\equiv_{E'}$ are equivalence relations over $A(\Pi_P, V)$ based on E and E' respectively. ⊣

Now, returning to the general situation, let $\equiv_1$, $\equiv_2$ be equivalence relations over $A(\Pi_P, V)$ such that $\equiv_1 \supset \equiv_2$ and $\equiv_1 \lceil B_P \times B_P = \equiv_2 \lceil B_P \times B_P$. Consider refutations of $P \cup \{G\}$ using universal unification w.r.t. $\equiv_1$ and $\equiv_2$ respectively. Since $\equiv_1 \lceil B_P \times B_P = \equiv_2 \lceil B_P \times B_P$, G has a refutation R_1 w.r.t. $\equiv_1$-unification iff G has a refutation R_2 w.r.t. $\equiv_2$-unification (and so especially, the success set of P w.r.t. $\equiv_1$ is the same as the success set of P w.r.t. $\equiv_2$). More precisely, for any $A, B \in \Lambda(\Pi_P, V)$,

A and B has a $\equiv_1$-unifier $\theta_1 \Rightarrow$ ($\exists \tau$: substitution) (A and B has a $\equiv_2$-unifier $\theta_1 \tau$)

and

A and B has a $\equiv_2$-unifier $\theta_2 \Rightarrow$ A and B has a $\equiv_1$-unifier θ_2 .

This means we can always obtain a more general answer substitution for $P \cup \{G\}$ if we use $\equiv_1$-unification than the case using $\equiv_2$-unification. This fact is rather useful to enumerate all answers for a given goal G in the practical phase as the following simple example illustrates.

Example 4-1-6. Take the program P as in Example 1-4. let $\equiv_2$ be the equivalence relation over $A(\Pi_P, V)$ generated by $E = \{ a=b \}$ and $\equiv_1$ be the equivalence relation over $A(\Pi_P, V)$ generated by $E=\{X=Y\}$. Let $G = \{\leftarrow A(Y)\}$. Then, if we use $\equiv_2$-unification, we get two answer substitutions $\{Y\leftarrow a\}$ and $\{Y\leftarrow b\}$ independently through twice search repetition. On the other hand, if we use $\equiv_1$-unification, we get only one answer substitution ε (empty substitution) to obtain the same result. Here, the complexity of determining that ε is the answer substitution for G w.r.t. $\equiv_1$-unification is essentially the same as the complexity of determining that $\{ Y\leftarrow b \}$ is the answer substitution for G w.r.t. $\equiv_2$-unification. So, the efficiency using $\equiv_1$-unification is obvious to the extent of search times. (In this case, we need not to check the universal unification of B(X) and B(b) twice.) ⊣

As the above example typically shows, we can conclude in general that, under the condition that $\equiv_1 \supset \equiv_2$ and $\equiv_1 \lceil B_P \times B_P = \equiv_2 \lceil B_P \times B_P$,

1. If the efficiency to enumerate complete set of $\equiv_1$-unifiers is not worse than the efficiency to enumerate complete set of $\equiv_2$-unifiers, we should employ $\equiv_1$-unification.

2. Even if the efficiency to enumerate complete set of $\equiv_1$-unifiers is worse than that of $\equiv_2$-unifiers, the total efficiency to find answers for a given goal G still depends on the character of the program P.

From this observation, we notice that, in a sense, it is best for us to find the maximum element of $R(\equiv_p) = \{ \equiv_i \mid \equiv_i$ is a substitution transitive equivalence relation over $A(\Pi_P, V)$ such that $\equiv_i \lceil B_P \times B_P = \equiv_p \}$, where $\equiv_p$ is the expected

equivalence relation over Bp and the partial order is taken for the set inclusion. In this context, the above definition ensures that

Proposition 4-1-7.

$\equiv$ is maximum in $R(\equiv_p)$ w.r.t. $\subseteq$, i.e., $\equiv = \cup R(\equiv_p)$

iff

$=$ is substitution complete.

Proof: Let $\equiv_p$ be an equivalence relation over Bp and $R(\equiv_p)$ be as above. By taking $\cup R(\equiv_p)$, it is obvious that $R(\equiv_p)$ has the maximum element. (Substitution transitivity is trivially preserved by union.) Let $\equiv_m$ be the maximum element of $R(\equiv_p)$ and $=$ be a substitution complete element of $R(\equiv_p)$. Then, $\equiv \subseteq \equiv_m$. In the following, assuming $= \neq \equiv_m$, we will get a contradiction.

Suppose there are $A, B \in (A(\Pi p, V) - Bp)$ such that

$$A =_m B \quad \text{but} \quad A \not\equiv B.$$

Since $\equiv_m$ is substitution transitive,

$(\forall \rho$: ground substitution$)\,(A\rho \equiv_m B\rho)$

$\Rightarrow$ $(\forall \rho$: ground substitution$)\,(A\rho =_m B\rho)$ by definition of $R(=_p)$

$\Rightarrow$ $(\forall \rho$: ground substitution$)\,(A\rho \equiv B\rho)$

$\Rightarrow$ $A \equiv B$ by substitution completeness of $\equiv$.

So, we get a contradiction. □

By the above Proposition 4-1-7, given an equivalence relation $\equiv_p$ over Bp, substitution complete equivalence relation $\equiv$ over $\Lambda(\Pi p, V)$ which extends $\equiv_p$ is uniquely determined. From this fact, it is of worth defining the following terminology.

Definition 4-1-8. Let $\equiv$ be a substitution transitive equivalence relation over $A(\Pi_p, V)$. Then, $\equiv_f$ defined as above by means of $f : Bp \rightarrow Bp/\equiv$ is called " the substitutive completion of $\equiv$. " ⊣

There is another model theoretic viewpoint which becomes the theoretical background of the notion of substitute completeness.

Proposition 4-1-9. Let E be a substitution complete (Horn clause) equational theory over $T(\Sigma p, V)$. Then, for any $\sigma, \tau \in T(\Sigma_p, V)$,

$$E \vdash \sigma = \tau \quad \text{iff} \quad E \underset{U_p/R}{\vdash} \sigma = \tau \text{ for all equivalence reltion R over } U_P$$
$$\text{iff} \quad Up/\equiv_E \vDash \sigma = \tau$$

where $\equiv_E$ is the equivalence relation over $T(\Sigma p, V)$ based on E.

Proof: Let E be a substitution complete equational theory over $T(\Sigma p, V)$. Let $\sigma, \tau \in T(\Sigma p, V)$.

1. $E \vdash \sigma = \tau \quad \Rightarrow \quad E \underset{U_p/R}{\vDash} \sigma = \tau$ for all equivalence relation R over Up
$\Rightarrow \quad Up/\equiv_E \vDash \sigma = \tau$

   is trivial.

2. $Up/\equiv_E \vDash \sigma = \tau$

$\Rightarrow$ ($\forall$ ground substitution p) ($Up/\equiv_E \vDash \sigma p = \iota p$)

$\Rightarrow$ ($\forall$ ground substitution p) ($\sigma p \equiv_E \tau p$)

$\Rightarrow \quad \sigma \equiv_E \tau$, because E is substitution complete

$\Rightarrow \quad E \vdash \sigma = \tau$.

(We implicitly assume that E contains basic axioms of equality or uses the equality inference rules. Precisely speaking, we should define the notion of "substitutive completion" over $T(\Sigma_P, V)$ instead of $A(\Pi_P, V)$. However, since the similarity is obvious, we omit the definition.) □

Note the point that the above σ, τ may include variables. There may be $\sigma, \tau \in (T(\Sigma_P, V) - U_P)$ such that

$E \models_{U_P/R} \sigma = \tau$ for all equivalence relation R over U_P

but

$\neg(E \vdash \sigma = \tau)$

for a certain substitution non-complete equational theory E. (The above Example 1.4 fits to this situation.) Considering the property of logic programming, where positive informations are expected to be managed within Herbrand universe, the above model theoretic property of substitution completeness also implicitly authorize the legitimacy of the notion.

§4-2. B-valued unification from a viewpoint of universal unification

First of all, by comparing two definitions, the reader should recognize that every universal unification is a B-valued unification. To see this at a theoretical level, it is enough to take $\sim = \equiv$, where $\sim$ is a substitution transitive relation over $A(\Pi_P, V)$ used in the construction of B-valued unification, and construct $[[\ \ ]] : B_P \to B$ so that it is subordinated to $\sim$ in the sense that

$(\forall A, B \in A(\Pi_P, V))(A \sim B \to (\forall \rho$: ground substitution for A and B)
$([[A\rho]] = [[B\rho]]))$.

This viewpoint permits the flexibility of $[[\ \ ]] : B_P \to B$.

Let $[[\ \ ]]$, $[[\ \ ]]'$ be maps which are subordinated to $\sim$. Then, for any $A, B \in A(\Pi_P, V)$,

θ is a B-valued unifier for A and B w.r.t. $\sim$ and $[[\ \ ]]$

iff $A\theta \sim B\theta$

iff θ is a B-valued unifier for A and B w.r.t. $\sim$ and $[[\ \ ]]'$.

So, for example, especially taking $B = 2$ and $[[\ \ ]] \in 2^{B_P}$ such that

$(\forall \sigma \in B_P)([[\sigma]] = 1)$,

we notice that this kind of Boolean-valued unification entirely depends on only $\sim$, and the subordinated $[[\ \ ]]$ can be neglected.

Note: Here, we ought to remark the next important point. Although any choice of $[[\ \ ]]_{\equiv} : B_P \to B$ subordinated to $\equiv$ may not change the procedural semantics of $(P, \equiv)$, the declarative semantics of Boolean-valued unification depends on $[[\ \ ]]_{\equiv}$ in itself. In other words, it is a map $[[\ \ ]]_{\equiv} : B_P \to B$ that makes up the meaning of the program $(P, \equiv)$ based on $\equiv$-unification. And usually, only one element $\mu_{\equiv}$ of $\{ [[\ \ ]]_{\equiv} \in B^{B_P} \mid (U_P, [[\ \ ]]_{\equiv}) \text{ is a Herbrand } F\text{-}$

model of P such that [[ ]]$_{\equiv}$ is subordinated to $\equiv$ } becomes an expected candidate for the proper meaning of (P, $\equiv$) for each chosen B and F, in the sense that (U_P, $\mu_{\equiv}$) becomes the standard F-model of (P, $\equiv$) in B-valued semantics. In §4-4, we will discuss this approach, where the notion of "subordinate" is shown to be a special case of "J-faithfulness" concerning [[ ]] and the existence of the standard model of (P, $\equiv$) is proved. However, as will be discussed soon, we can ignore this kind of argument at the elementary level application of B-valued unification to universal unification. ⊣

On the other hand, in the case that $\equiv$ is substitution complete, we can realize $\equiv$ by B-valued unification w.r.t. the map [[ ]] : $B_P \to B$ such that ($\forall \sigma, \tau \in B_P$) ($[[\sigma]] = [[\tau]]$ iff $\sigma \equiv \tau$) and the trivial relation $\sim$. (Remember that substitution complete $\equiv$ is uniquely determined by $\equiv | B_P \times B_P$ by the map f : $B_P \to B_P/\equiv$.) So, for example, Example 4-3-6 in the next section can be regarded as a typical case belonging to this interpretation.

Thus, we have shown two fundamental but rather extreme techniques to realize $\equiv$ by a Boolean-valued unification. However, in the practical phase, we often harmonize the above two to produce a new equivalence relation which usual equational methods can't embody by themselves. For example, Example 4-3-5 in the next section explains the phenominon.

In the examples presented so far, we have ignored not only the structures of B as a complete Boolean algebra, but also even the resulting value $[[\sigma]] \in B$ for each particular $\sigma \in B_P$. This means $\sim$ and [[ ]] works only to define an equivalence relation over B_P. At the elementary level where Boolean-valued unification is interpreted as a kind of universal unification in general, this sort of loose restriction on [[ ]] is sufficient. In other words, in a naive application of Boolean-valued unification, we can neglect the concrete assignment of [[ ]]:$B_P \to B$.

§4-3. Universal unification from a viewpoint of B-valued unification

Let $\equiv$ be a substitution transitive equivalence relation over $A(\Pi_P, V)$. Using the similar arguments as in the case of E-unification where E is an equational theory, it is easy to see that the derivation using $\equiv$-unification in general is logically both sound and complete in the sense that there is a standard model for $(P, \equiv)$.

Now, in order to enjoy the advantage of the logical completeness at a pure logic programming level, all we need is an algorithm to enumerate complete set of $\equiv$-unifiers. A typical algorithm is the method of narrowing which can be applied to the class $\{(P, \equiv) | \equiv$ is defined by means of a canonical (confluent and terminating) term rewriting system$\}$. In the following, we would like to categorize different classes of pairs $(P, \equiv)$, to which effective proper algorithms to enumerate complete sets of $\equiv$-unifiers can be applied. For this purpose, we employ Boolean-valued technology.

Now, in order to obtain a complete set of B-valued unifiers for arbitrary A, $B \in A(\Pi_P, V)$, there should be an algorithm to decide both $\sim$ and $[[\ \ ]]: B_P \rightarrow B$ first of all. Next, even if there is an appropriate algorithm to decide $\sim$ and $[[\ \ ]]$, it does not necessarily assert the existence of the enumeration algorithm of B-valued unifiers. However, in a practical situation, we usually employ only those $\sim$ and $[[\ \ ]]: B_P \rightarrow B$ such that we can easily decide whether θ is a B-valued unifier for A and B (w.r.t. $\sim$ and $[[\ \ ]]$) or not. Of course, in general, deciding whether θ is a B-valued unifier or not and enumerating all possible B-valued unifiers for A and B may be different. Nevertheless, there are many cases where algorithms to define $\sim$ and $[[\ \ ]]$ are enough to decide the enumeration algorithms.

As we have mentioned in the above paragraph, when we decide a concrete language in the practical phase, we fix $\sim$ so that there is an effective algorithm to decide whether $A \sim B$ or not for any $A, B \in \Lambda(\Pi_P, V)$ at the starting position. Several simple examples of $\sim$ may be the following.

Example 4-3-1. Let $A, B \in \Lambda(\Pi_P, V)$. Define $\sim$ so that

$A \sim B$ iff A and B are syntactically equal.

(Thus defined $\sim$ will be called "discrete.") ⊣

In this case, Boolean-valued unification becomes the usual syntactical unification and $[\![\ \ ]\!] : B_P \to B$ can't influence the definition of B-valued unification except B-value estimation, no matter what $[\![\ \ ]\!]$ may be.

Example 4-3-2. Define $\sim$ so that,

for any $A, B \in \Lambda(\Pi_P, V)$, $A \sim B$.

The equivalence class over $\Lambda(\Pi_P, V)$ determined by $\sim$ is only $\Lambda(\Pi_P, V)$ itself. (Thus defined $\sim$ is called "trivial") ⊣

In this case, since $\sim$ does not influence the definition, B-valued unifier is completely determined by the map $[\![\ \ ]\!] : B_P \to B$.

Above two cases are the extreme cases. A little more interesting is the technique defined in §1. Let W be an arbitrary set and suppose there is an effective algorithm to define a map $f : B_P \to W$. Let $A, B \in \Lambda(\Pi_P, V)$. Define $\sim$ so that

$A \sim B$ iff $f(A\rho) = f(B\rho)$ for any ground substitution ρ for A and B.

A concrete example is ;

Example 4-3-3. Let Π_P consist of predicates whose arities are finite, indefinite, unordered sets assigned by some kinds of attributes. That is, every element of Π_P has a form $\psi(\{X_1/\alpha, X_2/\beta, X_3/\gamma, \cdots\})$
where $\alpha, \beta, \gamma, \cdots$ are attributes used in P, say,
$\{\alpha = \text{what}, \beta = \text{when}, \gamma = \text{where}, \cdots\}$ etc. (Especially by fixing $\alpha = 1, \beta = 2, \gamma = 3, \cdots$, we notice that this general definition includes the usual definition of predicates with fixed ordered arities.)

Let $A, B \in A(\Pi_P, V)$. Define $\sim$ so that $A \sim B$ iff

the argument assigned by α in $A^{\#}$ = the argument assigned by α in $B^{\#}$
and the argument assigned by β in $A^{\#}$ = the argument assigned by β in $B^{\#}$
where = means syntactical equality including variable names.

Then, it is obvious that thus defined $\sim$ becomes a substitution transitive equivalence relation over $A(\Pi_P, V)$. Moreover, it is easy to see that $\sim$ becomes substitution complete.

So, by taking $W = B_P/\sim$ and defining $f: B_P \to W$ so that $(\forall \sigma \in B_P)\ (f(\sigma) = [\sigma]_{\sim})$, we notice that the above definition is equal to that $A \sim B$ iff $f(A\rho) = f(B\rho)$ for any ground substitution ρ for A and B, because of the uniqueness of substitution complete equivalence relation.
Here, using the original naive definition of $\sim$, we notice that there is an effective algorithm to decide $A \sim B$ or not and to enumerate complete set of $\sim$-unifiers for any $A, B \in A(\Pi_P, V)$. ⊣

In general, by taking subordinated map [[ ]], we can categorize $\{(P, \equiv) \mid \equiv$ is defined by argument-wise equality$\}$, where the general notion of

argument-wise equality should be obvious. In any example belonging to this class, the ≡-unification becomes, so to speak, "argument-wise unification." We already know that many practical unification techniques used in world-widely existing programs can be named "argument-wise" in this sense.

On the other hand, there are many techniques with which we can embody [[ ]]:$B_P \to B$ in a program. A few simple but important cases may be the following.

I. The case that B_P is finite.

In this case, whichever method we employ to define [[ ]]: $B_P \to B$, there always exists an algorithm because of the finiteness of B_P. A typical case is;

Example 4-3-4. Let Σ_P consist of finite number of constant symbols, say, $c_1, \cdots, c_n$ and finite number of constructors, say, [ ], { }, < > and finite number of functors, say, f, g, h, etc, where each constructor or functor does not allow the mixed nest whose depth is deeper than a fixed positive number, say, m. Then, the resulting Herbrand universe U_P becomes finite, and so B_P becomes finite. ⊣

II. The case that B is finite.

The restriction of B's finiteness forces us that the equivalence class $B_P/[[\ ]]$ becomes finite and so, at a first glance, the number of candidates with which an element $\sigma \in B_P$ is B-valued unified seems to become vast. However, this is not always true because of the role of ~. We even claim that this case can work to treat many exceptions which usual universal unification based on an equational theory can't handle.

Example 4-3-5.

Define ~, so that, for any A, B ∈ A(Πp, V), A~B iff the first argument of A = the first argument of B.

Let B=4 = { 1, b, ¬ b, 0 }. Select a finite subset Q⊂Bp. Our intention is that Q becomes a collection of atoms which assert exceptional cases w.r.t. some selected predicates in Πp. For that purpose, define [[ ]] : Bp→4 so that,

$$(\forall\sigma \in Bp - Q)([[\sigma]] \in \{1, 0\}) \wedge (\forall\sigma \in Q)([[\sigma]] \in \{b, \neg b\}).$$

Then, at each B-valued unification step, for any σ, τ ∈ (Bp − Q), we can unify σ and τ (using empty substitution) iff [[σ]] = [[τ]] and the first argument of σ = the first argument of τ.

Otherwise, we unify σ and τ iff σ = τ.

(Here, we consciously use the simplification that

(∀σ, τ ∈ Q) (σ ≠ τ → the first argument of σ ≠ the first argument of τ) by taking B=4.) ⊣

In the practical phase, the above assignment substancially means that we make a table of exceptional atoms and cite the table at each unification step as references. To be more precise, we use the notion of "all but finite exception unifier (a.b.f.e.-unifier)". Though we omit the precise definition concerning this concept, the reader may easily recognize the total image by the following simplest case study. Let A(a,b)∈Q. Suppose the situation that we want to unify a goal A(X,Y) and a clause A(a,c) ← . In this situation,we employ the substitution {X←a} as the unifier and check at the final stage where the derivation succeeds and gets an answser substitution θ=θ1···θn, whether we use a wrong substitution w.r.t. the variable Y or not. Here, "wrong" means "Y←b". Then, the above {X←a} is called a.b.f.e.-unifier for A(X,Y) and A(a,c) w.r.t. Q. Of course, the definition heavily depends on ~, but in anyway the merit of our using a.b.f.e.-unifier is obvious from a viewpoint of efficiency.

In general, let's call an equivalence relation $\equiv$ which is defined by means of both a suitable substitution transitive equivalence relation $\sim$ over $A(\Pi_P, V)$ and a finite exceptional set Q, "effective $\sim -Q$ relation." Then, we can categorize the class $\{(P, \equiv) \mid \equiv$ is an effective $\sim -Q$ equivalence relation in the sense of Boolean-valued unification$\}$.
For this kind of class, we can utilize a.b.f.e.-unification and obtain an effective algorithm to enumerate complete set of $\equiv$-unifiers.

III. The other cases

In general, too theoretical construction of $[[\ \ ]]: B_P \to B$ makes the algorithm be practically intractable. However, there are still many cases that we can embody $[[\ \ ]] : B_P \to B$ in a program.

One possible case may be that where we use usual syntactical unification algorithm almost everywhere except on a finite number of equivalence classes in B_P. For example;

Example 4-3-6. Let $\{\psi_1, \cdots, \psi_n\} \subseteq \Pi_P$ be selected predicates used in P with more than one arities. Let $\{t_1, \cdots, t_n\} \subset U_P$ be selected ground terms. Define $\Delta(\psi_i, t_i)$ for $1 \leqq i \leqq n$ so that, $\Delta(\psi_i, t_i) = \{$ground instance of $\psi_i \mid$ the first argument $= t_i\}$.

Let $\equiv$ be the equivalence relation over B_P defined by $\Delta(\psi_i, t_i)$, $1 \leqq i \leqq n$.
(So, especially, for any $\sigma, \tau \in (B_P - \bigcup_{1 \leq i \leq n} \Delta(\psi_i, t_i))$, $\sigma \equiv \tau$ iff $\sigma = \tau$.)

Define $[[\ \ ]] : B_P \to B$ so that

$(\forall \sigma, \tau \in B_P)\ (\sigma \equiv \tau$ iff $[[\sigma]] = [[\tau]])$.

Then, by taking $\sim$ being trivial, we have an effective algorithm to enumerate complete set of B-unifiers w.r.t. $\sim$ and [[ ]]. The resulting B-valued unification obviously embodies the substitutive completion of the original $\equiv$. $\dashv$

In the above, we may employ the method of "two-lane unification", the precise definition of which is omitted. Rough idea is;

1. Firstly, use the usual syntactical unification to obtain possible answer substitutions.
2. Secondly, try the other possibility concerning $\Psi_1, \cdots \Psi_n$.

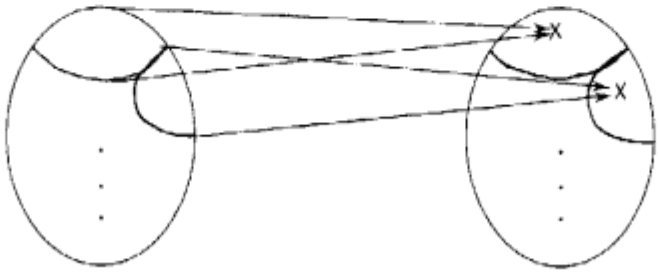
Thus, the machine runs on two-lane street, obeying the command from a suitably implemented program.

Generally, we can categorize the class of substitution transitive equivalence relation $\equiv$ over $\Lambda(\Pi_P, V)$ such that $\equiv$-unification becomes two-lane unification in the above sense and there is an effective algorithm to enumerate complete set of $\equiv$-unifiers. Any element of this class is called "two-lane relation." So, we can present the class $\{(P, \equiv) \mid \equiv$ is a two-lane equivalence relation$\}$ based on two-lane unification.

§4-4. Logical completeness of universal unification from a viewpoint of LIFE-III

Definition 4-4-1. Let $\equiv$ be an equivalence relation over Bp.

Let c: $Bp/\equiv \rightarrow Bp$ be a choice function, i.e., $(\forall[\sigma]\in Bp/\equiv)\ (c([\sigma])\in[\sigma])$. Then, an idempotent function J: $Bp \rightarrow Bp$ is called "the representative of $\equiv$ via c" iff, for any $\sigma \in Bp$, $J(\sigma)=c([\sigma])$.



⊣

Now, if J satisfies B-valued unification condition for a given ~over Bp, the result of the chapter II asserts that the success set of P w.r.t. B-valued unification depends only on $J``Bp(=\{c([\sigma])|[\sigma]\in Bp/\equiv\})$. This fact can be used to obtain a standard model of $(P, \equiv)$ based on the universal unification w.r.t. $\doteqdot$, where $\doteqdot$ is a substitution transitive equivalence relation over $A(\Pi p, V)$.
To be more precise;

Theorem 4-4-2. Let P be a program and $\equiv$ be a substitution transitive equivalence relation over $A(\Pi p, V)$. Let c be a choice function over $Bp/\equiv$ and J be the representative of $\doteqdot$ via c. Let B be a complete Boolean algebra and F be a complete filter over B. Let $[p_J]$ be the least element $\Gamma p(J)\cap Mp(B, F)$ based on J, B, F, where ~ is taken to be $\doteqdot$. Then, (Up, p_J) becomes the standard F-model of $(P, \equiv)$, that is, for any $\sigma \in Bp$,

$(P, =)\vdash \sigma$ iff $\mu_J(\sigma) \in F$.

Especially by specifying $B=2$ and $F=\{1\}$, (Up, μ_J) becomes the standard model of $(P, =)$ in the usual sense, i.e.,

$(P, =)\vdash \sigma$ iff $(Up, \mu_J \models \sigma$.

Proof: From the result of chapter II, it is enough to see that, for any $\sigma \in Bp$,

$(P, =)\vdash \sigma$ iff $\sigma \in Sp(\mu_J, \sim)$.

Now, since our concern is restricted only to ground atoms in this phase, it is sufficient to check that, for any atoms $A, B \in A(\Pi p, V)$,

θ is a ground $=$ -unifier for A and B

iff

θ is a ground B-valued unifier for A and B w.r.t. μ_J and $\sim$.

Here, θ is a ground $=$ -unifier for A and B

$\Leftrightarrow$ $A\theta = B\theta$

$\Leftrightarrow$ $[A\theta]=[B\theta]$ as an element of $Bp/=$

$\Leftrightarrow$ $A\theta \sim B\theta$ and $\mu_J(A\theta)=\mu_J(J(A\theta))=\mu_J(J(B\theta))=\mu_J(B\theta)$

(Remember that μ_J takes $\{0, 1\}$ as the range.)

$\Leftrightarrow$ θ is a ground B-valued unifier for A and B wr.t. μ_J and $\sim$. □

As direct consequences, we notice

Corollary 4-4-3. Let P be a program and E be an equational theory over $T(\Sigma p, V)$. Let $=$ be the finest congruence relation over $A(\Pi p, V)$ induced from

E, and c be a choice function over Bp/≡. Let J be the representative of ≡ via c. Let $[\mu_J]$ be the least element of $\Gamma p(J) \cap Mp(B, F)$ based on J, $B = 2$, $F = \{1\}$, where ~ is taken to be ≡. Then, (Up, μ_J) become the standard model of (P, E), i.e.,

$$(P, E) \vdash \sigma \quad \text{iff} \quad (Up, \mu_J) \models \sigma .$$

Proof: Obvious. □

Remark: The reader should notice that our proposing standard model of universal unification has the Herbrand universe Up, instead of Up/≡. ⊣

Corollary 4-4-4. Let c' : Bp/≡ → Bp be another choice function and J' be the representative of ≡ via c' in the above construction. Then, the least element of $\Gamma p(J) \cap Mp(B, F)$ = the least element of $\Gamma p(J') \cap Mp(B, F)$

Proof: Let $[\mu_{J'}]$ be the least of $\Gamma p(J') \cap Mp(B, F)$.

Using Theorem 4-4-2, for any $\sigma \in Bp$,

$\mu_J(\sigma) \in F$ iff $(P, \equiv) \vdash \sigma$ iff $\mu_{J'}(\sigma) \in F$.

Here, by the construction of $Tp(J) \uparrow \omega$ and $Tp(J') \uparrow \omega$, we can assume that both μ_J and $\mu_{J'}$ takes either 1 or 0 as their Boolean value. (Representatives modulo F)

$$\therefore \bigwedge_{\sigma \in Bp} (\mu_J(\sigma) \leftrightarrow \mu_{J'}(\sigma)) \in F$$

$$\Leftrightarrow [\mu_J] = [\mu_{J'}].$$ □

This means that the obtained B-valued F-model does not depend on the choice function c of ≡.

§4-5. Conclusion

In this chapter, we generalize the notion of universal unification based on an equational theory over logic programming and discover the novelty with the help of Boolean-valued logic programming language scheme LIFE-III by characterizing classes of pairs $(P, \equiv)$ for which there are proper effective algorithms to enumerate complete sets of $\equiv$-unifiers under certain suitable conditions respectively. The newly obtained classes amount to the replacements or alternatives of {$(P, \equiv)|=$ is defined by means of a canonical term rewriting system} based on the narrowing method in the conventional case. The classes include;

1. {$(P, \equiv)|\equiv$ is defined by argument-wise equality} based on argument-wise unification.
2. {$(P, \equiv)|\equiv$ is an effective $\sim_Q$ equivalence relation} based on a.b.f.e.-unification, where Q is the set of exceptions.
   (A concrete program $(P, \equiv)$ belonging to this class can be found in [17].)
3. {$(P, \equiv)|\equiv$ is a two-lane equivalence relation} based on two-lane unification.

For those who might already be familiar with a proposed unification technique like argument-wise unification in the practical phase, the result of this paper becomes the theoretical background (which has not existed so far). On the other hand, for those who have encountered the above techniques like two-lane unification for the first time, the results act as both technical tools and theoretical backgrounds.

Chapter V
A L-fuzzy Inferential System: F-LIFE

§5-1. Definition of F-LIFE

There have been several papers which discuss fuzzy inference (inference with certainty factors or quantitative deduction) on the base of pure Prolog. The rough idea is that they compute a certainty factor when a goal succeeds and try to utilize it to obtain an additional information concerning the goal, in addition to an answer substitution. The certainty factors used in their context are elements of [0, 1] or more generally, elements of a certain kind of (complete) lattices and the unifications used have been syntactical unifications. Here, we employ LIFE-III as the scheme of a new fuzzy inferential system. In other words, we take a complete Boolean algebra B as the range and we permit B-valued unification as the basis of inference. The difference between the conventional and ours may seem to be small at a first glance, but this is not so at all. For example, just consider the value assignment at each B-valued unification step. In case of syntactical unification, since the resulting unified atom should be syntactically equal, it is natural to assign one value to that unique atom (modulo substitutions in case that the atom is not ground). However, in case of B-valued unification, since the candidates of unified atoms may be syntactically different, the value between the chosen atom $A\theta$ in a goal (substituted by a B-valued unifier θ) and an input clause head $C^+\theta$ (substituted by θ) should be fully taken care of. This is one scene where the condition

$[[A\theta\rho]] = [[B\theta\rho]]$ for any ground substitution ρ

in the definition of B-valued unification plays its essential role. From now on, let's discuss some features of our system.

Let B be a complete Boolean algebra on which we want to build an inferential system. The features concerning B are;

1. We can take any set B as the support of B. Thus, each element of B may already be a structured set of data. ⊣

2. (In a usual fuzzy system, $\wedge$ and $\vee$ need not be uniquely interpreted as min and max. Similarly,) On the same support set B, we can impose different kinds of Boolean operations or (Boolean) lattice-orderings. B is one of them. ⊣

3. Since B is a complete Boolean algebra, B is a complete lattice. In some cases, we may ignore the effects of Boolean operation $\neg$ (and so $\rightarrow$) on B (, thus B works only as a general complete lattice), and in some other cases, we many essentially use $\neg$ to estimate values in B. ⊣

4. As a special case, target values in B may become linear w.r.t. the ordering on B and each element of B may be a simple symbol. In this case, the effect of our employing B as a range becomes substantially the same as employing [0, 1] as the range. ⊣

I. Now, suppose B is fixed. Let L be a formal language with which we would like to express the target knowledge K and $(P_L, \equiv)$ be a Horn clause program on L whose deduction is based on a universal unification w.r.t. $\equiv$, which is expected to formally represent K in a usual two-valued manner. By employing $[[\ ]] : B_{P_L} \rightarrow B$, we can generalize the Herbrand interpretation to B-valued Herbrand interpretation (Up, $[[\ ]]$). At the same time, we can utilize $[[\ ]]$ (with the help of $\sim$) to define B-valued unification in order to refine the original $\equiv$-unification and to improve $(P_L, \equiv)$ to $(P_L, [[\ ]])$, a better (and sometimes more efficiently implementable) formal expression of K. Here, as is obvious from the definition of B-valued unification,

5. For a fixed B, we can define many relations $\sim$, $\sim'$ etc over A(Πp, V). Moreover, for a fixed $\sim$, we can choose different maps, say, $[[\ ]]$, $[[\ ]]'$: $B_{P_L} \rightarrow B$ such that, for any atoms A, B and for any substitution θ,

   θ is a B-valued unifier for A and B w.r.t. $\sim$ and $[[\ ]]$

   iff

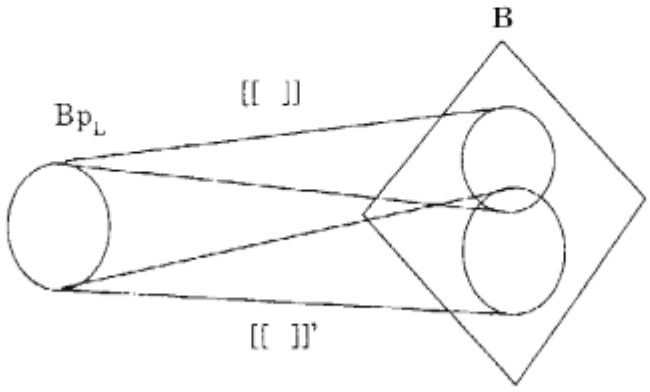
θ is a B-valued unifier for A and B w.r.t. $\sim$ and [[ ]]',

that is,

$[[A\theta\rho]]=[[B\theta\rho]]$ iff $[[A\theta\rho]]'=[[B\theta\rho]]'$

for any ground substitution ρ for $A\theta$ and $B\theta$.

In other words, B-valued unification w.r.t. $\sim$ and [[ ]] and B-valued unification w.r.t. $\sim$ and [[ ]]' become equal modulo value estimation.



⊣

Among many maps, we choose such maps that $(Up_L, [[\ ]])$ becomes a Herbrand F-model in the sense that $(\forall C\in P_L)$ $([[C]]\in F)$ w.r.t. a complete filter F over B. For such a $[[\ ]]: Bp_L \to B$, $(P_L, [[\ ]])$ is called a F-program. The reason of our employing F-program $(P_L, [[\ ]])$ among many other possibilities is,

6. $(P_L, [[\ ]])$ preserves semantic consistency as a program in the sense of B-valued model. ⊣

One obvious feature concerning F-program is,

7. If $(P, [[\ ]])$ is a F-program, then $(P, [[\ ]])$ becomes F'-program for any complete filter $F'\supseteq F$. ⊣

II. Secondly, suppose F is fixed. There are still many candidates $[[\ ]]\in B^{Bp_L}$ for $(P_L, [[\ ]])$ being a F-program. Among them is the notion of standard (or intended) F-model (Up_L, μ) w.r.t. B-valued unification in the sense that, for any atom $\sigma\in Bp_L$,

$P\cup\{\leftarrow\sigma\}$ has a refutation based on B-valued unification w.r.t. $(\sim, \mu)$

iff

$\mu(\sigma) \in F$.

Here, owing to the estimation by F,

8. This particular μ has a flexibility to the extent of

$[\mu] \in B^{Bp_L}/F$ in the sense that

$(\exists\, \mu, \mu' \in [\mu])\,(\mu \neq \mu'$ and $Sp(\mu, \sim) = Sp(\mu', \sim))$.

(see chapter II .) ⊣

In this situation,

9. It is a representative μ_F of $[\mu]$ whose values spread widely inside F that works interestingly. ⊣

III. Thirdly, suppose a standard $[\mu] \in M_{P_L}(B, F)$ is fixed. Then, choose an appropriate representative μ_F of $[\mu]$ by somehow or other. By the B-valued refutation procedure, for any goal G, if there is an B-valued refutation R of $P \cup \{G\}$ with the answer substitution θ, we obtain two values (w.r.t. G)

$v(R)$ (refutation value of G)

and

$v(\theta)$ (answer value of G)

which are based on μ_F. By the choice of μ_F, $v(R)$ and $v(\theta)$ may not be 1, though

10. $v(\theta) \geqq v(R)$

and

$v(\theta) \in F$ (soundness of B-valued refutation)

always holds. Moreover, even

$v(R) \in F$ holds for any special (ground) refutation.

(For the precise argument, see chapter I .) ⊣

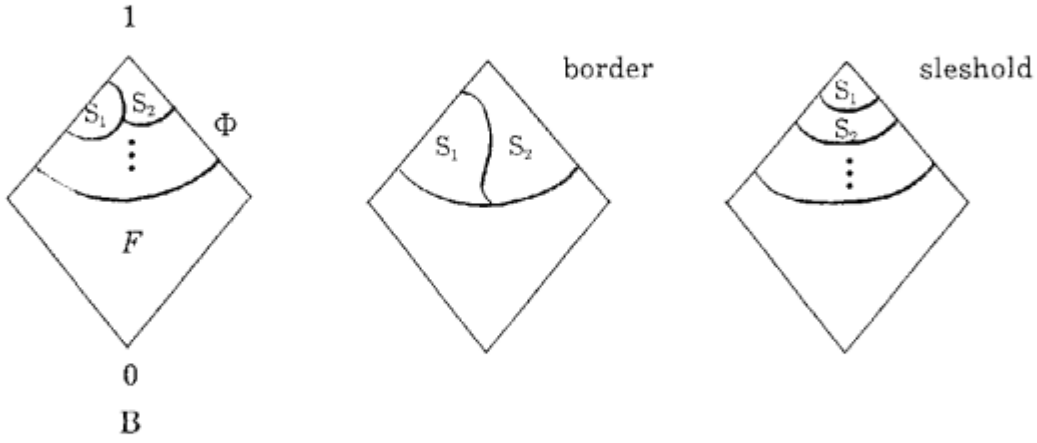
Next, suppose there is an effective condition Φ on F which divides F into pairwise disjoint finite subsets $S_1, \cdots, S_m$. In many cases, $m = 2$ and, in this case, Φ is said to define "a border inside F." On the other hand, if there is a series of complete filters $F_1, \cdots, F_m$ such that

$F_1 \subseteq F_2 \subseteq \cdots \subseteq F_m = F$

and

$S_1 = F_1, S_2 = F_2 - F_1, \cdots, S_m = F - F_{m-1}$,

then Φ is said to define "a sleshold inside F."



In the following, for simplicity, let's consider the case of $m=2$. Then, Φ can be interpreted to be an unary (2-valued) predicate over F. ($S_1 = \{b \in F \mid \Phi(b)\}$, $S_2 = F - S_1$.)

11. For this kind of Φ, we can check whether $\Phi(v(0))$ and/or $\Phi(v(R))$ hold or not. More precisely, let $(\theta_1, \cdots, \theta_n)$ and $(C_1, \cdots, C_n)$ be the sequence of B unifiers and input clauses used for R. Then, to determine $v(R)$, we ought to know "the (i-th) trace value $v(t_i)$ of R" $\mu_F(C_1{}^+\theta_1 \wedge \cdots \wedge C_i{}^+\theta_i)$ for each $1 \leq i \leq n$. (As in chapter I, $\mu_F(C_1{}^+\theta_1 \wedge \cdots \wedge C_i{}^+\theta_i)$ is the abbreviation of $\wedge(\mu_F(C_1{}^+\theta_1\rho) \wedge \cdots \wedge \mu_F(C_i{}^+\theta_i\rho))$) Then, we can check whether $\Phi(v(t_i))$ holds or not for $1 \leq i \leq n$.

⊣

Here, let $Z(\mu_F) = \{(R_1, \xi_1), \ldots, (R_k, \xi_k), \cdots\}$ be the collection of all pairs of a ground refutation R_k and the corresponding answer substitution ξ_k for $P \cup \{G\}$ based on μ_F. Then, we can divide $Z(\mu_F)$ into four different classes $Z_1(\mu_F, \Phi)$, $Z_2(\mu_F, \Phi)$, $Z_3(\mu_F, \Phi)$, $Z_4(\mu_F, \Phi)$ such that

$Z_1(\mu_F, \Phi) = \{(R, \xi) \in Z(\mu_F) \mid$ both $\Phi(v(\theta))$ and $\Phi(v(R))$ hold$\}$

$Z_2(\mu_F, \Phi) = \{(R, \xi) \in Z(\mu_F) \mid \Phi(v(\theta))$ holds but $\Phi(v(R))$ does not hold$\}$

$Z_3(\mu_F, \Phi) = \{(R, \xi) \in Z(\mu_F) \mid \Phi(v(\theta))$ does not hold but $\Phi(v(R))$ holds$\}$

$Z_4(\mu_F, \Phi) = \{(R, \xi) \in Z(\mu_F) \mid$ neither $\Phi(v(\theta))$ nor $\Phi(v(R))$ holds$\}$.

(In case that Φ defines a sleshold, since $\upsilon(\theta) \geqq \upsilon(R)$, $Z_3(\mu_F, \Phi)$ is always empty.)

It is this classification based on Φ that gives us new informations (for the goal G) which we can never obtain by 2-valued interpretations, and this is the place where our elaboration of computing B-values is fully repaid.

IV. Fourthly, suppose Φ over F is fixed. Here, remark the fact that

12. for different choices of representatives μ_F, μ_F' of the same $[\mu]$, $Z(\mu_F)$ and $Z(\mu_F')$ may be different. Of course, even if $Z(\mu_F) = Z(\mu_F')$, the result of classifications of the same $Z(\mu_F)$ becomes different if Φ is changed. In this sense, only the pair choice (μ_F, Φ) uniquely determines the classification. ⊣

Note: It entirely depends on each particular (applicational) case which we should decide firstly, μ_F in $[\mu]$ or Φ over F. ⊣

V). Fifthly, suppose (μ_F, Φ) is fixed. Here, assume the case that, instead of full classification, we happen to realize that we need only the information (R, ξ) belonging to, say, $Z_1(\mu_F, \Phi)$. A naive method to implement this case is, of course, we check every success branch (R, ξ) in $Z(\mu_F)$ one after another, using the judge Φ. However, in many cases, there are possibilities that we can check whether $(R, \xi) \in Z_1(\mu_F, \Phi)$ or not on the halfway of a branch of search tree of $P \cup \{G\}$.

13. If we can determine $(R, \xi) \notin Z_1(\mu_F, \Phi)$ on the halfway, it must be useful from a viewpoint of efficiency because, in this case, we can backtrack from the halfway point where we notice that the full information does not match the requirement of $Z_1(\mu_F, \Phi)$, even if it becomes a success branch $(R, \xi) \in Z(\mu_F)$ at the final leaf node. ⊣

As the final feature of our L-fuzzy inferential system,

14. we have decided to employ extended Horn clause program which permit the existence of "semantically controlling predicates" formally represented by dagger symbol with subscripts $\dagger_1, \cdots, \dagger_n$.

i) Each $\dagger_i$ (as a predicate) has no corresponding relational meaning in our intended knowledge K. Thus, the extended program has $L^{\dagger}=L\cup\{\dagger_1, \cdots, \dagger_n\}$ as the language.

ii) Each $\dagger_i$ is inserted only in a (possibly empty) body part of a clause of the original program P_L. However, the location of each $\dagger_i$ is not restricted to the top (left most) of each clause body.

iii) More than one daggers, say, $\dagger_1, \cdots, \dagger_j$ might be inserted in one clause $C\in P_L$.

Thus, from ii)+iii), for the same original clause $C\in P$ and for the same set of daggers $\{\dagger_1, \cdots, \dagger_n\}$, we can construct many syntactically different extended clauses $C_1^{\dagger}, \cdots, C_m^{\dagger}$ by the location of each $\dagger_i$.

Example 5-1-1. Let $C=A\leftarrow B$. Using $\dagger_1$, we can have two extended clauses $A\leftarrow\dagger_1, B$ and $A\leftarrow B, \dagger_1$. ⊣

As will be explained later, these two extensions might provides different effects.

iv) For each $\dagger_i$, the B-valued unification <u>always</u> succeeds and produces a B-value bi which is properly assigned to $\dagger_i$.

v). Suppose an extension $P_L^{\dagger}$ of P_L is fixed. Let $[[\ ]] : B_{P_L}\rightarrow B$ be an B-valued interpretation of P_L. Then, for the fixed $P_L^{\dagger}$, we can construct many semantically different versions

$[[\ ]]_1^{\dagger}, \cdots [[\ ]]_i^{\dagger}, \cdots : B_{P_L^{\dagger}}\rightarrow B$

such that $[[\ ]]_i^{\dagger} \mid B_{P_L}=[[\ ]]$ for $1\leqq i\leqq \cdots$.

In other words, the corresponding B-value bi for each $\dagger_i$ may be changed under the same syntax $P_L^{\dagger}$. ⊣

By iv) and v), we notice that

$S_{P_L^{\dagger}}([[\ ]]^{\dagger}, \sim)\cap B_{P_L}=S_{P_L}([[\ ]], \sim)$ for any $[[\ ]]\in B^{B_P}$ and $\sim$.

In this sense, each $\dagger_i$ works like the (two-valued) predicate "true" in Prolog, rather than a system predicate like, say, "$\leqq$". On the other hand, since the existence of each $\dagger_i$ essentially influence the answer gathering process for each fixed (μ_F, Φ),

the character may be said to be more similar to the cut "!". The point is that the effect of our dagger symbol is not uniquely determined by the system but is affected by the corresponnding B-values $b_1, \cdots, b_n$.

So far, we have been explaining the proper features which our L-fuzzy inferential system has. Since we have extended the notion of Horn clause program by inserting dagger symbols in a pure logic programming level, it is necessary to check that the extension is still of worth called "logic program." In the next section, we investigate the logical background within the framework of LIFE-III by applying the result of chapter II. From now on, let's call the above defined B-valued fuzzy inferential system as "F-LIFE (Fuzzily (but) Logic-oriented Inferential Framework Extension)" for the sake of simplicity.

§5-2. Theoretical Background of F-LIFE

In this section, we apply the result of chapter II to F-LIFE to establish the logical background as an inferential system. As is stated in §5-1, the basic programming style of F-LIFE is an extended form of Horn clause program by introducing dagger operators which control the procedural semantics via Φ by inserting auxiliary B-manipulations. Thus introduced dagger operators are not logical predicates in their original sense. On the other hand, since each dagger is assigned a B-value, we might be able to say that it is a ground atom in the sense that "a ground atom is a primitive formal expression based on a language which is uniquely assigned a certain truth value."

How should we dissolve this apparent dilemma concerning dagger operators in F-LIFE. For the solution, sticking to logic from the beginning to the end, let's employ the following formalism.

Let L be a language and B be a complete Boolean algebra with which we want to make up a *F*-program $(P_L, [[\ ]])$, which is expected to formally express our intended knowledge. Now, we consider that the daggers $\{\dagger_1, \cdots, \dagger_n\}$ is a set of new 0-ary predicate symbols, where the number n is implicitly determined by P_L.

Thus, $\dagger_i$ is an atomic proposition in the sense of propositional logic for each $1 \leqq i \leqq n$. Here, expand L to $L^\dagger = L \cup \{\dagger_1, \cdots, \dagger_n\}$ and extend P_L to $P_L{}^\dagger$ so that $P_L{}^\dagger$ is the program constructed from P_L by inserting $\dagger_i$ in a suitable place of (possibly empty) bodi(es) in appropriate clause(s) in P_L, in addition to assertions of the form

$\dagger_i \leftarrow$

for $1 \leqq i \leqq n$.

Thus, $P_L{}^\dagger$ consists of the following five different types of clauses.

1. $A \leftarrow$ (pure assertion)
2. $A \leftarrow \dagger_i$ (daggered assertion)
3. $A \leftarrow B_1, \cdots, B_K$ (pure clause)
4. $A \leftarrow \dagger_{i_1}, B_1, \dagger_{i_2}, B_2, \cdots, \dagger_{i_K}, B_K, \dagger_{i_{K+1}}$ (daggered clause, where $\dagger_{i_j}$ may be omitted for some $1 \leqq j \leqq K+1$.)
5. $\dagger_i \leftarrow$ (dagger)

Then, we can obtain many extentions

$[[\ \ ]]^\dagger : B_{P_L{}^\dagger} \rightarrow B$

for the same $[[\ \ ]] : B_{P_L} \rightarrow B$ by just determining each B-value $[[\dagger_i]]^\dagger$ for $1 \leqq i \leqq n$. This is the formalism we employ to dissolve the dilemma and to embody the operational function concerning † discussed in the previous section all in one. The key idea of the above formalism is that B-value of $\dagger_i$ is not constant for the same P_L, though $[[\dagger_i]]^\dagger$ is uniquely determined once $[[\ \ ]]^\dagger$ is fixed. By the construction, it is easy to see that

$S_{P_L{}^\dagger}([[\ \ ]]^\dagger, \sim) = S_{P_L}([[\ \ ]], \sim) \cup \{\dagger_1, \cdots, \dagger_n\}$

where $\sim$ over $A(\Pi_{P_L{}^\dagger}, V)$ is the trivial extension of $\sim$ over $A(\Pi_{P_L}, V)$ such that

$\dagger_i \sim \dagger_j$ iff $\dagger_i = \dagger_j$ for $1 \leqq i, j \leqq n$.

Now, since each original clause $A \leftarrow B_1, \cdots, B_K$ in P_L satisfies

$([[A]] \leftarrow [[B_1 \wedge \cdots \wedge B_K]] \in F$,

$([[A]]^\dagger \leftarrow [[B_1 \wedge \cdots \wedge B_K]]^\dagger \wedge [[\dagger_{i_1}]]^\dagger \wedge \cdots \wedge [[\dagger_{i_j}]]^\dagger \in F$

holds for any subset $\{\dagger_{i_1}, \cdots, \dagger_{i_j}\}$ of $\{\dagger_1, \cdots, \dagger_n\}$.

By the commutativity of Boolean operation $\wedge$, this means that, for any extended clause $C^{\dagger}$ of type 2 or 4,

$$[[C^{\dagger}]]^{\dagger} \in F.$$

Moreover, since the aim of our introducing "†" is to affect the procedural semantics based on Φ and Φ has the domain F by the definition, without loss of generality, we can assume that

$$[[\dagger_i]]^{\dagger} \in F$$

except at most one dagger, say, $\dagger_1$. The expected aim of $\dagger_1$ is to exclude any extended clause $C^{\dagger}(\dagger_1)$ containing $\dagger_1$ from the positive collection w.r.t. Φ. In other words, any refutation R which use $C^{\dagger}(\dagger_1)$ as an input clause always satisfies the condition

$$\upsilon(R) \notin F$$

and so $\upsilon(R)$ trivially satisfies the condition

$\Phi(\upsilon(R))$ does not hold.

For this purpose, we may choose any B-value b_1 such that

$b_1 \notin F$.

Let b_1 be such that there is a complete filter $F' \supset F$ such that

$b_1 \in F'$.

(We assume that F is not ultra.)

Then, the extended program $(P_L^{\dagger}, [[\ \ ]]^{\dagger})$ becomes a F'-program by the definition.

As a query for $(P_L^{\dagger}, [[\ \ ]]^{\dagger})$, we also permit an extended form

$$\leftarrow \dagger_{i_1}, A_1, \dagger_{i_2}, \cdots, \dagger_{i_m}, A_m, \dagger_{i_{m+1}}$$

where $\dagger_{i_j}$ may be omitted for some $1 \leqq j \leqq m+1$.

Thus, we take $(P_L^{\dagger}, [[\ \ ]]^{\dagger})$ as the starting B-valued F'-program based on the language $L^{\dagger}$ to apply the result of chapter II, §2-3. In this situation, let's specify $\Delta = \{\{\dagger_1\}, \cdots, \{\dagger_n\}\}$ and $\Lambda = \{b_1, \cdots, b_n\}$ where $[[\dagger_i]]^{\dagger} = b_i$ for $1 \leqq i \leqq n$,

and consider $E_{P_L}{}^{\dagger}(\Delta, \Lambda)$. Since each $W_i = \{\dagger_i\}$ is a singleton, we notice that Δ satisfies B-valued unification condition w.r.t. $\sim$. Moreover, since $\{b_1, \cdots, b_n\} \subseteq F'$, we recognize that $E_{P_L}{}^{\dagger}(\Delta, \Lambda)$ satisfies $Tp(\Delta, \Lambda)$-consistent property and Λ trivially

satisfies $F^{\prime}$-consistent property. Finally, by the above construction, each $\dagger_i$ itself is the eliminator of $\{\dagger_i\}$ for $1 \leqq i \leqq n$. So, we can apply the result of Theorem 2-3-20 to obtain

$$Sp_{L^{\dagger}}(\mu_{\Delta,\Lambda}, \sim) = B\mu_{\Delta,\Lambda}[F^{\prime}]$$

where $[\mu_{\Delta,\Lambda}]$ is the least element of $Ep_{L^{\dagger}}(\Delta, \Lambda) \cap Mp_{L^{\dagger}}(B, F^{\prime})$.

Though the above is of worth being called the " theoretical background" of F-LIFE, the result in this form of completeness has substantial meaning only when we use restricted kind of B-unification. Under a general situation where we essentially use a proper character of B-unification, we ought to consider

$$Ep_{L^{\dagger}}(\Delta, \Lambda) \cap Mp_{L^{\dagger}}(B, F^{\prime}) \cap \Gamma p_{L^{\dagger}}(J)$$

where $\Gamma p_{L^{\dagger}}(J)$ is a complete sublattice of $Pp_{L^{\dagger}}$ constructed by using the notion of "J-faithfulness." By adjusting J, we can obtain general $\equiv$-unification in addition to (Δ, Λ)-absoluteness on the same syntactical program $P_{L^{\dagger}}$, where $\equiv$ is a substitution transitive equivalence relation over $A(\Pi p_{L^{\dagger}}, V)$. In the following, by applying the general result proved in chapter II, §2-4 to F-LIFE, we will review the semantic character of $\dagger$ operators again.

Now, let P_{L} and $P_{L^{\dagger}}$ be as above. Applying the general result proved in chapter II, §2-2, we notice that

$$Sp(\mu_J, \sim) = B\mu_J[F], \quad \cdots (\star)$$

where $[\mu J]$ is the least element of $\Gamma p_L(J) \cap Mp_L(B, F)$. On the other hand, by applying the result proved in chapter II, §2-4, we get

$$Sp_{L^{\dagger}}(\mu_{\Delta,\Lambda,J^{\dagger}}) = B\mu_{\Delta,\Lambda,J^{\dagger}}[F^{\prime}], \quad \cdots (\triangle)$$

where $[\mu_{\Delta,\Lambda,J^{\dagger}}]$ is the least element of

$$Ep_{L^{\dagger}}(\Delta, \Lambda) \cap \Gamma p_{L^{\dagger}}(J^{\dagger}) \cap Mp_{L^{\dagger}}(B, F^{\prime})$$

and (Δ, Λ) is specified as above by daggers and $J^{\dagger}$ is the natural extension of J which is consistent with $\{\{\dagger_1\}, \cdots, \{\dagger_n\}\}$.

In the following, we show that

$$[\mu_{\Delta,\Lambda,J^{\dagger}}]Bp_{L} = [\mu_J]. \quad \cdots (\bigcirc)$$

In other words, let $\mu_{J^{\dagger}}$ be the natural extension of μ_J to $BPL^{\dagger}$ such that

$\mu_J{}^{\dagger}(\dagger_i) = \mathbf{b}_i$ for $1 \leqq i \leqq n$

and

$\mu_J{}^{\dagger}(\sigma) = \mu_J(\sigma)$ for $\sigma \in Bp_L$.

Then, $[\mu_J{}^{\dagger}] = [\mu_{\Delta,\Lambda,J}{}^{\dagger}]$. Thus, we recognize the fact that, in order to obtain $[\mu_{\Delta,\Lambda,J}{}^{\dagger}]$, all we need is to construct $[\mu_J]$ in $\Gamma p(J) \cap Mp_L(\mathbf{B}, F)$.

Now, to check

Theorem 5-2-1. $[\mu_{\Delta,\Lambda,J}{}^{\dagger} | Bp_L] = [\mu_J]$

Proof:

It is enough to see the following two directions.

i) $[\mu_{\Delta,\Lambda,J}{}^{\dagger} | Bp_L] \leqslant_F [\mu_J]$

Let $\mu_J{}^{\dagger}$ be the natural extension of μ_J to $Bp_L{}^{\dagger}$.

Then, we notice that

$[\mu_J{}^{\dagger}] \in Ep_L{}^{\dagger}(\Delta, \Lambda) \cap \Gamma p_L{}^{\dagger}(J^{\dagger}) \cap Mp_L{}^{\dagger}(\mathbf{B}, F^{\dagger})$.

Here, since $[\mu_{\Delta,\Lambda,J}{}^{\dagger}]$ is the least element of

$Ep_L{}^{\dagger}(\Delta, \Lambda) \cap \Gamma p_L{}^{\dagger}(J^{\dagger}) \cap Mp_L{}^{\dagger}(\mathbf{B}, F^{\dagger})$,

we obtain

$[\mu_{\Delta,\Lambda,J}{}^{\dagger}] \leqslant_F [\mu_J{}^{\dagger}]$ in $\mathbf{B}^{Bp_L{}^{\dagger}}/F^{\dagger}$.

So, $[\mu_{\Delta,\Lambda,J}{}^{\dagger} | Bp_L] \leqslant_F [\mu_J]$ in $\mathbf{B}^{Bp_L{}^{\dagger}}/F$

because $[\mu_{\Lambda,\Lambda,J}{}^{\dagger}(\dagger_i) = \mu_J{}^{\dagger}(\dagger_i) = \mathbf{b}_i$ for $1 \leqq i \leqq n$.

ii). $[\mu_J] \leqslant_F [\mu_{\Delta,\Lambda,J}{}^{\dagger} | Bp_L]$.

Since $[\mu_J]$ is the least element of $\Gamma p_L(J) \cap Mp_L(\mathbf{B}, F)$ and $[\mu_{\Delta,\Lambda,J}{}^{\dagger} | Bp_L]$ trivially satisfies

$[\mu_{\Delta,\Lambda,J}{}^{\dagger} | Bp_L] \in \Gamma p_L(J)$,

it is enough to check $[\mu_{\Delta,\Lambda,J}{}^{\dagger} | Bp_L] \in Mp_L{}^{\dagger}(\mathbf{B}, F)$. ··· $(\diamond)$

($(\diamond)$ is not so trivial, because

$\mu_{\Delta,\Lambda,J}{}^{\dagger}(C^{\dagger}) \in F^{\dagger} \Rightarrow \mu_{\Delta,\Lambda,J}{}^{\dagger} | Bp_L(C) \in F$

is not obvious as the following simple example illustrates.

Example. Let $\mathbf{B} = \{1, \mathbf{0}, \mathbf{b}_1, \neg \mathbf{b}_1\}$, $F = \{1\}$, $F^{\dagger} = \{1, \mathbf{b}_1\}$ and

$C^{\dagger} = \{p \leftarrow \dagger_1, q\}$, $[[p]] = \mathbf{0}$, $[[q]] = \neg \mathbf{b}_1$, $[[\dagger_1]] = \mathbf{b}_1$.

Then, $[[C^\dagger]] = (0 \leftarrow \neg b_1 \wedge b_1) = (0 \leftarrow 0) = 1 \in F'$.

However, $[[C]] = (0 \leftarrow \neg b_1) = b_1 \in F'$ but $b_1 \notin F$. ⊣

Of course, in this case, th deficiency comes from the fact that [[ ]] is not the least.)

Here, from the fixed-point construction of $[\mu_{\Delta,\wedge,J^\dagger}]$, since $\{b_1, \cdots, b_n\} \subset F'$, we notice that we can choose a representative μ_2 of $[\mu_{\Delta,\wedge,J^\dagger}]$ modulo F' such that

$(\forall \sigma \in Bp_{L^\dagger})\ (\mu_2(\sigma) = 1 \text{ or } 0)$.

So, especially, for any $C^\dagger \in P_{L^\dagger}$, $\mu_2(C^\dagger) = 1$.

Now, consider $\mu_2 \lceil Bp_{L^\dagger}$. Then, we notice

$\mu_2 \lceil Bp_{L^\dagger}(C) \in F$ for any clause C in $P_{L^\dagger}$.

The reason is the following:

⌈ Suppose there is a $C \in P_{L^\dagger}$ such that $\mu_2 \lceil Bp_{L^\dagger}(C) \notin F$. Let $C^\dagger$ be the corresponding daggered extension in $P_{L^\dagger}$.

Then,

$\mu_2(C^\dagger) = \mu_2(C^+) \leftarrow (\mu_2(C^-) \wedge \mu_2(\dagger_{i_1} \wedge \cdots \wedge \dagger_{i_K}))$ for suitable daggers $\dagger_{i_1}, \cdots, \dagger_{i_K}$

$= \mu_2(C^+) \leftarrow (\mu_2(C^-) \wedge 1$, by the choice of μ_2 and

the fact $\{\dagger_{i_1}, \cdots \dagger_{i_K}) \subseteq F'$

$= \mu_2(C^+) \leftarrow (\mu_2(C^-)$

$= \mu_2(C) \notin F$.

This contradicts the fact $\mu_2(C^\dagger) = 1$. ⊣⌋

Since $\lceil \mu_2 \lceil Bp_{L^\dagger}] = [\mu_{\Delta,\wedge,J^\dagger} \lceil Bp_{L^\dagger}]$ in $B^{Bp_{L^\dagger}}/F$,

We obtain

$\mu_{\Delta,\wedge,J^\dagger} \lceil Bp_{L^\dagger}(C) \in F$ for any clause C in $P_{L^\dagger}$.

$\therefore$ $[\mu_{\Delta,\wedge,J^\dagger}] Bp_{L^\dagger} \in Mp_{L^\dagger}(B, F)$. □

By (☆) + (△) + (○), we obtain

$Sp_{L^\dagger}(\mu_J, \sim) \cup \{\dagger_1, \cdots, \dagger_n\}$

$= B\mu_J[F] \cup \{\dagger_1, \cdots, \dagger_n\}$ by (☆)

$= B\mu_{\Delta,\wedge,J^\dagger} | Bp_{L^\dagger}[F] \cup \{\dagger_1, \cdots, \dagger_n\}$ by (○)

$= B\mu_{\Delta,\wedge,J^\dagger}[F']$

$= Sp_L^{\dagger}(\mu_{\Delta, \wedge, J^{\dagger}}, \sim)$ by ($\triangle$)

This is what our intuition concerning F-LIFE requires.

Here, before closing the section, let's briefly check the effects of the order of a dagger operator appeared in a clause. For simplicity, compare two clauses of the form

$$p \leftarrow \dagger_1, q \qquad \cdots (1)$$

and

$$p \leftarrow q, \dagger_1 . \qquad \cdots (2)$$

By the commutativite of $\wedge$, the declarative semantics of both should be the same. At the same time, the global procedural semantics as a whole are also the same in the sense that, if (1) is used during a refutation (R, θ) of a goal, then (2) can also be used to produce the same answer substitution θ and the same B-values $v(R)$, $v(\theta)$. However, it we employ Φ as a tool of intelligent backtracking, the effects of (1) and (2) become different and thus affect the local procedural semantics. In a practical phase, if we employ, as usual, left to right goal-selection strategy as a computation rule, a B-value estimation w.r.t. $\dagger_1$ is done firstly for (1). So, in such a case that $[[\dagger_1]]^{\dagger}$ (without $[[q]]^{\dagger}$) already witnesses the unsuitability of this branch w.r.t.Φ, using (1) is more efficient than usig (2). On the other hand, for (2), B-valued unification w.r.t. q is done firstly. So, in such a case that q fails to be B-valued unified with the required clause in $P_L^{\dagger}$ but $[[\dagger_i]]^{\dagger}$ does pass through the sieve by Φ, using (2) is more efficient than using (1).

By the way, can we forsee which is more suitable w.r.t. efficiency for a candidate to obtain an extension P_L^*. The uniform prediction is, of course, impossible, because the effect heavily depends on the choice of (Φ, μ_P). However, for example, concerning $\dagger_1$ whose corresponding B-value b_1 is living outside of F, we had better always insert $\dagger_1$ at the top (leftmost) of a body of a targetted clause. The reason is obvious. The purpose of our inserting $\dagger_1$ is to mark such clauses

which we don't want to use during the expected refutation procedure. In anyway, the choice of ordering w.r.t. daggers is ours for God's sake.

Finally, a word of caution. It is no business of this kind of theoretical paper to ask whether the inserting of daggers is done dynamically (depending on each query) or not. The dynamical marking may be technically possible and may have some sense in some cases. On the other hand, it may be impossible or have no importance in other cases in which every dagger is marked statically depending only on (Φ, μ_F).

Chapter Ⅵ Appendix
Qualitative Deduction, Universal Unification and Non-monotonic Reasoning

§6-0. Introduction

Let P be a definite clause program. Let $A(\Pi_P, V)$ and $T(\Sigma_P, V)$ be the set of all atoms in P and the set of all terms in P whose variables appear in V respectively. Let $\equiv$ be an equivalence relation over $A(\Pi_P, V)$. Then,

i) $\equiv$ is "substitution transitive" iff

$(\forall A, B \in A(\Pi_P, V))\ (\forall \theta$: substitution over $T(\Sigma_P, V))\ (A \equiv B \rightarrow A\theta \equiv B\theta)$.

ii) Let $\equiv$ be a substitution transitive equivalence relation over $A(\Pi_P, V)$. Then, for any $A, B \in A(\Pi_P, V)$, θ is a $\equiv$-unifier for A and B iff $A\theta \equiv B\theta$. ⊣

Obviously, the above definition is a generalized version of the conventional notion of universal unification based on equational theory over $T(\Sigma_P, V)$. In [4], we explained a few merits of the generalization and proposed a few notions which belong to the generalized version. Among them are those of "two-lane unification" and "a.b.f.e. unification (all but finite exception unification)". Both are related to certain kinds of heuristics we use at the stage of knowledge acquisition. The purpose of this paper is to propose a simple example of logic program, the procedural semantics of which can be interpreted from a viewpoint of not only two-lane unification but also a.b.f.e.-unification as the inference steps.

In §6-1, we designate a logic program P_D and prepare the stuff which can possibly appoint an equivalence relation over U_{P_D}, the Herbrand universe of P_D. In this phase, the effects of P_D is interpreted as the normal program based on syntactical unification, and we need a system predicate or other appropriate methods to obtain the expected results. Here, the idea of constructing an

equivalence relation over U_{P_D} may already be interesting from an aspect of qualitative deduction used in expert systems etc. §6-2 is devoted to reinterprete the effects of P_D from an angle of two-lane unification. The characteristics of this facet is its irregularity as a universal unification. The cause comes from the demand-driven property used at gathering answers for certain kinds of queries. This irregularity is the partial reason why we call this universal unification "two-lane". In §6-3, we review the same program from the procedural semantics of a.b.f.e.-unification. By extracting exceptional data from normal ones, we gain an efficiency of the algorithm to enumerate expected data. At the same time, we can refine the answer gathering processes concerning given data. The refinement of processes can be explained using the terminology of non-monotonic reasoning. Since the procedural semantics of a.b.f.e.-unification in this example is a little complex, the logical background based on Boolean-valued unification is discussed in the next §6-4. Finally, in §6-5, we give a few remarks about the results in this paper and the related works.

§6-1. A preliminary example of qualitative deduction

In this section, we present an example which will be helpful to understand the maintopics. Before giving the example, let's consider the background. In general, given a set of vast data, there are a lot of cases that we want to grasp the quality of inclination as a whole instead of considering particular property of each datum. One method which is available in this situation may be the following. Firstly, devide the data area into a class of unit sets and choose a representative from each unit. Then, ask something about the set of representatives in order to comprehend the tendency of all data. Lastly, if necessary, enumerate elements in the special units which pass the test question.

Here, as a very simple and special case, let's take a set D of data, each of which has the symbolical form

$$<i, j>$$

where $0 \leqq i, j \leqq 9$, with dot function $\cdot(\ )$ possibly applied only to the second argument j. Our intention is that $<1, 2>$ means, for example, the real number 1.2 . The dot function '$\cdot$' expresses a restricted form of circular decimals. Thus, $<1, \cdot(2)>$ stands for 1.22···. The necessity of our employing '$\cdot$' should be obvious by just imaging a real number r which satisfies a simple condition, say, $3 \times r = 10$.

Remark: Don't confuse this expressional notation as an element of data and the semantics of each expression. Though both $<0, \cdot(9)>$ and $<1, 0>$ denote the same number 1, they are different as elements in D. ⊣

In this example, let's take each interval $[n, n+1)$ for $0 \leqq n \leqq 9$ as the boundary for unit sets. Let ξ be a representative function whose value becomes a representative for each $[n, n+1)$. The representative may be a typical element from each unit set or may be a suitable entity which characterize each unit set. Here, the point is that, in order to decide ξ, we don't usually assign a value for each unit set one by one where the number of unit sets is still large. In other

words, there ought to exist (or find by somehow or other) a natural characteristics which can define the representative function ξ algorithmically.

Note: To tell the truth, it is this characteristics that determine the suitable class of unit sets. ⊣

In the following, let's define ξ by

$(\forall d \in D)(d \in [n, n+1) \rightarrow \xi(d) = n)$ for $0 \leqq n \leqq 10$.

To be more precise, for any datum $<i, j>$,

$$\xi(<i,j>) = \begin{cases} i+1 & , \quad \text{if } j = \cdot(9) \\ i & , \quad \text{otherwise.} \end{cases}$$

On the ground of this definition of ξ, our purpose becomes to ask a question concerning $\{\xi(d) \mid d \in D\}$ and to find out each set $\{d \in D \mid \xi(d) = n\}$, for which the representative n is an answer of the question.

Now, in order to manage the above example within a (Prolog-like) logic program, we assume first of all that there is a program P_N which covers sufficient portion of number theory (between 0 and 10.) In the following, without loss of generality, we can assume that expressions like

sum (X, 1, N), eq(X, N)

are all restricted to natural numbers (with additional type assignment Nat(X) etc).

Next, to treat the information about D, we add the following clauses to P_N.

$$\left.\begin{array}{l} D = [d_1, \\ \quad \vdots \\ \quad \vdots \\ \quad d_k] \end{array}\right\} \quad \text{(Database)} \qquad \cdots A_D$$

(Each d_n has the form of expression $<i, j>$ in D.)

$$\left.\begin{array}{l} r_\xi(<X, \cdot(9)>, N) \leftarrow \text{member}(<X, \cdot(9)>, D),\ \text{sum}(X, 1, N) \\ \\ r_\xi(<X, Y>, N) \leftarrow Y \neq \cdot(9),\ \text{member}(<X, Y>, D),\ \text{eq}(X, N) \end{array}\right\} \quad \cdots B_D$$

(The meaning of r_ξ should be obvious from the above argument. Since the definition of ξ has a typical if-then-else form, this part B_D can be rewritten in the following more practical form by using data-type checking equipment $\in_D$ and if-then-else predicate.

$$r_\xi(X \in_D D, N) \leftarrow \text{if-then-else}(\lambda_2(X) = \cdot(9), \text{sum}(\lambda_1(X), 1, N), \text{eq}(\lambda_1(X), N))$$

where functions $\lambda_1(X)$ and $\lambda_2(X)$ pick up the first and the second arguments of X respectively. Throughout this paper, our purpose is not to write a precise program but to give a rough idea how we use a definite clause to express the required concept.)

Appropriate clauses containing predicates which are concerned with elements of D. $\cdots C_D$

In anyway, the Herbrand universe is enlarged by the constructor $<,>$ and the dot function symbol '$\cdot$'. Let $P_D = P_N \cup A_D \cup B_D \cup C_D$. If we want to ask a question concerning the relation between an element of D and a property which talks something about the tendency of D, the required query for P_D might has the form

$$\leftarrow r_\xi(X, Y), \psi(Y)$$

where ψ is a predicate over $\{0, 1, \cdots, 10\}$, the set of representatives of D. Here, an answer substitution for this query becomes, say

$$\{X \leftarrow d, Y \leftarrow n\}$$

for each $d \in D$ and $n \in Nat$ such that $\xi(d) = n \wedge \psi(n)$. However, there often happens the case that we want to enumerate all elements of a unit whose representative satisfies a certain property. There may be many methods to treat this kind of case. One naive method using programming technique only is to define a predicate $\varepsilon_D(X, Y)$ such that $Y = [X]_{\equiv_D}$, where $[\quad]_{\equiv_D}$ is the equivalence class over $D \cup [0, \cdots, 10]$ such that, for any $\tau \in D \cup [0, \cdots, 10]$.

$$[\tau]_{\equiv_D} = \begin{cases} \{ d \in D \mid \xi(d) = \xi(\tau) \} \cup \{n\}, & \text{if } \tau \in D \text{ and } \xi(\tau) = n \\ \{ d \in D \mid \xi(d) = \tau \} \cup \{\tau\}, & \text{otherwise.} \end{cases} \quad \cdots(1)$$

For this purpose, we need to define the (abstract) set expression

$$\{ X \in D \mid r_\xi(X, N) \}.$$

Though some system predicates can really do this task, the tax of the efficiency is not so cheap at least compared with the programming technique discussed in §6-3. (Check the definition of the predicate r_ξ! It is a heavy task to execute member (X, D)). In the next section, we see the fact that we can obtain a more useful tool than $\varepsilon_D(X, Y)$ by using a demand-driven command from a viewpoint of universal unification.

§6-2. Two-lane unification

Let P_D be as in the previous section. Let $\equiv_D$ be the equivalence relation over $D \cup [0, \cdots, 10]$ implicitly defined by (1). By definition, $\equiv_D$ substantially works only on

$$D \cup \{n \in [0, \cdots, 10] \mid (\exists d \in D)(n = \xi(d))\}$$

Consider $(P_D, \equiv_D)$. (Precisely speaking, we should extend $\equiv_D$ to an equivalence relation over $A(\Pi P_D, V)$. The detail will be discussed later in this section.) We can regard $(P_D, \equiv_D)$ as a program P_D based on $\equiv_D$-unification in the following sense. Let $\psi(X)$ be a predicate concerning natural numbers. By the query

$$\leftarrow \psi(X)$$

w.r.t. P_D based on syntactical unification, we can obtain a sequence $n_1, \cdots, n_k$ of natural numbers which satisfy $\psi(X)$ in such a manner as, say,

$X = n_1$;

$X = n_2$;

:

$X = n_k$.

Here, between an answer substitution "$X = n_i$" ($1 \leqq i \leqq k-1$) and ";", we may interpolate system command

"D- enumeration"

to obtain the set $\{d_{i_1}, \cdots, d_{i_m}\}$ of data satisfying

$$\{d_{i_1}, \cdots, d_{i_m}\} = \{d \in D \mid d \equiv_D n_i\}.$$

Then, the resulting answer features become something like

$X = n_1$, D-enumeration ?

yes, $\{d_{1_1}, \cdots, d_{1_k}\}$;

$X = n_2$, D-enumeration ?

no ;

$X = n_3$, D-enumeration ?

:

where each response "yes" or "no" is obtained via user interface. The algorithm concerning D-enumeration may not be directly connected with the original predicate r_ξ anymore, though it is certainly related to the representative function ξ semantically. Looking from a viewpoint of universal unification, we recognize that this kind of $\equiv_D$-unification is irregular. The irregularity owes to the character that we basically employ syntactical unification and add the device of $\equiv_D$-unification by demand. In other words, we mix two different kinds of unifications in the same program P_D. The distinction of the two is done by user interface. In this sense, we may name the above kind of irregular universal unification "two-lane." Thus, we can define;

Definition 6-2-1. Let P be a program and $\equiv$ be a substitution transitive equivalence relation over $A(\Pi_P, V)$. Then, an (irregular) universal unification based on $\equiv$ is a "two-lane unification" iff,

for any goal $\leftarrow \Phi(X)$ for P,

i) Use syntactical unification to obtain a ground answer a_1 for X.

ii) If necessary by demand, enumerate other possible answers $\{\sigma \in U_P \mid a_1 \equiv \sigma\}$.

Else, omit this second process.

iii) Repeat the above process w.r.t. the other answer substitutions

$X \leftarrow a_2, \cdots, X \leftarrow a_k$

(based on syntactical unification) one after another. ⊣

This is an applicational form of the notion of "two-lane unification" in general. The word "two-lane" originates the separation of syntactical unification part from $\equiv$-unification.

Let's return to our special example. As we remarked before, we should extend $\equiv_D$ to an equivalence relation $\equiv_A$ over $A(\Pi_{P_D}, V)$ in order to define the notion of universal unification legitimately. Here, we can easily extend $\equiv_D$ to the corresponding relation $\equiv_D$ over U_{P_D}. However, we must not unconsciously

extend $\equiv_D$ (over U_{P_D}) to the corresponding relation $\equiv_B$ over B_{P_D} because of the qualitative difference between raw data and their representatives. For example, there may exist predicates ψ_1, ψ_2 such that, for a natural number n and a datum $d \in D$ which satisfy $n \equiv_D d$,

$\psi_1(n)$ holds but there is no necessity to hold $\psi_1(d)$ or simply $\psi_1(d)$ does not hold and

$\psi_2(d)$ holds but there is no necessity to hold $\psi_2(n)$ or simply $\psi_2(n)$ does not hold.

In anyway, there might be predicates $\psi_1, \cdots, \psi_k$ which satisfy

$n_i \equiv_D d_i$ but $\psi_i(n_i) \not\equiv_B \psi_i(d_i)$ for $1 \leqq i \leqq k$.

For this kind of predicate ψ_i, we can't naturally extend the relation $\equiv_D$ to the corresponding relation over B_{P_D}.

Example 6-2-2. Suppose, for simplicity, that

$\{d \in D \mid d \in [0, 2)\} = \{<0, 2>, <0, 4>, <0, \cdot(5)>, <0, \cdot(9)>, <1, 0>, <1, \cdot(3)>, <1, 8>\}$. Then,

$[0]_{\equiv_D} = \{<0, 2> <0, 4>, <0, \cdot(5)>\} \cup \{0\}$

and

$[1]_{\equiv_D} = \{<0, \cdot(9)> <1, 0>, <1, \cdot(3)>, <1, 8>\} \cup \{1\}$.

Let $\psi_1 = \mathrm{odd}(X)$ be the predicate already defined in P_N which states "X is an odd number." Obviously, we hope that

odd $(<0, \cdot(9)>)$ and odd $(<1, 0>)$ hold,

but we expect that neither odd $(<1, \cdot(3)>)$ nor odd $(<1, 8>)$ holds.

This typically shows the situation that

$1 \equiv_D <1, \cdot(3)>$ but odd $(1) \not\equiv_B$ odd $(<1, \cdot(3)>)$.

(The above effects can be implemented by, say, simply adding the clauses

odd $(X) \leftarrow X \in \mathrm{type}(D)$, $\lambda_2(X) = \cdot(9)$, even $(\lambda_1(X))$

$\cdots (2)$

odd $(X) \leftarrow X \in \mathrm{type}(D)$, $\lambda_2(X) = 0$, odd $(\lambda_1(X))$

to C_D. Here, as before, the clauses can be hidden by using system command via user interface.) ⊣

Of course, there might be predicates $\Phi_1, \cdots, \Phi_l$ which naively satisfy that, for any $n \in Nat$ and $d \in D$,

$$n \equiv_D d \quad \rightarrow \quad \Phi_i(n) \equiv_B \Phi_i(d) \quad \text{for} \quad 1 \leqq i \leqq l.$$

Example 6-2-3. Let $[0]_{\equiv_D}$ and $[1]_{\equiv_D}$ be as in Example 2-2.

Let $\Phi_1 = <(1)\,(X)$ be the predicate already defined in P_N which states "X is a number which is smaller than 1." ($<(1)(X)$ is usually written as "$X < 1$".)

Then, for any element o in $[0]_{\equiv_D}$, we hope that

$<(1)\,(\sigma)$ holds.

At the same time, for any element τ in $[1]_{\equiv_D}$, we expect that

$<(1)\,(\tau)$ does not hold.

This means that, for any $n \in \{0, 1\}$ and $d \in [0, 2) \cap D$,

$$n \equiv_D d \quad \rightarrow \quad \Phi_i(n) \equiv_B \Phi_i(d).$$

(The above effects can be implemented by, say, simply adding

$$<(1)\,(X) \quad \leftarrow \quad X \in type\,(D), r_\xi(X, Y), <(1)\,(Y) \qquad \cdots (3)$$

to C_D. Again, the clause can be hidden by using some other technique.)

Note: Remember that we are primarily interested in positive informations by closed world assumption. In other words, what we want to check is

$$n \equiv_D d \quad \rightarrow \quad \Phi_i(n) \equiv_B \Phi_i(d) \quad \text{or not}$$

for such $\Phi_i(n)$ that holds in our intended model, i.e., in P_N. ⊣

In short, $\equiv_B$ becomes a subset of the natural extention of $\equiv_D$. This is the place where another kind of irregularity of two-lane unification appears. However, these irregularities become merits of two-lane unification. Applying usual system predicate, say, "a-bag-of" to Ψ_1 or employing a certain parallel processing device, we do obtain a set $[n_1, \cdots, n_k]$ of answers w.r.t. Ψ_1 once for all.

Moreover, as is pointed in the previous section, we may gather elements in a unit set whose representative is n_i by using r_ξ (X, n_i) for each $1 \leqq i \leqq k$. Contrastingly, the advantages employing the technique of two-lane unification comes from the following three ideas.

1. The separation of main (prototypical) program P_N and the module concerning D. By doing so, we can easily replace D to the other database or can modify the unit sets for the fixed D or can even change the representative function for the fixed unit sets. These alternations do not affect the semantical attitude of P_N in its essential level. In addition, if we prepare a big program P from the beginning, we may parallelly process a variety of data sets $D_1, \cdots, D_n$, where data types of D_i and those of D_j are different for $i \neq j$.
2. The selectiveness of demand-driven property of D-enumeration via user interface. Usual enumerating function is not selective. The all-or-nothing character without user interface often outputs unnecessary information concerning D as a whole. It sometimes happens the case that we are not interested in data in a unit set whose representative satisfies a certain semantical condition. The point is we can do the selection during answer gathering process w.r.t. Ψ_1 without asking a query like $\leftarrow r_\xi(X, n_i)$ explicitly.
3. Efficiency in case that D is huge. Considering r_ξ, the reason should be obvious. Any algorithm which suitably embodies $\equiv_D$-unificatinon (D-enumeration) is permitted.

Now, once $\equiv_B$ over BP_D is fixed, the substitutive completion of $\equiv_B$ defines the expected equivalence relation $\equiv_A$ over $A(\Pi P_D, V)$.

Under such a restricted form of $\equiv_A$, the above defined two-lane unification really becomes an irregular universal unification in general.

§6-3. A.B.F.E.-unification and non-monotonic reasoning

In the previous section, we watch a programming technique of selective enumeration from a viewpoint of irregular universal unification. Now, if we stand for the side of universal unification, we might farther be able to gain the efficiency of answer gathering (w.r.t. the same hardware) by suitably evolving the unification algorithm. Of course, the same efficiency may be obtained through a proper implement technique just by imitating the evolved unification technique, once it is revealed. The point here is, however, what kind of heuristics we should employ in order to discover a direction of evolution. This is the theme in this section (and of AI in general).

Let's review the definition of $\equiv_D$ used in $(P_D, =_A)$. $\equiv_D$ has been defined by the relation that, for any $\sigma, \tau \in D \cup [0, \cdots, 10]$

$$\sigma \equiv_D \tau \quad \text{iff} \begin{cases} \xi(\sigma) = \xi(\tau) & , \quad \text{if } \sigma, \tau \in D \\ \xi(\sigma) = \tau & , \quad \text{if } \sigma \in D \text{ and } \tau \in [0, \cdots, 10] \\ \xi(\tau) = \sigma & , \quad \text{if } \tau \in D \text{ and } \sigma \in [0, \cdots, 10] \\ \sigma = \tau & , \quad \text{otherwise.} \end{cases}$$

(Then, $\equiv_D$ is naturally extended to the corresponding equivalence relation $\equiv_D$ over U_{P_D}).

Thus, $\equiv_D$ is completely determined by ξ and ξ was defined by, for any $<i, j> \in D$,

$$\xi(<i,j>) = \begin{cases} i+1 & , \quad \text{if} \quad j = 9 \\ i & , \quad \text{otherwise.} \end{cases} \qquad \cdots(4)$$

Here, if we replace the above definition of ξ by the simpler condition that, for any $<i, j> \in D$,

$$\xi'(<i,j>) = i\,, \qquad \cdots(5)$$

what happens to the semantics of $\equiv_D$? By the original meaning of the representative, almost every element in a unit set can suitably pass the test (5)

to produce its representative except the elements having the expression <i, ·(9)> located on the boundary for each unit. That is, the "if " part in (4) is used only to manage those exceptions. The crucial point is that the set of exceptions is relatively small compared with the original data set D. (This is a motivation how the unit set is determined from a viewpoint of qualitative deduction.) With this fact in mind, consider the following new-type unification.

Let $\equiv'_D$ be the equivalence relation over $D \cup [0, \cdots, 10]$ based on ξ' as before, where we employ (5) for the definition of ξ' instead of (4). Let $\Delta \subset D$ be the finite set of exceptions w.r.t. ξ' in the above sense i.e., $\Delta = \{d \in D | \lambda_2(d) = \cdot(9)\}$. Let $\equiv''_D = (\equiv_D \cap \equiv'_D) \mid (D \cup [0, \cdots, 10] - \Delta)$. Using $\equiv''_D$ and Δ, we can define the original equivalence relation $\equiv_D$ over $D \cup [0, \cdots, 10]$ and so can define the same equivalence relation $\equiv_A$ over $A(\Pi P_D, V)$. Now, consider $(P_D, \equiv_A)$ again. Since the equivalence relation $\equiv_A$ is the same, the resulting success set w.r.t. $\equiv_A$-unification should be the same as before. But this time, the procedural semantics of $\equiv_A$-unification based on $(\equiv''_D, \Delta)$ is refined so that the second process is divided into the following steps.

ii) [1] If necessary by demand, enumerate other possible answers based on $\equiv'_D$.

[2] (If necessary by demand,) check elements in Δ so that $\equiv'_D$ is adjusted to $\equiv''_D$. As a result, at most one element in Δ is deleted from the answer list in [1].

[3] (If necessary by demand,) check elements in Δ so that $\equiv''_D$ is adjusted to $\equiv_D$. As a result, at most one element in Δ is added to the answer list in [2].

[4] Else, omit the step(s).

(A simple example will be given soon afterwords.)

What is the advantage of thus defined procedural semantics of universal unification based on $(\equiv''_D, \Delta)$. One merit concerns the efficiency of the algorithm of answer enumeration. Since the definition of ξ' based on (5) is

extremely simpler than original ξ and Δ is relatively small compared with D, the efficiency thus gained is obvious as far as we depend on the same hardware. In general, when D is (partially) ordered by a relation $\leqq_D$ and the defining algorithm of ξ consistently obeys the relation $\leqq_D$, we had better employ this kind of technique. Especially when D is huge, $\leq_D$ is simple and Δ is very small, the merit becomes tremendous. Another merit concerns the choice of expected answers. For example, if there is a case in which we can totally ignore boundary data for certain predicates, we just employ [1] and [2] as the second process for the enumeration of answers w.r.t. those predicates. Moreover, by just using step [1], we can embody fuzzy inference in a wider sense at the level of unit set. The approximation of inference thus obtained might be interesting from a viewpoint of qualitative deduction. This kind of process separation can only be inspired by the heuristics that exceptional (abnormal) data should be distinguished from the normal ones.

Putting the weight on this heuristics, we can restate the second merit using the terminology of non-monotonic reasoning in the following way. For simplicity, let's employ the example used in the previous section to argue the key points.

By definition, we recognize that

$$[0]_{\equiv'_D}=\{<0,2>, <0,4>, <0,\cdot(5)>, <0,\cdot(9)>\} \cup \{0\},$$

$$[1]_{\equiv'_D}=\{<1,0>, <1,\cdot(3)>, <1,8>\} \cup \{1\},$$

$$[0]_{\equiv''_D}=\{<0,2>, <0,4>, <0,\cdot(5)>\} \cup \{0\},$$

$$[1]_{\equiv''_D}=\{<1,0>, <1,\cdot(3)>, <1,8>\} \cup \{1\}$$

and

$$<0,\cdot(9)> \in \Delta.$$

Let's take the predicate $<(1)\,(X)$. Based on $\equiv'_D$, $<(1)\,(X)$ is implicitly realized by the simple clause

$$<(1)\,(X) \leftarrow X \in \mathrm{type}\,(D),\ <(1)\,(\lambda_1(X)) \qquad \cdots (6)$$

which is essentially the same as (3) except the condition concerning ·(9). Then, we notice that

$<(1)(<0, \cdot(9)>)$

holds at step [1].

Checking $<0, \cdot(9)> \in \Delta$ by somehow or other, we simply suspend the statement

$<(1)(<0, \cdot(9)>)$

at step [2].

Finally, by retrieving data in Δ, we recognize that no datum is to be added concerning $<(1)(X)$ any more. Thus, we experience a non-monotonic reasoning

$<(1)(<0, 2>), <(1)(<0, 4>), <(1)(<0, \cdot(5)>), <(1)(<0, \cdot(9)>), <(1)(0)$:
all hold

$\Rightarrow$

$<(1)(<0, 2>), <(1)(<0, 4>), <(1)(<0, \cdot(5)>), <(1)(0)$: hold

$<(1)(<0, \cdot(9)>)$: deleted

$\Rightarrow$

$<(1)(<0, 2>), <(1)(<0, 4>), <(1)(<0, \cdot(5)>), <(1)(0)$: hold
no addition concerning $<(1)(X)$.

Here, $<(1)(<\lambda_1(X)>)$ becomes, so to speak, default. Generally, we may say that (5) is a default rule from a viewpoint of $\equiv_A$-unification in the above sense.

We would like to name the universal unification based on $(\equiv''_D, \Delta)$ with the procedural semantics as above "a.b.f.e.-unification (all but finite exception unification)". The root of the name should be obvious from the above arguments, though this is only a simple example of more variety kinds of "a.b.f.e.-unification" based on Boolean-valued unification.

§6-4. Boolean-valued interpretation

Since the procedural semantics of the above a.b.f.e.-unification is a little more complex than usual (non-demand-driven) universal unification, it is natural for us to worry about the theoretical background as a logic programming. In this section, we secure it by showing the fact that the above is nothing but a very special case of Boolean-valued unification.

Let's start by defining the notion of Boolean-valued unification.

Definition 6-4-1. Let P be a program and B be a complete Boolean algebra. Let $[[\ ]] : B_P \to B$ be a map and $\sim$ be a substitution transitive relation over $A(\Pi_P, V)$. Then, for any $A, B \in A(\Pi_P, V)$, a substitution θ over $T(\Sigma_P, V)$ is a "Boolean-valued unifier for A and B w.r.t. $\sim$ and $[[\ ]]$" iff

$A\theta \sim B\theta$

and

$[[A\theta\rho]] = [[B\theta\rho]]$ for any ground substitution ρ for $A\theta$ and $B\theta$.

⊣

Before explaining a Boolean-valued semantics of a.b.f.e.-unification, let's review the definition of two-lane unification. The irregularity of two-lane unification has two roots. One comes from the demand-driven property of D-enumeration and another depends on the fact that there might be predicates ψ_1, ···, ψ_k which satisfy

$n_i \equiv_D d_i$ but $\psi_i(n_i) \not\equiv_B \psi_i(d_i)$ for $1 \leqq i \leqq k$.

Though the former property only superficially affects the semantics of $(P_D, \equiv_A)$, the latter fact does essentially change the semantics and this is the reason why $\equiv_B$ becomes a subset of natural extension of $\equiv_D$ to B_{P_D}.

Remark: The former is a matter of an efficiency and user's convention. For example, users can uniformly answer "yes" for the command "D-enumeration?"

at first and then they may disregard some irrelevant data in order to obtain the same effect..

However, once $\equiv_B$ is fixed, the resulting $\equiv_A$ over $A(\Pi P_D, V)$ is uniquely determined by the definition of substitutive completion. In short, $\equiv_B$ can completely decide the semantics of $(P_D, \equiv_A)$. This is the situation in the case of two-lane unification.

Now, the difference between two-lane unification and a.b.f.e.-unification in our example is located in the refinement of steps [1]~[4] in process ii). The original $\equiv_D$ is obtained through steps [1]+[2]+[3](+[4]). Here, if we choose, say, only [1](+[4]) as the step(s) in ii), the resulting equivalence relation $\equiv'_D$ becomes different from the original $\equiv_D$ and so $\equiv'_B$ over BP_D which is decided by $\equiv'_D$ and the irregularity also becomes different from $\equiv_B$. Thus, the choice of steps in ii) really alters the semantics of P_D. This alternation is expected to be reflected by the modification of map $[[\ \ ]] : BP_D \rightarrow B$, because it is the map that dynamically transforms the semantics of Boolean-valued program.

With the above observation in mind, let's interprete our example in the following Boolean-valued fashion. Let $\equiv_D$, $\equiv'_D$, $\equiv''_D$ be as before and $\equiv_B$, $\equiv'_B$, $\equiv''_B$ be the corresponding equivalence relations over BP_D. Let $\sim_D$ be the transitive closure of $(\equiv_D \cup \equiv'_D)$. The crucial property of $\sim_D$ is that, for any exceptional datum $<i, \cdot(9)>$, two unit sets $[i, i+1)$ and $[i+1, i+2)$ are stapled by the boundary element $<i, \cdot(9)>$. In other words, two equivalence classes w.r.t. $\equiv_D$ and $\equiv'_D$ whose representatives are $i+1$ and i respectively and having datum $<i, \cdot(9)>$ commonly (i.e., $\{d \in D | \xi(d) = i+1$ in (4)$\} \cup \{i+1\}$ and $\{d \in D | \xi'(d) = i$ in (5)$\} \cup \{i\}$) become one equivalence class (the stapled chain) in $\sim_D$.

Let $\sim_B$ be the natural extension of $\sim_D$ to BP_D. Then, (transitive closure of $(\equiv_B \cup \equiv'_B)) \subset \sim_B$ and each equivalence classe based on $\sim_B$ becomes the largest possible in our context.

Example 6-4-2. Take odd(X) in Example 6-2-2.

We already know that

$[odd(1)]_{\equiv_B} = \{odd(1), odd(<1, 0>), odd(<0, \cdot(9)>)\}$

and

$[odd(1)]_{\equiv'_B} = \{odd(1), odd(<1, 0>)\} = [odd(1)]_{\equiv''_B}$

and

$[odd(<0, \cdot(9)>)]_{\equiv'_B} = \{odd(<0, \cdot(9)>)\} = [odd(<0, \cdot(9)>)]_{\equiv''_B}$

Moreover, if negation as failure is available, we obtain

$[odd(0)]_{\equiv_B} = \{odd(0), odd(<0, 2>), odd(<0, 4>), odd(<0, \cdot(5)>)\}$

$= [odd(0)]_{\equiv'_B}$

$= [odd(0)]_{\equiv''_B}$

$[odd(<1, \cdot(3)>)]_{\equiv_B} = \{odd(<1, \cdot(3)>), odd(<1, 8>)\} = [odd(<1, \cdot(3)>)]_{\equiv'_B}$

$= [odd(<1, \cdot(3)>)]_{\equiv''_B}$.

On the other hand, since $[1]_{\sim_D} = [0]_{\sim_D} = \{<0, 2>, <0, 4>, <0, \cdot(5)>, <0, \cdot(9)>, <1, 0>, <1, \cdot(3)>, <1, 8>, 1, 0\}$ by definition,

$[odd(1)]_{\sim_B} = [odd(0)]_{\sim_B}$

$= \{odd(<0, 2>), odd(<0, 4>), odd(<0, \cdot(5)>), odd(<0, \cdot(9)>),$

$odd(<1, 0>), odd(<1, \cdot(3)>), odd(<1, 8>), odd(1), odd(0).\}$

⊣

Define $\sim$ over $A(\Pi P_D, V)$ so that it is the substitutive completion of $\sim_B$. Next, define $[[\ \ ]]_{\equiv_B} : BP_D \cdot B$ so that $[[\ \ ]]_{\equiv_B}$ refines the static relation $\sim_B$ to $\equiv_B$. By definition, it is this $[[\ \ ]]_{\equiv_B}$ that concerns both the irregularity and special status of exceptional data. (Any concrete assignment will do, if only $[[\ \ ]]_{\equiv_B}$ devide the equivalence class based on $\sim_B$ into subclasses based on $\equiv_B$ by following the Boolean-valued semantics). Similarly, we can define $[[\ \ ]]_{\equiv'_B}$ and $[[\ \ ]]_{\equiv''_B}$.

Example 6-4-3. Just watch only atoms in Example 6-4-2, that is, consider

$[[\ \ ]]_{=_B} \mid [\mathrm{odd}(1)]_{\sim_B}$. We would like to devide $[\mathrm{odd}(1)]_{\sim_B}$ into disjoint subclasses $[\mathrm{odd}(1)]_{=_B}$ and $[\mathrm{odd}(0)]_{\equiv_B}$ and $[\mathrm{odd}(<1, \cdot(3)>)]_{=_B}$. Let F be a complete filter over B which is used to witness that $(P_D, [[\ \ ]]_{=_B})$ becomes a F-model. Then, for any $\sigma \in [\mathrm{odd}(1)]_{\sim_B}$, we can take any value such that,

$$[[\ \sigma\ ]]_{=_B} = \begin{cases} \mathbf{1} & , \text{if} \quad \sigma \in [\mathrm{odd}(1)]_{=_B} \\ \mathbf{a} \notin F & , \text{if} \quad \sigma \in [\mathrm{odd}(0)]_{\equiv_B} \\ \mathbf{b} \notin F & , \text{otherwise, i.e.,} \sigma \in [\mathrm{odd}(<1, \cdot(3)>)]_{=_B}\ , \end{cases}$$

where $\mathbf{a} \neq \mathbf{b}$.

Moreover, if negation as failure is available, we should strengthen the conditions $\mathbf{a} \notin F$ and $\mathbf{b} \notin F$ to $\mathbf{a} \in I$ and $\mathbf{b} \in I$ respectively, where I is the dual ideal of F. (In this case, it is natural to take $\mathbf{a} = \mathbf{0}$.) Here, the value assignments are based on, say, (2) introduced in Example 6-2-2. Similarly, we would like to assign

$$[[\ \sigma\ ]]_{='_B} = \begin{cases} \mathbf{1} & , \text{if} \quad \sigma \in [\mathrm{odd}(1)]_{='_B} \\ \mathbf{e} \notin F & , \text{if} \quad \sigma = \mathrm{odd}(<0, \cdot(9)>) \\ \mathbf{a} \notin F & , \text{if} \quad \sigma \in [\mathrm{odd}(0)]_{='_B} \\ \mathbf{b} \notin F & , \text{otherwise, i.e., } \sigma \in [\mathrm{odd}(<1, \cdot(3)>)]_{='_B}\ , \end{cases}$$

where $\mathbf{a} \neq \mathbf{b} \neq \mathbf{e}$.

Using negation as failure, change conditions $\mathbf{a} \notin F$ and $\mathbf{b} \notin F$ and $\mathbf{e} \notin F$ to $\{\mathbf{a}, \mathbf{b}, \mathbf{e}\} \subseteq I$. (It may be natural to take $\mathbf{a} = \mathbf{0}$.) In this case, using the semantics of $\equiv'_D$, we may implement the above assignment by, say,

$$\mathrm{odd}(X) \leftarrow \mathrm{member}(X, D), \lambda_2(X) = 0, \mathrm{odd}(\lambda_1(X)) \qquad \cdots (7)$$

(Compared with (2), the clause concerning '(9) is lacked, which has been the original aim of our employing $\equiv'_D$.)

Finally, we assign

$$[[\ \sigma\ ]]_{\equiv''_B} = \begin{cases} 1 & \text{, if } \quad \sigma \in [\mathrm{odd}(1)]_{\equiv''_B} \\ \mathbf{e} \notin F \cup I\text{, if} & \quad \sigma = \mathrm{odd}(<0, \cdot(9)>) \\ \mathbf{a} \notin F & \text{, if } \quad \sigma \in [\mathrm{odd}(0)]_{\equiv''_B} \\ \mathbf{b} \notin F & \text{, otherwise, i.e., } \quad \sigma \in [\mathrm{odd}(<1, \cdot(3)>)]_{\equiv''_B}\text{,} \end{cases}$$

where $\mathbf{a} \neq \mathbf{b}$. Using negation as failure, change conditions $\mathbf{a} \notin F$ and $\mathbf{b} \notin F$ to $\{\mathrm{a}, \mathrm{b}\} \subseteq I$. (It may be natural to take $\mathbf{a} = 0$.) In this case, we employ (7) to obtain the same effects as $[[\quad]]_{\equiv'_B}$ except the value assignment of $[[\ \mathrm{odd}(<0, \cdot(9)>)]]_{\equiv''_B}$, which should be vague in the sense of B by the semantics of Δ. (The reason will be clear soon.)

Thus, through steps [1]→[2]→[3], corresponding Boolean values changes

$[[\mathrm{odd}(<0, \cdot(9)>)]]_{\equiv'_B} \in I$

$\Rightarrow \quad [[\mathrm{odd}(<0, \cdot(9)>)]]_{\equiv''_B} \notin F \cup I$

$\Rightarrow \quad [[\mathrm{odd}(<0, \cdot(9)>)]]_{\equiv_B} = 1 \in F.$

On the contrary, according to the arguments discussed in the previous section, we notice that,

$[[<(1)(<0, \cdot(9)>)]]_{\equiv'_B} \in F$

$\Rightarrow \quad [[<(1)(<0, \cdot(9)>)]]_{\equiv''_B} \notin F \cup I$

$\Rightarrow \quad [[<(1)(<0, \cdot(9)>)]]_{\equiv_B} \in I$ ⊣

Now, let $\psi \in \Pi_{P_D}$ be arbitrary and $[\psi]$ be the set of all ground atoms whose predicate is ψ. Since the difference between, say, $\equiv_D$ and $\equiv'_D$ is related only to exceptional data, as the above simple Example 6-4-3 typically shows, the difference between $\equiv_B \mid [\psi]$ and $\equiv'_B \mid [\psi]$ also related only to those atoms, of which (at least one) exceptional datum becomes an argument. From the arbitraryness of ψ, we notice that the difference between $[[\quad]]_{\equiv_B}$ and $[[\quad]]_{\equiv'_B}$ concerns only atoms with exceptional data in its argument(s). The same arguments apply to $([[\quad]]_{\equiv_B}, [[\quad]]_{\equiv''_B})$ and $([[\quad]]_{\equiv'_B}, [[\quad]]_{\equiv''_B})$, too. Here, the transformation $[[\quad]]_{\equiv'_B} \rightarrow [[\quad]]_{\equiv''_B} \rightarrow [[\quad]]_{\equiv_B}$ of value assignment becomes the Boolean-valued expression of non-monotonicity.

Note: Let $Q=\{\sigma\in Bp|[[\sigma]]=_B\neq[[\sigma]]='_B\}$. (Practically, Q becomes finite.) During steps [1]→[2]→[3], Q works as, so to speak, oracle. Generally, any atom concerning Δ which is B-valued in [1] is checked in [2] by Q to possibly change the B-value in [3]. Thus, Q judge the B-valued revolution

$$\mathbf{e}\in F \rightarrow \mathbf{e}\in I$$

and/or $$\mathbf{e}\in I \rightarrow \mathbf{e}\in F$$

as well as stable transfer $\mathbf{e}\in F\rightarrow\mathbf{e}\in F$ and $\mathbf{e}\in I\rightarrow\mathbf{e}\in I$. (Consider, for example, $<2(X)$. For this predicate, we notice $\mathbf{e}\in F\rightarrow\mathbf{e}\in F$.) In this sense, any B-value concerning these atoms in [1] becomes neutral once in [2] to be sentenced in [3].

⊣

References

[1] K.R. Apt and M.H. van Emden, "Contributions to the Theory of Logic Programming", J. ACM, vol. 29, No. 3 (1982), 841-862.

[2] R.G. Bandes, "Constraining-unification and the Programming Language: Unicorn", in Logic Programming: Relations, Functions and Equations, ed. by D. DeGroot and G. Lindstrom, Prentice-Hall (1985), 397-410.

[3] 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.

[4] J. Darlington, A.J. Field and H. Pull, "The Unification of Functional and Logic Languages", in Logic Programming: Relations, Functions and Equations, ed. by D. DeGroot and G. Lindstrom, Prentice-Hall (1985), 37-70.

[5] M. Dincbas and P. van Hentenryck, "Extended Unification Algorithms for the Integration of Functional Programming into Logic Programming", J. Logic Programming, vol. 4 (1987), No.4, 199-227.

[6] M.H. van Emden, "Quantitative Deduction and Its Fixpoint Theory", J. Logic Programming, vol. 3 (1986), 37-53.

[7] M.H. van Emden and R.A. Kowalski, "The Semantics of Predicate Logic as a Programming Language", J. ACM, vol. 23, No. 4 (1976), 733-742.

[8] F. Fages and G. Huet, "Complete Sets of Unifiers and Matchers in Equational Theories", Theoretical Computer Science 43 (1986), 189-200.

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

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

[11] J. A. Goguen and J. Meseguer, "EQLOG: Equality, Types and Generic Modules for Logic Programming", J. Logic Programming 1 (1984), 179-209.

[12] J. Jaffar, J.L. Lassez and M.J. Maher, "Logic Programming Language Scheme", in Logic Programming: Relatios, Functions and Equations, ed. by D. DeGroot and G. Lindstrom, Prentice-Hall (1985), 441-467.

[13] 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.

[14] J. Jaffar, J.L. Lassez and M.J. Maher, "A Theory of Complete Logic Programs with Equality", Proceedings of the International Conference on Fifth Generation Computer Systems, 1984, ed. by ICOT, 175-184.

[15] J. Jaffar, J.L. Lassez and M.J. Maher, "Prolog-II as an Instance of the Logic Programming Language Scheme" Technical Report, Monash University, 1984.

[16] J. Jaffar and P.J. Stuckey, "Logic Program Semantics for Programming with Equations", Proceedings of the 3rd International Conference on Logic Programming, ed. by G. Goos and J. Hartmanis, Lecture Notes in Computer Science 225 (Springer-Verlag 1986), 313-326.

[17] J.L. Lassez and M.J. Maher, "Optimal Fixedpoints of Logic Programs", Theoretical Computer Science, vol. 39 (1985), 15-25.

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

[19] W.A. Kornfeld, "Equality for Prolog", in Logic Programming : Relations, Functions and Equations, ed. by D. DeGroot and G. Lindstrom, Prentice-Hall (1985), 279-293.

[20] R. Kowalski, "Logic for Problem Solving", North-Holland, 1979.

[21] J.W. Lloyd, "Foundations of Logic Programming", Springer-Verlag, 1984.

[22] D.W. Loveland, "Automated Theorem Proving: a Logical Bases", North-Holland, 1978.

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

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

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

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

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

[28] U.S. Reddy, "On the Relationship between Logic and Functional Languages" in Logic Programming: Relations, Functions and Equations, ed. by D. DeGroot and G. Lindstrom, Prentice-Hall (1985), 3-35.

[29] R. Reiter, "On Closed World Data Bases", in: H. Gallaire and J. Minker (ed.), Logic and Data Bases (Plenum Press 1978), 55-76.

[30] J. Siekmann and P. Szabó, "Universal Unification and a Classification of Equational Theories", Proceedings of the 6th Conference on Automated Deduction, ed. by D.W. Loveland, Lecture Notes in Computer Science 138 (Springer-Verlag 1982), 369-389.

[31] P.A. Subrahmanyam and J.H. You, "Funlog : A Computational Model Integrating Logic Programming and Functional Programming", in Logic Programming : Relations, Functions and Equations, ed. by D. DeGroot and G. Lindstrom, Prentice-Hall (1985), 157-198.

[32] J. Yamaguchi, "Boolean-valued Logic Programming Language Paradigm: LIFE-Ω —Philosophical Background—", NEC LR-5196, revised version to appear.

[33] J. Yamaguchi, "Boolean-valued Logic Programming Language Paradigm: LIFE-Ω —Theoretical Background of LIFE- I , II, III —", NEC LR-5197, revised version to appear.

[34] J. Yamaguchi, "Logical Completeness of LIFE-II and LIFE-III and Its Applications to the Foundation of Logic Programming", NEC LR-5346, revised version to appear.

[35] J. Yamaguchi, "Universal Unification from a Viewpoint of LIFE-II", NEC LR-5347, revised version to appear.

[36] J. Yamaguchi, "Boolean-valued Logic Programming Language Scheme: LIFE-III —A Summary —", NEC LR-5357, revised version to appear.

[37] J. Yamaguchi, "Two-lane Unification and A.B.F.E.-unification —One Step Toward an Application of LIFE-II —", NEC LR-5426, revised version to appear.

[38] J. Yamaguchi, "Boolean-valued Logic Programming Language Scheme: LIFE-III [1] Inferentially Transforming Logic Programs", (in Japanese) 5th Conference Proceedings Japan Soc. Software Sci. & Tech., 1988, 237~240.

[39] J. Yamaguchi, "Boolean-valued Logic Programming Language Scheme: LIFE-III [2] A Technique to Execute a Logic Program Efficiently", (in Japanese) 5th Conference Proceedings Japan Soc. Software Sci. & Tech., 1988, 241~244.

[40] J. Yamaguchi, "F-LIFE: a L-fuzzy Inferential System", NEC LR-5544, revised version to appear.

[41] J. Yamaguchi, "Inference Transformation", in preparation.