A Construction of Logic-Constrained Functions with Respect to Awareness

Susumu Yamasaki

Department of Computer Science, Okayama University, Japan
email: yamasaki@momo.cs.okayama-u.ac.jp

Abstract

A logic-constrained function is motivated by modelling the behaviour of the electronic device (like a mobile phone) with respect to evocation caused by awareness. This paper presents an analysis to develop a logic-constrained function system. We firstly have the contradiction removal procedure for a proof system which is expected with *negation as failure* rule (as denoting *unawareness*) to derive the constraint (supposedly as *awareness*) with reference to the (abstract) functions like λ-terms. Though the system deriving constraints can be the logic programming system (Alferes,J.J. et al.) of coherence principle, a limited reasoning (with negation as failure) is treated in this paper to cope with inconsistency (contradiction) caused by classical negation, in terms of contradiction removal facility. We then have a description of the logic-constrained function. As a logic-constrained function, we have an outlook on a literal-constrained term, an extended λ-term, where the literal may be derivable from a proof system. The term conversions are defined such that a system originating from awareness may have abstract function applications, based on the literal-constrained term.

1 Introduction

As a ubiquitous system, we can assume the electronic device (like a mobile phone) in place of the PC in distributed environments through internets or wireless communications. (i) The device is expected to hold functional and objective knowledge abstractly. (ii) The device as a resource is bounded to be left alone until it is evoked and of use. The evocation of the device can be supported by constraint awareness of (a set of) states (as in [11]) which some predicate (with or without classical negation) may denote. That is, the evocation may be interpreted to be realized by an awareness of states (i.e., of a predicate). (iii) As long as we are concerned with awareness of a predicate for such a set of states, it is implementable as derivability from a proof system. (iv) In a proof system, negation as failure is applicable to unawareness, owing to derivability of the designated predicate. (Note

that the closed world assumption is broader than negation as failure with reference to unawareness.)

Therefore we have an illustrative structure on logical awareness as in Table 1.

Derivability basis		Evocation		Device
A proof system	$\Rightarrow$	Awareness	$\Rightarrow$	Function construction

Table 1: Logical awareness

We then have technical problems: (1) Function constructions constrained by predicates (constructions of logic-constrained function) are to be modelled for the mobile phone evoked by awareness of a state set. (2) If we regard awareness as caused by derivability from a proof system, paraconsistency should be denied. If a complementary pair of predicates with and without negation is derivable, one of them must be removed.

Regarding the problems, abstract functions have been deeply studied in the λ-calculus such that there were many compact textbooks among which we can see the one ([9]). From knowledge views on function applications, an analysis of the function (application) constrained by logic concerning awareness is also a problem. For the function to conceive logical awareness, the logic-constrained function may be constructed, where the logical formula (which constrains functions) is often interpreted as: procedure or process ([12]) and state set ([11]) possibly inducing space and/or time notions, such that the logical formula can be concerned with awareness of some agent. In this paper, the notion of the state set is adopted. The denotation of a state set is described by a predicate (with or without classical negation) which may be derived (deduced) from a proof system. As regards derivability, *negation as failure* in computational logic ([12]), the inference rule "$\nvdash p \Rightarrow \sim p$ (or denoted as *not* p)" has been well studied. Unawareness may be defined in terms of negation as failure.

Motivated by the above problems, this paper is to analyze the logic-constrained function construction, where: (a) the logic programming system derives the literals with and without classical negation, by means of reasoning containing negation as failure, and (b) a logic-constrained function is defined by means of a combination of λ-terms with the logical formulae.

Regarding the logic programming system, the negative literals are as significant as positive ones, with reference to the closed world assumption, the default and/or negation as failure rules ([2, 17]). As an analysis, we need the negation as failure to denote unawareness with a contradiction-free reasoning. As before pointed out for the second problem, a complementary

pair of literals with and without classical negation causes a contradiction even with respect to awareness. That is, the contradiction is not allowed for logic-constrained function: the contradiction is to be removed from awareness view such that reasoning to remove contradiction must be restricted. Concerning the logic-constrained function, the established λ-calculus is fundamental and may be extended to the one where the (λ-)term constrained by logical formulae is definable. This paper focuses on the definable terms generally denoting the logic-constrained functions, where the logic-constrained function is a λ-term with respect to awareness expressed by a logical formula. The awareness is regarded as caused by derivability from a proof system, while negation as failure for derivability supposedly denotes unawareness.

The paper of the analysis for logic-constrained functions is organized for the problem as motivations described here. In Section 2, the logic programming system is reviewed from the propositional logic version, where an idea of the contradiction removal is presented. Section 3 presents a contradiction removal procedure and its soundness is shown. In Section 4, we have an outlook on an extended λ-term with the literal (possibly containing classical negation) and its conversions. Section 5 gives comments regarding logic-constraints on the λ-term.

2 Logical Expressions

We now review the terminologies in propositional logic programming, as a proof system to possibly derive literals with and without classical negation. Negation as failure is to denote an *unawareness*, while contradiction caused by a pair of derivable literals (*awareness*) with and without classical negation must be removed by a limited reasoning.

(1) A set of symbols to stand for propositions is assumed.

(2) Two kinds of negation sign are taken: the classical negation "$\neg$" and the negation as failure "$\sim$".

(3) A literal l is either an atom a, or a classical negation $\neg a$ of an atom a. An atom is an expression consisting of a symbol to denote a proposition.

Classical negation vs. negation as failure

The class of extended logic programs in the propositional logic is treated, where two kinds of negation may be contained. As a theory for the *literal-constrained term* which would be mentioned later, an extended logic program (ELP, for short) is a set of clauses of the form

$$l \leftarrow l_1, \ldots, l_m, \sim l_{m+1}, \ldots, \sim l_n \ (0 \leq m \leq n),$$

where l and l_i are literals, and "$\sim$" stands for the negation as failure (NAF, for short). The literal $\sim l$ is called a negation-as-failure literal. The literal l of the clause is said to be its head, and the literal sequence $l_1, \ldots, l_m$, $\sim l_{m+1}, \ldots, \sim l_n$ is its body. The classical negation $\neg l$ means

(i) $\neg a$ if $l = a$, and

(ii) a if $l = \neg a$,

for an atom a. The pair of literals a and $\neg a$ is said to be complementary.

The expressions L, L_1, ..., L_m, ..., M, M_1, ..., M_n, ...are reserved to denote literals or negation-as-failure literals. The expressions α, α_1, ..., α_m, ..., β, β_1, ..., β_n, ...are reserved to denote sequences of literals or negation-as-failure literals.

A goal is an expression of the form $\leftarrow l_1, \ldots, l_m, \sim l_{m+1}, \ldots, \sim l_n$ $(0 \leq m \leq n)$, where l_i are literals. The goal of the form $\leftarrow \sim m_1, \ldots, \sim m_q$ $(q \geq 0)$ is said to be a negative goal. The negative goal is the empty clause, denoted by $\Box$, if it contains no literal. For the ELPs, we must note a well established paraconsistent reasoning (which is regarded as a proof procedure to possibly derive literals) equipped with *coherence principle* ([1]):

> "For any objective literal $\neg l$, if $\neg l$ is entailed by the semantics, then $\sim l$ is also entailed."

In the paper, a proof procedure with coherence principle is given, while some condition for the program to be consistent is shown.

We have an outlook on SLD resolution and negation as failure, where we can see [12, 17] among so many papers.

SLD resolution applied to goals is a rule to derive

$$\leftarrow L_1, \ldots, L_{i-1}, M_1, \ldots, M_n, L_{i+1}, \ldots, L_m$$

from a goal $\leftarrow L_1, \ldots, L_{i-1}, l, L_{i+1}, \ldots, L_m$ and a clause $l \leftarrow M_1, \ldots, M_n$ in the given ELP. That is, a literal l of a goal may be replaced by the body $M_1, \ldots, M_n$ of a clause $l \leftarrow M_1, \ldots, M_n$, whose head is just the literal l.

Negation as failure is a rule to infer a negation-as-failure literal $\sim l$, when the literal l is not derived by some proof procedure. We refine negation as failure in relation to SLD resolution such that the succeeding and failing (derivations) of a goal (with reference to a given ELP) are defined recursively as follows:

(i) The goal $\leftarrow l$ succeeds if $\leftarrow l$ is reduced to $\Box$ by applying finitely many SLD resolutions.

(ii) The goal $\leftarrow l$ fails if one of following conditions holds:

  (a) there is no clause whose head is the literal l.

  (b) there are goals $\leftarrow \alpha_1, L_1, \beta_1, \ldots, \leftarrow \alpha_n, L_n, \beta_n$ $(n \geq 1)$, derived by SLD resolutions for the goal $\leftarrow l$ such that all the goals $\leftarrow L_1$, $\ldots, \leftarrow L_n$ fail.

  (c) the goal $\leftarrow \neg l$ succeeds.

(iii) The goal $\leftarrow \sim l$ succeeds if the goal $\leftarrow l$ fails.

(iv) The goal $\leftarrow \sim l$ fails if the goal $\leftarrow l$ succeeds.

Note the sense that if the goal $\leftarrow l$ may succeed, then we may be aware of l.

Contradiction removal

Compared with the method of [1] involving the coherence principle, we have a different method of removing contradictory succeeding derivations of both the goal $\leftarrow a$ and the goal $\leftarrow \neg a$. If both goals may succeed, contradictory awareness of a and $\neg a$ is made, which should be escaped.

Now assume a propositional ELP

$$Q = \{r \leftarrow q, \neg q; q \leftarrow \sim p; p \leftarrow \sim q; \neg q \leftarrow\}.$$

For the goal $\leftarrow r$, we have the derivation by SLD resolution: A goal $\leftarrow r$ is reduced to a goal $\leftarrow q, \neg q$, as illustrated below.

$$\begin{array}{ll} \leftarrow r & \\ \quad | & \text{(with a clause } r \leftarrow q, \neg q) \\ \leftarrow q, \neg q & \end{array}$$

The goal $\leftarrow \neg q$ can succeed with the clause $\neg q \leftarrow$, where it is reduced to the goal $\Box$ as shown below.

$$\begin{array}{l} \leftarrow \neg q \\ \quad | \qquad \text{(with a clause } \neg q \leftarrow) \\ \quad \Box \end{array}$$

It follows that the goal $\leftarrow q, \neg q$ is reduced to the goal $\leftarrow q$, while the goal $\leftarrow q$ cannot succeed, for contradiction removal, after the success of the goal $\leftarrow \neg q$.

Alternatively, if we have a derivation of the goal $\leftarrow q$ to $\Box$, whose details is omitted, the goal $\leftarrow q, \neg q$ is reduced to the goal $\leftarrow \neg q$, which is to be suspended. Here we can see that the goal $\leftarrow r$ cannot succeed. Finally the following requirements are implemented.

(i) At most one of goals $\leftarrow q$ and $\leftarrow \neg q$ is permitted to succeed. (By means of some memory for the succeeding derivation to eliminate contradictions, the contradiction-free derivation is automated.)

(ii) If the goal $\leftarrow q$ succeeds, then the goal $\leftarrow \neg q$ can be regarded as failing.

3 Reasoning Procedure

For reasoning to be apart from paraconsistency, we have two sets to be kept, which are to be transformed through succeeding and failing derivations:

(i) the set of literals to remove contradictory succeeding derivations

(ii) the set of negation-as-failure literals

The former set (i) is expressed by Σ, Σ_1, ..., and the second one (ii) is by Δ, Δ_1, ... Given an ELP P, the predicate $suc_P(G; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$ is derived, when a goal G succeeds with the assumed sets Σ_1 and Δ_1, to acquire the sets Σ_2 and Δ_2. The predicate $fail_P(G; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$ is derived, when a goal G fails with the assumed sets Σ_1 and Δ_1, to acquire the sets Σ_2 and Δ_2. Following the derivations as above, we have the rules. Intuitively the succeeding derivation expresses some awareness, while the failing derivation detects unawareness, in relation to negation-as-failure rules.

We assume an ELP P and have the relational representations of succeeding and failing derivations, where the relations suc_P and $fail_P$ are defined simultaneously by recursion to be the least set satisfying the following closure. Because the relations demonstrate the implementation as procedural methods for the given ELP, they can be regarded as providing behaviours

such that the abstraction of the representation is more general than those in [19]. The subscript P for the ELP P may be omitted if it is clear in the context.

(0) $suc_P(\Box; \Sigma; \Delta; \Sigma; \Delta)$ for any Σ and Δ.

(1) $suc_P(\leftarrow L_1, \ldots, L_{i-1}, M_1, \ldots, M_m, L_{i+1}, \ldots, L_n; \Sigma_1 \cup \{l\}; \Delta_1; \Sigma_2; \Delta_2)$ for $\neg l \notin \Sigma_1$ and $(l \leftarrow M_1, \ldots, M_m) \in P \Rightarrow$
$suc_P(\leftarrow L_1, \ldots, L_{i-1}, l, L_{i+1}, \ldots, L_n; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$.

(2) $suc_P(\leftarrow L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$ and $\sim l \in \Delta_1 \Rightarrow$
$suc_P(\leftarrow L_1, \ldots, L_{i-1}, \sim l, L_{i+1}, \ldots, L_n; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$.

(3) $suc_P(\leftarrow L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n; \Sigma_2'; \Delta_2'; \Sigma_2; \Delta_2)$ and
$fail_P(\leftarrow l; \Sigma_1; \Delta_1 \cup \{\sim l\}; \Sigma_2'; \Delta_2')$ for $\sim l \notin \Delta_1 \Rightarrow$
$suc_P(\leftarrow L_1, \ldots, L_{i-1}, \sim l, L_{i+1}, \ldots, L_n; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$.

(4) There is no clause in P, which contains l in the head $\Rightarrow$
$fail_P(\leftarrow L_1, \ldots, L_{i-1}, l, L_{i+1}, \ldots, L_n; \Sigma; \Delta; \Sigma; \Delta)$ for any Σ and Δ.

(5) For any clause $l \leftarrow M_1^j, \ldots, M_{n_j}^j \in P$ $(1 \leq j \leq k)$ of all the clauses which contain l in the head,
$fail_P(\leftarrow L_1, \ldots, L_{i-1}, M_1^j, \ldots, M_{n_j}^j, L_{i+1}, \ldots, L_n; \Sigma_j; \Delta_j; \Sigma_{j+1}; \Delta_{j+1}) \Rightarrow fail_P(\leftarrow L_1, \ldots, L_{i-1}, l, L_{i+1}, \ldots, L_n; \Sigma_1; \Delta_1; \Sigma_{k+1}; \Delta_{k+1})$.

(6) $suc_P(\leftarrow \neg l; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2) \Rightarrow fail_P(\leftarrow l; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$.

(7) $fail_P(\leftarrow L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$ and $\sim l \in \Delta_1 \Rightarrow$
$fail_P(\leftarrow L_1, \ldots, L_{i-1}, \sim l, L_{i+1}, \ldots, L_n; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$.

(8) $suc_P(\leftarrow l; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$ for $\sim l \notin \Delta_1 \Rightarrow$
$fail_P(\leftarrow L_1, \ldots, L_{i-1}, \sim l, L_{i+1}, \ldots, L_n; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$.

We can see the characteristics of the relational representations:

(a) If we have the relation $suc_P(G; \emptyset; \emptyset; \Sigma; \Delta)$ for a goal G, the whole interpreter for the logic program can detect corresponding derivations. The literals of the goal G in the relation suc_P are regarded as made aware of, while if the goal G is included in the relation $fail_P$ then the literals of the goal is concerned with unawareness.

(b) In the relation $suc_P(G; \emptyset; \emptyset; \Sigma; \Delta)$, the set Δ contains the extraction of negation-as-failure literals.

(c) By the set Σ of the relation $suc_P(G; \emptyset; \emptyset; \Sigma; \Delta)$, we can trace the literals used for SLD resolution.

We see an illustration.

Example. Take a simple ELP $P = \{p \leftarrow, \neg p \leftarrow\}$.

(i) Clearly we have the empty clause $\Box$ from a goal $\leftarrow p$ with a given clause $p \leftarrow$ such that

$$suc_P(\leftarrow p; \emptyset; \emptyset; \{p\}; \emptyset).$$

(ii) Similarly we have $suc_P(\leftarrow \neg p; \emptyset; \emptyset; \{\neg p\}; \emptyset)$. However, we cannot have

$$suc_P(\leftarrow \neg p; \{p\}; \emptyset; \{\neg p\}; \emptyset),$$

after that we have got $suc_P(\leftarrow p; \emptyset; \emptyset; \{p\}; \emptyset)$, as in the case of (i).

(iii) After we have got $suc_P(\leftarrow p; \emptyset; \emptyset; \{p\}; \emptyset)$ which is concerned with awareness of the literal p, the inference rule gives

$$fail_P(\leftarrow \neg p; \emptyset; \emptyset; \{p\}; \emptyset),$$

which detects unawareness of the literal $\neg p$.

In the following sense, the relation suc_P, which may involve the effect of the relation $fail_P$, is sound. This is paraphrased to the sense of consistency that if a goal $\leftarrow l$ succeeds, then $\sim l$ cannot be included in the set of negation-s-failure literals. It also contains an interpretation that awareness reasoned by the relation suc_P is consistent with unawareness detected by the relation $fail_P$. The proof is made. Here we see its outline.

Definition 3.1 For a set Σ, we define the set $\tilde{\Sigma}$ to be $\{\sim \neg l \mid l \in \Sigma\}$.

Theorem 3.2 *Assume that* $suc_P(\leftarrow l; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$ *such that* $\Sigma_1 = \Delta_1 = \emptyset$. *Then* $\sim l \notin \Delta_2 \cup \tilde{\Sigma_2}$.

Proof (Outline) (1) By the definition of the relation suc_P, $\neg l \notin \Sigma_2$. It follows that $\sim l \notin \tilde{\Sigma_2}$.
(2) On the contrary to the assumption that $\sim l \notin \Delta_2$, suppose that $\sim l \in \Delta_2$. Then, by the construction of the set Δ_2, it follows that

$$fail_P(\leftarrow l; \Sigma_1'; \Delta_1'; \Sigma_2'; \Delta_2')$$

for some sets $\Sigma_1', \Delta_1', \Sigma_2', \Delta_2'$ such that $l \in \Delta_1' \subseteq \Delta_2' \subseteq \Delta_2$. There are the following cases to support this relation $fail_P$.

(i) In case that $suc_P(\leftarrow \neg l; \Sigma_1'; \Delta_1'; \Sigma_2'; \Delta_2')$, this contradicts the first assumption that $suc_P(\leftarrow l; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$.

(ii) In case that there is no clause whose head is l for the relation $fail_P$, this contradicts the first assumption that $suc_P(\leftarrow l; \Sigma_1; \Delta_1; \Sigma_2; \Delta_2)$ such that there is some clause whose head is l.

(iii) In case that $fail_P(\leftarrow \sim m_1, \ldots, \sim m_n; \Sigma_1''; \Delta_1''; \Sigma_2''; \Delta_2'')$ for some sets $\Sigma_1'', \Delta_1'', \Sigma_2'', \Delta_2''$, which may be caused for some negative goal

$$\leftarrow \sim m_1, \ldots, \sim m_n$$

derivable from the goal $\leftarrow l$, it is concluded that $n \neq 0$. Otherwise, the negative goal $\leftarrow \sim m_1, \ldots, \sim m_n$ is $\Box$ and it contradicts the relation $fail_P$. For the negative goal

$$\leftarrow \sim m_1, \ldots, \sim m_n,$$

there is some literal $m_i \notin \Delta_1''$ $(1 \leq i \leq n)$ such that $suc_P(\leftarrow m_i; \Sigma_1''; \Delta_1''; \Sigma_2''; \Delta_2'')$, because of the relation $fail_P$. On the assumption that $\sim m_i \in \Delta_2$, we repeat the same discussion. We finally reach the case that:

$$fail_P(\leftarrow \sim m_1^f, \ldots, \sim m_n^f; \Sigma_1^f; \Delta_1^f; \Sigma_2^f; \Delta_2^f)$$

for some sets $\Sigma_1^f, \Delta_2^f, \Sigma_2^f, \Delta_2^f$, but there is no literal $\sim m_i^f$ such that $\sim m_i^f \notin \Delta_1^f$. This causes the case that $\leftarrow \sim m_1^f, \ldots, \sim m_n^f = \Box$, which contradicts that $fail_P(\leftarrow \sim m_1^f, \ldots, \sim m_n^f; \Sigma_1^f; \Delta_2^f; \Sigma_2^f; \Delta_2^f)$. This concludes the proof.

Q.E.D.

4 Literal-Constrained Term

We here have the form: a constrained literal followed by a (function) term, where

(a) the literal is derivable from some proof system like the logic programming system such that the literal may be interpreted as awareness (of an agent), and

(b) the function term is held (by an agent) under the awareness of the literal,

such that the form may be regarded as denoting a behaviour of an agent.

Syntax

An extended term from the original is shown below, where a logic-constraint may be made by a literal. If we prefer to the ELP, which derives literals, then the derivable literals may be constraints.

Definition 4.1 On the assumption of a proof system (say, Γ), a literal-constrained term (*term*, for short) is recursively defined as follows:

(i) If x is a variable, then x is a term.

(ii) If M, N are terms, then $(M\ N)$ is a term.

(iii) If x is a variable and M a term, then $(\lambda x.M)$ is a term.

(iv) If p is a literal and M a term, then $p < M >$ is a term.

Semantics

By the term $p < M >$ with some system Γ (which may possibly be the ELP in the previous section), we mean that:

- If p is derivable from Γ, then the term M is supported.
- Unless p is derivable from Γ, then the term M is not supported.

When the ELP may be taken as a proof system Γ, the contradiction removal procedure is significant, because $p < M >$ and $\neg p < M >$ cannot be coherent.

Illustration

Assume the following functional program.

$$\begin{aligned} Even(x) = {} & \textbf{if } x = 0 \textbf{ then } true \\ & \textbf{else if } x = 1 \textbf{ then } false \\ & \textbf{else } Odd(x-1) \end{aligned}$$

$$Odd(x) = \textbf{if } x = 0 \textbf{ then } false$$

else if $x = 1$ **then** $true$
else $Even(x - 1)$

As a standard way, by means of a *fixed point operator*:

$$\text{Let Y} = \lambda f.(\lambda x.f(x\ x))(\lambda x.f(x\ x)),$$

$$\begin{array}{l}
\text{Let } evenfn = \lambda g.\lambda n.(\text{if } n = 0 \text{ then } true \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{else if } n = 1 \text{ then } false \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{else } g\ (-\ n\ 1)) \\
\text{Let } oddfn = \lambda f.\lambda m.(\text{if } m = 0 \text{ then } false \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{else if } m = 1 \text{ then } true \\
\qquad\qquad\qquad\qquad\qquad\qquad \text{else } f\ (-\ m\ 1)) \\
\text{Let } even = Y\ (evenfn\ oddfn) \\
\text{Let } odd = Y\ (oddfn\ evenfn)
\end{array}$$

The terms "$evenfn$" and "$oddfn$" may be included in the expressions such as:

$$p < evenfn > \text{ and } q < oddfn >,$$

where the literals may denote awareness, which acknowledges the application of $p < evenfn >$ to $q < oddfn >$ for the usual term $(evenfn\ oddfn)$ to $even$, and vice versa for the term $(oddfn\ evenfn)$ to odd.

Condition and conversion

By the expression $p \longrightarrow q$, we mean that if p is derivable from Γ, then q is derivable from Γ.

In addition to the standard α, β, and η conversions, the following two conversions are to be presented:

$$(\gamma 1)\ \frac{(p < M >\ \ q < N >) \quad p \longrightarrow q}{q < (M\ N) >}$$

$$(\gamma 2)\ \frac{(\lambda x.p < M >)}{p < \lambda x.M >}$$

Assume the program.

$f(x) = \textbf{if } x = 0 \textbf{ then } 1 \textbf{ else } x \times f(x-1)$

Let $factorial = Y\ factorialfn$,
Let $factorialfn = \lambda f.\lambda x.(((iszero\ x)\ 1)\ (times\ x\ (f\ (-x\ 1))))$,

where the λ-term $((B\ X_1)\ X_2)$ can denote the conditional sentence **if** B **then** X_1 **else** X_2. If we have the fixed point operator $p < Y >$, the if-part $q < (iszero\ 0)\ 1 >$, and the else-part

$$r < Times\ x\ (f\ (-\ x\ 1)) >$$

such that $p \longrightarrow q$ and $q \longrightarrow r$, then

$$r < Y\ factorialfn > = r < factorial >.$$

The relation $\Rightarrow^*$ denotes a reflexive and transitive closure based on α, β, η, $\gamma 1$ and $\gamma 2$ conversions.

Church-Rosser theorem

Because

(a) the calculus for terms constructed by using only (i), (ii) and (iii) conceives the *Church-Rosser Theorem*, and

(b) α, β and η conversions are commutative with $\gamma 1$ and $\gamma 2$ conversion applications,

we may see that the term constructed by Definition 4.1 is transformed to a *normal form* (which any conversion except α cannot be applied to) uniquely up to α conversion, with respect to the relation $\Rightarrow^*$, if the term has one. It is stated as:

Theorem 4.2 *If the term M has a normal form, it is unique up to α conversion.*

Proof (Outline) Applications of $\gamma 1$ and $\gamma 2$ may be sound with respect to the transformation to the normal form by the following senses:

Assume two terms M and N whose normal forms are M_{normal} and N_{normal}, respectively. For any terms M_1 and N_1 such that

$$M \Rightarrow^* M_1 \Rightarrow^* M_{normal}, \text{ and}$$
$$N \Rightarrow^* N_1 \Rightarrow^* N_{normal},$$

we have:

$$\dfrac{\dfrac{\dfrac{p < M > \Rightarrow^* p < M_1 > \quad q < N > \Rightarrow^* q < N_1 >}{(p < M_1 > \quad q < N_1 >) \quad p \longrightarrow q}}{q < (M_1\ N_1) >}}{q < M_{normal}\ N_{normal} >}$$

with respect to $\gamma 1$ conversion, and

$$\dfrac{\dfrac{\dfrac{\lambda x.p < M > \quad p < M > \Rightarrow^* p < M_1 >}{\lambda x.p < M_1 >}}{p < \lambda x.M_1 >}}{p < \lambda x.M_{normal} >}$$

with respect to $\gamma 2$ conversion such that we intuitively see the induction for the proof. Q.E.D.

If we adopt the ELP P as a formal system (Γ as above), whether

$$p \longrightarrow q$$

can be determined by reasoning that on condition that we have $suc_P(\leftarrow p; \emptyset; \emptyset; \Sigma; \Delta)$, we then have $suc_P(\leftarrow q; \Sigma; \Delta; \Sigma'; \Delta')$ for $\Delta \subseteq \Delta'$ and $\Sigma \subseteq \Sigma'$.

5 Concluding Remarks

The problem of treatments for the logic-constrained function is related to the backgrounds: (i) Logic and database views are fundamental to analyze knowledge structure ([14, 17]), to understand dynamic structure with reference to knowledge ([15, 16]). (ii) Process algebra deals with sequence structure of communications (evaluations) ([10, 13]) even for distributed systems ([5]). (iii) The logic programming system (in computational logic) contains the notion of negatives such that both classical negation (in the literal regarding awareness) and negation as failure (regarding unawareness) are combined for more powerful representations ([1]).

For a literal-constrained term, the literal should be derivable from an indicated proof system. If derivability is concerned with awareness, the term (as a function application) is regarded as originating from awareness with constraint. In this paper, the logic programming system with a contradiction removal procedure is presented, not allowed to be paraconsistent.

As regards awareness in terms of derivability, we can also have the formal systems: hybrid logic (with modality and nomination) (as in [3]), and action logic (as in [8]).

With reference to actions like those of [15, 16], analyses of whether or not we can apply them to the literal-constrained term are needed.

As the proof system itself, we summarize some points on specific expressiveness of the ELP. (1) This paper presents an abstract representation of reasoning for its application to the reasoning of contradiction removal regarding derivability vs. awareness, independent of abduction reasoning as in [4, 6]. (2) The backgrounds of soundness of succeeding and failing derivations may be closely related to model theory, following [1, 19]. (3) The notion of exceptions for each literal to be constrained by is implementable in the derivations we present, while we may apply *weak negation* to it as in [20]. (4) There is a problem of whether or not a non-grounded version of the literals (for awareness constraints) may be built in the proof system to derive literals for constraints. A non-grounded version of negation as failure is relevant to the discussions as in [17, 18].

References

[1] Alferes,J.J., Damásio,C.V. and Pereira,L.M., A logic programming system for nonmonotonic reasoning, J. of Automated Reasoning, 14, pp.93-147, 1995.

[2] Besnard,P., An Introduction to Default Logic, Springer-Verlag, 1989.

[3] Brauner,T., Natural deduction for hybrid logic, JLC, 14, 3, pp.329–353, 2004.

[4] Brogi,A., Lamma,P., Mancarella,P. and Mello,P., A unifying view for logic programming with non-monotonic reasoning, Theoretical Computer Science, 184, 1-2, pp.1-59, 1997.

[5] Bruns,G., Distributed Systems Analysis with CCS, Prentice-Hall, 1996.

[6] Dung,P.M., An argumentation-theoretic foundation for logic programming, J. of Logic Programming, 22, pp.151–177, 1995.

[7] Gelfond,M. and Lifschitz,V., The stable model semantics for logic programs, Proc. of 5th ICLP, pp.1070–1080, 1988.

[8] Giordano,L., Martelli,A. and Schwind,C., Ramification and causality in a modal action logic, JLC, 10, 5, pp.625–662, 2000.

[9] Gordon,M.J.C., Programming Language Theory and its Implementation, Prentice Hall, 1988.

[10] Hoare,C.A.R., Communicating Sequential Processes, Prentice-Hall, 1985.

[11] Kucera,A. and Esparza,J., A logical viewpoint on process-algebra, J. of Logic and Computation, 13, 6, pp.863–880, 2003.

[12] Lloyd,J.W., Foundations of Logic Programming, 2nd, Extended Edition, Springer-Verlag, 1993.

[13] Milner,R., Communication and Concurrency, Prentice-Hall, 1989.

[14] Minker,J. (ed.), Foundations of Deductive Databases and Logic Programming, Morgan Kaufmann Publishers, Inc., 1987.

[15] Mosses,P.M., Action Semantics, Cambridge University, 1992.

[16] Reiter,R., Knowledge in Action, The MIT Press, 2001.

[17] Shepherdson,J.C., Negation in Logic Programming, in: Minker,J. (ed.), Foundations of Deductive Databases and Logic Programming, pp.19-88, 1987.

[18] Yamasaki,S. and Kurose,Y., Soundness of abductive proof procedure with respect to constraint for non-ground abducibles, Theoretical Computer Science, 206, pp.257-281, 1998.

[19] Yamasaki,S. and Kurose,Y., A sound and complete procedure for a general logic program in non-floundering derivations with respect to the 3-valued stable model semantics, Theoretical Computer Science, 266, pp.489–512, 2001.

[20] Yamasaki,S., Logic programming with default, weak and strict negations, Theory and Practice of Logic Programming, 6, pp.737-749, 2006.