2/9/2010
LRBA 2010

# A Construction of Logic-Constrained Functions with Respect to Awareness

Susumu Yamasaki

Department of Computer Science
Okayama University, Japan

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 which some predicate (with or without classical negation) may denote.

(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.

| Derivability basis | Evocation | Device |
|---|---|---|
| A proof system | Awareness | Function construction |

Table 1: Logical awareness

*Backgrounds*

(1) Henessy-Milner Logic (HML)

Formulae in HML have the following syntax:

$$\varphi ::= \mathrm{tt} \mid \varphi \wedge \psi \mid \neg\varphi \mid < a > \varphi$$

The denotation $[\![\varphi]\!]$ of a formula $\varphi$ on a transition system $\mathcal{T} = (S, \mathcal{A}, \rightarrow)$ is defined as follows:

$$\begin{aligned} [\![\mathrm{tt}]\!] &= S \\ [\![\varphi \wedge \psi]\!] &= [\![\varphi]\!] \cap [\![\psi]\!] \\ [\![\neg\varphi]\!] &= S - [\![\varphi]\!] \\ [\![< a > \varphi]\!] &= \{s \in S \mid \exists t \in S : s \rightarrow^a t \wedge t \in [\![\varphi]\!]\}. \end{aligned}$$

We say that $s$ satisfies $\varphi$ (denoted $s \models \varphi$) if $s \in [\![\varphi]\!]$.

Kucera,A. and Esparza,J., A logical viewpoint on process-algebra, JLC, 13, 6, pp.863–880, 2003.

(2) Action Logic

We define an action sequence as follows:

$$A ::= \varepsilon \mid A; a$$

while we have the following axiom schemas.

(i) $[a_i](\alpha \to \beta) \to ([a_i]\alpha \to [a_i]\beta)$

(ii) $(c)(\alpha \to \beta) \to ((c)\alpha \to (c)\beta)$

(iii) $[\varepsilon](\alpha \to \beta) \to ([\varepsilon]\alpha \to [\varepsilon]\beta)$

(iv) $\Box(\alpha \to \beta) \to (\Box\alpha \to \Box\beta)$

(v) $\Box\alpha \to \alpha$

(vi) $\Box\alpha \to \Box\Box\alpha$

(vii) $[A; a]\alpha \leftrightarrow [A][a]\alpha$

(viii) $\Box\alpha \to [a_i]\alpha$

(ix) $\Box\alpha \to [\varepsilon]\alpha$

(x) $[a_i](c)\alpha \to [a_i]\alpha$

(xi) $[\varepsilon](c)\alpha \to [\varepsilon]\alpha$.

- A modal operator $[a]$ denotes a primitive action, where $[a_1; a_2; \ldots; a_n]$ and $[\varepsilon]$ represent a sequence of actions $a_1, \ldots, a_n$ and the empty sequence of actions, respectively.
- A modality $(c)$ denotes information which is caused as a ramification effect of an action.

Giordano,L. et al., Ramification and causality in a modal action logic, JLC, 10, 5, pp.625–662, 2000.

(3) Hybrid Logic

The formulas of the hybrid modal logic are defined by the grammar

$$S ::= p \mid a \mid S \wedge S \mid S \rightarrow S \mid \bot \mid \Box_i S \mid a : S \mid \forall a S \mid\downarrow a S$$

where $p$ is a propositional symbol and $a$ is a nominal.

A model is a triple $(W, R_1, \ldots, R_m, V)$ where:

- $(W, R_1, \ldots, R_m)$ is a frame, and
- $V$ is a (valuation) function that to each pair consisting of an element of $W$ and a propositional symbol assigns an element of $\{0, 1\}$.

An assignment for a model $(W, R_1, \ldots, R_m, V)$ is a function $g$ that to each nominal assigns an element of $W$. Given assignments $g'$ and $g$, $g' \sim^a g$ means that $g'$ agrees with $g$ on all nominals save possibly $a$.

Given a model $\mathcal{M} = (W, R_1, \ldots, R_m, V)$, the relation

$$\mathcal{M},\, g,\, w \models \phi$$

is defined by induction, where $g$ is an assignment, $w$ is an element of $W$, and $\phi$ is a formula.

$$\begin{array}{rll}
\mathcal{M}, g, w \models p & \text{iff} & V(w,p) = 1 \\
\mathcal{M}, g, w \models a & \text{iff} & w = g(a) \\
\mathcal{M}, g, w \models \phi \wedge \psi & \text{iff} & \mathcal{M}, g, w \models \phi \text{ and } \mathcal{M}, g, w \models \psi \\
\mathcal{M}, g, w \models \phi \rightarrow \psi & \text{iff} & \mathcal{M}, g, w \models \phi \text{ implies } \mathcal{M}, g, w \models \psi \\
\mathcal{M}, g, w \models \bot & \text{iff} & \text{falsum} \\
\mathcal{M}, g, w \models \Box_i \phi & \text{iff} & \text{for any } v \in W \text{ such that } wR_iv, \\
 & & \mathcal{M}, g, v \models \phi \\
\mathcal{M}, g, w \models a : \phi & \text{iff} & \mathcal{M}, g, g(a) \models \phi \\
\mathcal{M}, g, w \models \forall a \phi & \text{iff} & \text{for any } g' \sim^a g, \mathcal{M}, g', w \models \phi \\
\mathcal{M}, g, w \models\!\downarrow a\phi & \text{iff} & \mathcal{M}, g', w \models \phi \text{ where } g' \sim^a g \\
 & & \text{and } g'(a) = w.
\end{array}$$

A formula $\phi$ is valid in a frame if and only if $\mathcal{M} \models \phi$ that is based on the frame in question. A formula $\phi$ is valid in a class of frames if and only if $\phi$ is valid in any frame of the class.

Areces,C. et al., Repairing the interpolation in quantified logic, Annals of Pure and Applied Logic, 123, 287–299, 2003.

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

(4) Sequentiality of Relations

**Syntax**

$$R ::= \varepsilon \mid r \mid RR \mid r \rhd R \mid \bigcirc R \mid R \to R \mid \overline{R}$$

A partial order $\preceq$:

- A member of $O^*$ (the set of sequences on the given set $O$) is a sequence of objects.
- The function $Sem$ assigns a state transition to each sequence of objects so that it gives a meaning of a sequence over objective knowledge.
- In what follows, $\alpha(G)$ is the first object of the sequence $G$ if $G$ is not empty, and $\varepsilon$ if $G$ is empty. $rest(G)$ is the sequence obtained by cutting $\alpha(G)$ off from $G$.

**Definition** For sequences $G, G' \in O^*$, we define a relation $\prec$ ($\subseteq O^* \times O^*$) recursively as follows. $G' \prec G$ if:

(1) $G' = \varepsilon$,

(2) $\alpha(G') = \alpha(G) \Rightarrow rest(G') \prec rest(G)$, or

(3) $\alpha(G') \neq \alpha(G) \Rightarrow G' \prec rest(G)$.

## Semantics

- The semantic function of a relation at each site and/or the internet assigns a transition set between states to each relation expression.
- The copy of a relational expression at a common space to each site is allowable, while the copy of the relation at a site to the common space is restricted.
- If the restriction on the copy would be relaxed, then the transfer of relations from a site to another is possible such that we can see a distributed system where the communications between any two sites are free.

Yamasaki,S. et al., A calculus effectively performing event formation with visualization, Lecture Notes in Computer Science 4759, pp.287–294, 2008.

*Predicate Logic as Programming Language*
R.A.Kowalski (1974)

In Horn (clause) propositional logic:

(Procedure declaration) clause $A \leftarrow B_1, \ldots, B_m \ (m \geq 0)$
(Procedure evocation) goal $\leftarrow C_1, \ldots, C_n \ (n \geq 0)$

Resolution is a deduction caused by the rule:

$$\frac{\leftarrow G_1, A, G_2 \quad A \leftarrow G}{\leftarrow G_1, G, G_2}.$$

Negation as Failure
−Interpretation of Default Negation
K.L.Clark (1978)

Implementation vs. Model Theory: $\frac{\nvdash A}{not\ A}$

By (SLD) resolution with negation as failure
(in propositional logic):

$$\leftarrow A \text{ fails } \Rightarrow \leftarrow not\ A \text{ succeeds},$$
$$\leftarrow A \text{ succeeds } \Rightarrow \leftarrow not\ A \text{ fails}.$$

*Semantics for Logic Programs with Negation-as-Failure*
(3-Valued Model Theory, Alternating Fixpoint Semantics)
M.Fitting (1985) K.Kunen (1987) J.C.Shepherdson (1989) A.Van Gelder (1993) P.M.Dung (1995)

A normal logic program (in propositional logic):
A set of clauses of the form:

$$A \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n$$

**Definition** Let $K \subseteq B_P$ for a normal logic program $P$, where $B_P$ is the Herbrand base of $P$. We define a set:

$$P[K] = \{A \leftarrow B_1, \ldots, B_m \mid \exists(A \leftarrow B_1, \ldots, B_m, not\ C_1, \ldots, not\ C_n) \in P. \; C_1, \ldots, C_n \in K\}.$$

The mapping $S_P$: $2^{B_P} \to 2^{B_P}$ is defined to be

$$S_P(K) = T_{P[K]} \uparrow \omega,$$

where we have the least fixpoint of the mapping $T_Q$

$$T_Q \uparrow \omega = \cup_{i \in \omega} T_Q \uparrow i$$

for a normal logic program $Q$ without negation-as-failure.

The set generated by the mapping $T_Q$:

$$T_Q \uparrow i = \begin{cases} \emptyset & (i = 0), \\ T_Q(T_Q \uparrow (i-1)) & (i > 0). \end{cases}$$

The mapping $T_Q$: $2^{B_Q} \to 2^{B_Q}$ is defined to be

$$T_Q(J) = \{a \mid \exists (a \leftarrow a_1, \ldots, a_n) \in Q. \\ a_1, \ldots, a_n \in J\}$$

for a set $J \subseteq B_Q$.

The alternating mapping $\Theta_P : 2^{B_P} \to 2^{B_P}$ is defined to be

$$\Theta_P(K) = \overline{S_P(\overline{S_P(K)})}.$$

**Definition** Assume that $(I, J)$ is a 3-valued Herbrand interpretation of a normal logic program $P$.

(1) If $J$ is a fixpoint of $\Theta_P$ such that $I = S_P(J)$, we say that $(I, J)$ is a 3-valued stable model of $P$.

(2) If $J$ is the least fixpoint of $\Theta_P$ such that $I = S_P(J)$, we say that $(I, J)$ is the well-founded model of $P$.

e.g. Assume a normal logic program $P$:

$$p \leftarrow not\ q$$
$$q \leftarrow not\ p$$

The sets $\emptyset$, $\{p\}$, $\{q\}$ are the fixpoints of the mapping $\Theta_P$ such that we have three 3-valued stable models:

(i) $(\emptyset, \emptyset)$ (the well-founded model)

(ii) $(\{q\}, \{p\})$

(iii) $(\{p\}, \{q\})$

e.g. Let $P$ be the program containing:

$$p(x) \leftarrow not\ q(x)$$
$$q(x) \leftarrow not\ p(f(x))$$
$$r(a) \leftarrow p(a)$$

We have 3-valued stable models:

(i) $(\emptyset, \{r(f^i(f(a))) \mid i \in \omega\})$ (the well-founded model)

(ii) $(\{q(f^i(a)) \mid i \in \omega\}, \{p(f^i(a)), r(f^i(a)) \mid i \in \omega\})$

(iii) $(\{r(a), p(f^i(a)) \mid i \in \omega\}, \{q(f^i(a)), r(f^i(f(a))) \mid i \in \omega\})$

*A Sound and Complete Procedure*
*with Respect to 3-Valued Stable Model*
S.Yamasaki and Y.Kurose (2001)

Infinite-computation extended from the propositional rule closure of:

(1) $\dfrac{}{Suc_P(\Box; \Delta; \Delta)}$

(2) $\dfrac{Suc_P(\leftarrow G; \Delta_1; \Delta_2) \quad A \leftarrow G \text{ is in } P}{Suc_P(\leftarrow A; \Delta_1; \Delta_2)}$

(3) $\dfrac{not\ A \in \Delta}{Suc_P(\leftarrow not\ A; \Delta; \Delta)}$

(4) $\dfrac{not\ A \notin \Delta_1 \quad Fail_P(\leftarrow A; \Delta_1 \cup \{not\ A\}; \Delta_2)}{Suc_P(\leftarrow not\ A; \Delta_1; \Delta_2)}$

(5) $\dfrac{\text{No clause of the form } A \leftarrow G \text{ is in } P}{Fail_P(\leftarrow A; \Delta; \Delta)}$

(6) $\dfrac{\text{For any } A \leftarrow G_i\ (1 \leq i \leq n)\ Fail_P(\leftarrow G_i; \Delta_i; \Delta_{i+1})}{Fail_P(\leftarrow A; \Delta_1; \Delta_{n+1})}$

(7) $\dfrac{Suc_P(\leftarrow A; \Delta_1; \Delta_2)}{Fail_P(\leftarrow not\ A; \Delta_1; \Delta_2)}$

(I) $\dfrac{Suc_P(\leftarrow G_1; \Delta_1; \Delta_2) \quad Suc_P(\leftarrow G_2; \Delta_2; \Delta_3)}{Suc_P(\leftarrow G_1, G_2; \Delta_1; \Delta_3)}$

(II) $\dfrac{Fail_P(\leftarrow G; \Delta_1; \Delta_2)}{Fail_P(\leftarrow G_1, G, G_2; \Delta_1; \Delta_2)}$

e.g. Let $P$ be the program containing:

$$p(x) \leftarrow not\ q(x)$$
$$q(x) \leftarrow not\ p(f(x))$$
$$r(a) \leftarrow p(a)$$

We can have the relation $Suc_P(\leftarrow r(a); \emptyset; \Delta)$, where

$$\Delta = \{not\ q(f^i(a)) \mid i \in \omega\}.$$

*Distributed Programming (in Propositional Logic) and Non-Monotonic Mapping*

S.Yamasaki (2003, 2004)

1. Negation as Failure through Network

For a distributed logic program $< P_1, \ldots, P_m >$:

$$\frac{\text{A goal} \leftarrow A \text{ fails for all } P_i}{not\ A}$$

e.g. Take the program $< P_1, P_2 >$, where

$$\begin{aligned} P_1 :&\ p \leftarrow not\ q \\ &\ q \leftarrow r \\ P_2 :&\ q \leftarrow p \end{aligned}$$

We see that: $\dfrac{\leftarrow q \text{ fails for both } P_1 \text{ and } P_2}{\leftarrow p \text{ succeeds for } P_1}$.

2. Both Positive and Negative Propositions to Be Communicated

For a distributed logic program $< P_1, \ldots, P_m >$:

$$\frac{\vdash_{P_i} A \quad \text{for some } P_i}{A \ (\text{for any } P_j)}$$

$$\frac{\nvdash_{P_i} A \quad \text{for all } P_i}{not\ A \ \ (\text{for any } P_j)}$$

e.g. For the above program $< P_1, P_2 >$, the goal $\leftarrow p$ cannot succeed even for $P_1$.

(Note) A non-monotonic mapping is associated with the distributed program, to describe the whole behaviour.

e.g. Assume a program $< P_1, P_2, P_3 >$ such that:

$P_1$ $syndrome \leftarrow large\text{-}scale\_e\text{-}mailing$,
$large\text{-}scale\_e\text{-}mailing \leftarrow virus\_A$,
$large\text{-}scale\_e\text{-}mailing \leftarrow virus\_B$,
$virus\_A \leftarrow not\ patch\_A$,
$virus\_B \leftarrow not\ patch\_G$,
$modifiers\_files \leftarrow virus\_A, not\ patch\_A$.

$P_2$ $patch\_A \leftarrow OS\_version\_D$,
$patch\_B \leftarrow OS\_version\_D$,
$OS\_version\_D \leftarrow$ .

$P_3$ $patch\_A \leftarrow tool\_C$.

(1) A goal sequence:

$\leftarrow syndrome$,
$\leftarrow large\text{-}scale\_e\text{-}mailing$,
$\leftarrow virus\_A$,
$\leftarrow not\ patch\_A$.

The goal $\leftarrow patch\_A$ may succeed for the program $P_2$ so that it cannot fail for all the programs $P_1$, $P_2$ and $P_3$.

(2) Another goal sequence:

$\leftarrow syndrome$,
$\leftarrow large\text{-}scale\_e\text{-}mailing$,
$\leftarrow virus\_B$,
$\leftarrow not\ patch\_G$.

The goal $\leftarrow patch\_G$ fails for all the programs $P_1$, $P_2$ and $P_3$. Hence the goal $\leftarrow syndrome$ may succeed through communications of negatives.

*Classical negation vs. negation as failure*

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.

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.

*Coherence principle* (Alferes,J.J., 1995):

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

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.

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.

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

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

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

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 $\leftarrow \neg q$.

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

(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 $\Sigma$ and $\Delta$.

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

*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 $\neg p$.

*Logic-Constrained Function*

*Syntax*

On the assumption of a proof system (say, $\Gamma$), 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 $\Gamma$ (which may possibly be the ELP in the previous section), we mean that:

- If $p$ is derivable from $\Gamma$, then the term $M$ is supported.
- Unless $p$ is derivable from $\Gamma$, then the term $M$ is not supported.

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

*Illustration*

Assume the following functional program.

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

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

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

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

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

*Awareness Ordering*

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

In addition to the standard $\alpha$, $\beta$, and $\eta$ 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 $\lambda$-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

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

*Primary Theorem*

Because:

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

(b) $\alpha$, $\beta$ and $\eta$ conversions are commutative with $\gamma 1$ and $\gamma 2$ conversion applications,

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

**Theorem** If the term $M$ has a normal form, it is unique up to $\alpha$ conversion.