ソフトウェア基礎論　27－8
プログラミング言語　19－8
（1988. 12. 9）

型付きラムダ計算による CCS の定式化と
その効率のよい実行機構

Formulation of CCS with Typed Lambda Calculus and Its Efficient Interpretation Mechanism

堀田英一
Eiichi Horita

NTT ソフトウェア研究所
NTT Software Laboratories,
3-9-11 Midori-Cho, Musashino-Shi, Tokyo 180 Japan

概要： Milner の CCS を型付きラムダ計算を用いて，ある種の多ソート代数の拡張として定式化する．これにより， CCS に現れるパラメータの意味を明かにする．また 新しい関数記号を基の多ソート代数に導入することにより， CCS の本質的な拡張が可能であることを示す．　次に，この体系の実行機構について論ずる．まず実行機構の完全性を定義する．それに基づき，空間的・時間的に効率がよくかつ完全な実行機構を提案し，その実装結果について述べる．

Abstract: First, Milner's CCS is formulated as an extension of a many-sorted algebra with typed lambda calculus. The formulation makes the role of value parameters of CCS clear. Moreover, it is shown that introducing new function symbols to the base algebra extends CCS essentially. Then, interpretation mechanisms of CCS are discussed. The *Completeness* of an interpretation mechanism is defined; a complete interpretation mechanism is proposed, which is efficient with respect to space and time. Finally an imprementation of the mechanism is mentioned.

1 Introduction

This paper consists of two parts. In the first part, Milner's CCS (Calculus of Communicating Systems) (bib:Miln:80) is formulated as an extension of a many-sorted algebra with the typed λ-notation ([Hind 86]) for denoting processes with parameters and the μ-notation ([Bakk 80]) for denoting fixed points. The language resulting from this formulation is named ITCS (Indexed Theory of Communicating Systems). The language ITCS makes the role of value parameters of CCS clear. Moreover, it is shown that introducing new function symbols to the base algebra extends CCS essentially.

In the second part, interpretation mechanisms of ITCS are discussed. First, denotational semantics based on the process domain due to de Bakker and Zucker ([Bakk 82]) is proposed. The intention in discussing interpretation mechanisms is to investigate the problem whether the language ITCS can be interpreted according to the denotational semantics. For that purpose, the notions of *soundness* and *completeness* of an interpretation mechanism of ITCS under the above semantics are defined. Then, an interpretation mechanism is proposed, and its soundness and completeness are shown. Moreover, it is shown the mechanism is efficient with respect to space and time.

Finally, an interpretation system of ITCS, named CLITCS, is mentioned.

The rest of this paper is organized as follows. In Section 2, the language ITCS is proposed. In Section 3, the denotational semantics of ITCS is presented. In Section 4, interpretation mechanisms of ITCS are discussed. In Section 5, the interpretation system CLITCS is mentioned. Finally in Section 6, remaining problems and related works are discussed.

2 Language ITCS

The language ITCS is an indexed or parameterized version of the language TCS ([Hori 88]), and it is a formulation of Milner's CCS with the typed λ-notation for denoting processes with parameters and the μ-notation ([Bakk 80]) for denoting fixed points. Note that the symbol 'μ' is used here for the symbol 'Fix' in [Hori 88] and [Miln 83].

2.1 Signature $\mathcal{S}_{\mathrm{ITCS}}$

Here, a signature means a combination of a set of sorts and a set of function symbols among the sorts.

A signature $\mathcal{S}_{\mathrm{ITCS}}$ is almost the same as that of CCS. The signature $\mathcal{S}_{\mathrm{ITCS}}$ has five sorts, i.e., Pro, IPro, Lab, ILab, Val, which are the sort of processes without parameters, the sort of processes with parameters, the sort of labels of actions , the sort of labels of actions with parameters , and the sort of values passed between processes respectively. We use variables $P_0, P_1, \cdots$ of sort Pro, $Q_0, Q_1, \cdots$ of sort IPro, $\xi_0, \xi_1, \cdots$ of sort Lab, $\eta_0, \eta_1, \cdots$ of sort ILab, and $X_0, X_1, \cdots$ of sort Val. And we denote by $\mathcal{V}_{\mathrm{Pro}}$, $\mathcal{V}_{\mathrm{IPro}}$, $\mathcal{V}_{\mathrm{Lab}}$, $\mathcal{V}_{\mathrm{ILab}}$, and $\mathcal{V}_{\mathrm{Val}}$ the sets of variables of sorts Pro, IPro, Lab, ILab, and Val respectively, and let $\mathcal{V} = \mathcal{V}_{\mathrm{Pro}} \cup \mathcal{V}_{\mathrm{IPro}} \cup \mathcal{V}_{\mathrm{Lab}} \cup \mathcal{V}_{\mathrm{ILab}} \cup \mathcal{V}_{\mathrm{Val}}$.

Definition 2.1 $\mathcal{S}_{\mathrm{ITCS}}$ has the following constant symbols and function symbols. For a constant symbol C, a function symbol F, and sorts $S, S_1, \cdots, S_n$, we denote by "$C : S$" that C's sort is S, and by "$F : (S_1, \cdots, S_n) \to S$" that F's arity and sort are $(S_1, \cdots, S_n)$ and S respectively.

1. A countably infinite set of constant symbols
   $\alpha_0, \alpha_1, \cdots$: Lab,
   and a countably infinite set of constant symbols
   $\beta_0, \beta_1, \cdots$: ILab,
   and a constant symbol
   Stop : Pro.

2. $\mathcal{S}_{\mathrm{ITCS}}$ has the function symbols in Table 1.

2.2 Language ITCS

Notation 2.1 The following notations are used.

1. The word "iff" will be used for "if and only if". The usual logical symbols, "$\wedge$"(and), "$\vee$"(or), "$\neg$"(not), "$\Rightarrow$"(implies), "$\Leftrightarrow$"(iff), "$\forall$"(for all), and "$\exists$"(exists).

2. The usual λ-notation for denoting functions is used. For a set A, a variable x, and an expression $E(x)$, the expression $(\lambda x \in A : E(x))$ denotes the function $\{(x, E(x)) : x \in A\}$.

3. For a set X, we denote by $\sharp(X)$ the cardinality of X. For two sets X and Y, we denote by $(X \to Y)$ the set of functions from X to Y. The set of natural numbers is denoted by ω. For $n \in \omega$, $[n] = \{m \in \omega : 1 \leq m \leq n\}$. The empty sequence is denoted by ϵ.

The language ITCS is an extension of the many-sorted algebra based on the signature $\mathcal{S}_{\mathrm{ITCS}}$ using the the typed λ-notation for denoting processes with parameters and μ-notation ([Bakk 80]) for denoting fixed points.

The language ITCS consists of the terms defined below. We use S-expression notation ([McCa 60]) for expressing terms. First, the set of terms of sort Val, Lab, ILab, written $\mathcal{T}_{\mathrm{Val}}$, $\mathcal{T}_{\mathrm{Lab}}$, and $\mathcal{T}_{\mathrm{ILab}}$ respectively, are defined as in usual many-sorted algebra.

To construct terms of of sort Pro and IPro, μ-operators are used, which have the form

"$\mu_i\ (P_1, \cdots, P_n)$" ($i \in [n]$) or "$\mu\ Q$".

For an expression of the form

"$(\mu_i\ (P_1, \cdots, P_n) \cdots)$"

the inside of the outermost parenthesis is said to be the scope of the μ-operator "$\mu_i\ (P_1, \cdots, P_n)$" and any occurrence of a variable P_i ($i \in [n]$) in the scope is said to be bound by the μ-operator "$\mu_i\ (P_1, \cdots, P_n)$". Similarly, the scope of the μ-operator "$\mu\ Q$" and its variable binding are

Table 1: Function Symbols of $\mathcal{S}_{\text{ITCS}}$

Symbol	Arity	Sort	Meaning	Notation in [Miln 80]
τ	(Pro)	Pro	Silent Action	$\tau.E$
?	(Lab, Pro)	Pro	Input Action	$\xi.E$
!	(Lab, Pro)	Pro	Output Action	$\bar{\xi}.E$
$?_1$	(ILab, IPro)	Pro	Input Action with Parameter	$\eta X.E$
$!_1$	(ILab, Val, Pro)	Pro	Output Action with Parameter	$\bar{\eta}V.E$
+	(Pro, Pro)	Pro	*Nondeterministic Choice*	$E_1 + E_2$
&	(Pro, Pro)	Pro	*Concurrent Composition*	$E_1 \Vert E_2$
/	(Pro, Lab)	Pro	Restriction of Labels of Sort Lab	$E\backslash\xi$
$/_1$	(Pro, ILab)	Pro	Restriction of Labels of Sort ILab	$E\backslash\eta$
$\$$	(Pro, Lab, Lab)	Pro	Renaming of Labels of Sort Lab	$E[\xi_1/\xi_2, \xi_2/\xi_1]$
$\$_1$	(Pro, ILab, ILab)	Pro	Renaming of Labels of Sort ILab	$E[\eta_1/\eta_2, \eta_2/\eta_1]$
If	(Val, Pro, Pro)	Pro	Condional	If V then E_1 else E_2
Apl	(IPro, Val)	Pro	Application	b(V)

defined. Free variables of an expression G, written FV(G), is defined as usual.

The set of terms of sort Pro and IPro written $\mathcal{T}_{\text{Pro}}$ and $\mathcal{T}_{\text{IPro}}$ respectively, are defined as follows.

Definition 2.2
(1) $P \in \mathcal{V}_{\text{Pro}} \Rightarrow P \in \mathcal{T}_{\text{Pro}}$. Stop $\in \mathcal{T}_{\text{Pro}}$.
(2) $Q \in \mathcal{V}_{\text{IPro}} \Rightarrow Q \in \mathcal{T}_{\text{IPro}}$.
(3) $\xi, \xi_1 \in \mathcal{T}_{\text{Lab}} \wedge \eta, \eta_1 \in \mathcal{T}_{\text{ILab}}$
$\wedge E, E_1 \in \mathcal{T}_{\text{Pro}} \wedge V \in \mathcal{T}_{\text{Val}} \wedge F \in \mathcal{F}_{\text{IPro}} \Rightarrow$
$(\tau\ E), (?\ \xi\ E), (!\ \xi\ E), (+\ E\ E_1), (\&\ E\ E_1)$,
$(/\ E\ \xi), (/_1\ E\ \eta)$, $(\$\ E\ \xi\ \xi_1), (\$_1\ E\ \eta\ \eta_1)$,
$(!_1\ \eta\ V\ E), (?_1\ \eta\ F) \in \mathcal{T}_{\text{Pro}}$.
(4) $F \in \mathcal{T}_{\text{IPro}} \wedge V \in \mathcal{T}_{\text{Val}} \Rightarrow (\text{Apl}\ F\ V) \in \mathcal{T}_{\text{IPro}}$.
(5) $E \in \mathcal{T}_{\text{Pro}} \wedge X \in \mathcal{V}_{\text{Val}} \Rightarrow (\lambda\ X\ E) \in \mathcal{T}_{\text{IPro}}$.
(6) $E, E_1 \in \mathcal{T}_{\text{Pro}} \wedge V \in \mathcal{T}_{\text{Val}} \Rightarrow (\text{If}\ V\ E\ E_1) \in \mathcal{T}_{\text{Pro}}$.
(7) $P_1, \cdots, P_n$ are distinct variables $\wedge$
$E_1, \cdots, E_n \in \mathcal{T}_{\text{Pro}} \wedge$
$\forall i \in [n], \forall G[G$ is a sub-expression of E_i
of the form $(\mu_l\ \cdots)$ with some l, or $(\mu\ \cdots) \Rightarrow$
$\forall j \in [n][P_j \notin \text{FV}(G)]\ ] \Rightarrow$
$(\mu_k\ (P_1 \cdots P_n)\ (E_1 \cdots E_n)) \in \mathcal{T}_{\text{Pro}}$
$(k \in [n])$.
(8) $F \in \mathcal{T}_{\text{IPro}} \wedge$
$\forall G[G$ is a sub-expression of F
of the form $(\mu_l\ \cdots)$ with some l, or $(\mu\ \cdots) \Rightarrow$
$Q \notin \text{FV}(G)] \Rightarrow$
$(\mu\ Q\ F) \in \mathcal{T}_{\text{IPro}}$.

For two terms E and G, we mean by $E \equiv G$ that E and G are syntactically identified.

A term of the form "$(\mu_i(P_1 \cdots P_n)\ (E_1\ \cdots\ E_n))$" or "$(\mu\ Q\ F)$" is called a *$\mu$-term*.

We denote by
$E[G_1/P_1, \cdots, G_n/P_n]$
the result of simultaneously replacing free occurrences of P_i with G_i , for each i. When using this notation we always assume that none of the free variables in G_i occur as bound variables in E $(i \in [n])$.

For a sort S, let $\mathcal{T}^{\circ}_{\text{S}}$ be the set of closed terms of sort S. Elements of $\mathcal{T}^{\circ}_{\text{Pro}}$ are said to be ***process specifications*** or ***processes***. Elements of $\mathcal{T}^{\circ}_{\text{IPro}}$ are said to be ***process schemata***.

3 Denotational Semantics of ITCS

3.1 Semantic Domains

The denotational semantics of ITCS is presented based on the process domain proposed by de Bakker and Zucker ([Bakk 82]), as in [Hori 88].

First, let $\mathbf{D}_{\text{Lab}}$, $\mathbf{D}_{\text{ILab}}$, and $\mathbf{D}_{\text{Val}}$ be the domains of sort Lab, ILab, and Val respectively. We assume that $\epsilon \in \mathbf{D}_{\text{Val}}$. Moreover, let

$$\begin{aligned}&\mathbf{D}_{\text{Act}} = \\ &\{?, !\} \times \mathbf{D}_{\text{Lab}} \\ &\cup (\{?, !\} \times \mathbf{D}_{\text{ILab}}) \times \mathbf{D}_{\text{Val}} \cup \{\tau\}.\end{aligned}$$

A function $\ominus : \mathbf{D}_{\text{Act}} \to \mathbf{D}_{\text{Act}}$ is defined by

$$\ominus(a) = \begin{cases} (!, \xi) & \text{if } a = (?, \xi), \\ (?, \xi) & \text{if } a = (!, \xi), \\ ((!, \eta), v) & \text{if } a = ((?, \eta), v), \\ ((?, \eta), v) & \text{if } a = ((!, \eta), v), \\ \tau & \text{if } a = \tau. \end{cases}$$

For a metric space (X, d) and a set of labels Γ, we define a metric d' on $\Gamma \times X$ by

$$d'((\gamma_1, x_1), (\gamma_2, x_2)) = \begin{cases} (1/2) \cdot d(x_1, x_2) & \text{if } \gamma_1 = \gamma_2, \\ 1 & \text{otherwise} \end{cases}$$

for $(\gamma_1, x_1), (\gamma_2, x_2) \in \Gamma \times X$.

Let $\mathcal{P}_{\text{c}}(\Gamma \times X)$ be the set of closed subsets of $(\Gamma \times X, d')$, and d'_{H} be the *Hausdorff metric* induced by d', i.e., a metric defined by

$$\begin{aligned}&d'_{\text{H}}(A, B) \\ &= \max(\sup_{a \in A}[\inf_{b \in B}[d'(a, b)]], \\ &\qquad \sup_{b \in B}[\inf_{a \in A}[d'(a, b)]]),\end{aligned}$$

for $A, B \in \mathcal{P}_{\text{c}}(\Gamma \times X)$.

By convention, $\inf(\emptyset) = 1$ and $\sup(\emptyset) = 0$.

In [Bakk 82], the authors constructed a complete metric space $(\mathbf{P}(\Gamma), d)$ that satisfies the following.

$$(\mathbf{P}(\Gamma), d) \cong (\mathcal{P}_c(\Gamma \times \mathbf{P}(\Gamma)), d'_H), \qquad (1)$$

i.e., $\exists\Phi[\Phi$ is an isometry from$(\mathbf{P}(\Gamma), d)$
onto $(\mathcal{P}_c(\Gamma \times \mathbf{P}(\Gamma)), d'_H)]$.

Let the domain of sort Pro, written $\mathbf{D}_{\mathrm{Pro}}$, be the metric space $\mathbf{P}(\mathbf{D}_{\mathrm{Act}})$ (the $\mathbf{P}(\Gamma)$ for $\Gamma = \mathbf{D}_{\mathrm{Act}}$).

For $p \in \mathbf{P}(\mathbf{D}_{\mathrm{Act}})$, we identify $\Phi(p)$ with p.

Finally, let the domain of sort IPro, written $\mathbf{D}_{\mathrm{IPro}}$, be the function space $(\mathbf{D}_{\mathrm{Val}} \rightarrow \mathbf{D}_{\mathrm{IPro}})$.

3.2 Denotational Interpretation of Function Symbols

We define interpretations of function symbols in Section 1 as functions among the above domains. The interpretation of Stop, τ, ?, !, $?_1$, $!_1$, +, and & are written stop, $\tilde{\tau}$, $\tilde{?}$, $\tilde{!}$, $\tilde{?}_1$, $\tilde{!}_1$, $\tilde{+}$, and $\tilde{\&}$ respectively, and defined as follows. Function symbols /, $/_1$, $\$$, and $\$_1$ are omitted here. Their interpretations can be defined similarly to that of '?'.

Definition 3.1 **(1)** stop $= \emptyset$.
(2) $\tilde{\tau} = (\lambda p \in \mathbf{D}_{\mathrm{Pro}} : \{(\tau, p)\})$.
(3) $\tilde{?} = (\lambda(\gamma, p) \in \mathbf{D}_{\mathrm{Lab}} \times \mathbf{D}_{\mathrm{Pro}} : \{((?, \gamma), p)\})$.
(4) $\tilde{!} = (\lambda(\gamma, p) \in \mathbf{D}_{\mathrm{Lab}} \times \mathbf{D}_{\mathrm{Pro}} : \{((!, \gamma), p)\})$.
(5) $\tilde{?}_1 = (\lambda(\gamma, q) \in \mathbf{D}_{\mathrm{ILab}} \times \mathbf{D}_{\mathrm{IPro}} :$
$\{(((?, \gamma), v), q(v)) : v \in \mathbf{D}_{\mathrm{Val}}\})$.
(6) $\tilde{!}_1 = (\lambda(\gamma, v, p) \in \mathbf{D}_{\mathrm{ILab}} \times \mathbf{D}_{\mathrm{Var}} \times \mathbf{D}_{\mathrm{IPro}} :$
$\{(((!, \gamma), v), p) : v \in \mathbf{D}_{\mathrm{Val}}\})$.
(7) $\tilde{+} = (\lambda(p_1, p_2) \in (\mathbf{P}(\tilde{A}))^2 : p_1 \cup p_2)$.

Definition 3.2 In [Bakk 82], it was shown that there is a function $\tilde{\&} : \mathbf{D}_{\mathrm{Pro}} \times \mathbf{D}_{\mathrm{Pro}} \rightarrow \mathbf{D}_{\mathrm{Pro}}$ satisfying the following property. Let $\tilde{\&}$ be the interpretation of &.

$$\begin{aligned}
&\forall p_1, p_2 \in \mathbf{D}_{\mathrm{Pro}} \\
&[\ \tilde{\&}(p_1, p_2) = \\
&\quad \{(\gamma, \tilde{\&}(p'_1, p_2)) : (\gamma, p'_1) \in p_1\}^{\mathrm{C}} \\
&\quad \cup \{(\gamma, \tilde{\&}(p'_2, p_1)) : (\gamma, p'_2) \in p_2\}^{\mathrm{C}} \\
&\quad \cup \{(\tau, \tilde{\&}(p'_1, p'_2)) : (\gamma_1, p'_1) \in p_1 \\
&\qquad \wedge (\gamma_2, p'_2) \in p_2 \\
&\qquad \wedge \tau \neq \gamma_1 = \ominus(\gamma_2)\}^{\mathrm{C}}].
\end{aligned}$$

Here, for a set X, X^{C} is the topological closure of X.

3.3 Denotational Interpretation of Terms

Let $\mathcal{T}^\circ$ be the set of closed terms of ITCS, and let $\mathbf{D} = \mathbf{D}_{\mathrm{Val}} \cup \mathbf{D}_{\mathrm{Lab}} \cup \mathbf{D}_{\mathrm{ILab}} \cup \mathbf{D}_{\mathrm{Pro}} \cup \mathbf{D}_{\mathrm{IPro}}$. In [Hori 88], it was shown that there is an semantic function $\mathcal{M} : \mathcal{T}^\circ \rightarrow \mathbf{D}$ which preserves sorts and satisfies the following conditions. We write $[\![E]\!]$ for $\mathcal{M}(E)$.

(1) For each function symbol F with arity $(S_1, \cdots, S_n)$ and sort Pro,

$$\begin{aligned}
&\forall T_1 \in \mathcal{T}^\circ_{S_1}, \cdots, \forall T_n \in \mathcal{T}^\circ_{S_n} \\
&[[\![F(T_1, \cdots, T_n)]\!] = f([\![T_1]\!], \cdots, [\![T_n]\!])].
\end{aligned}$$

Here f is the interpretation of the function symbol F defined above.

(2) For each μ-term

$$E'_i \equiv (\mu_i(P_1 \cdots P_n)\ (E_1\ \cdots\ E_n)),$$

it holds that

$$[\![E'_i]\!] = [\![E_i[E'_1/P_1, \cdots, E'_n/P_n]]\!], \ (i \in [n]).$$

Similar proposition holds for each μ-term of the form "$(\mu\ Q\ E)$", also.

4 Complete Interpretation Mechanism of ITCS

In this section, we assume the language ITCS has a constant symbol $\mathcal{C}(v)$, called the *name* of v, for each $v \in \mathbf{D}_{\mathrm{Val}}$. It is assumed that $[\![\mathcal{C}(v)]\!] = v$.

4.1 Completeness of an Interpretation Mechanism

We discuss interpretation mechanisms of the sort which reduce parallel execution to nondeterministic sequential execution. Moreover, we assume that an interpreter of ITCS has the procedural framework shown in Figure 1. Here, $E \in \mathcal{T}^\circ_{\mathrm{Pro}}$ and Ability $\in (\mathbf{D}_{\mathrm{Act}} \times \mathcal{T}^\circ_{\mathrm{Pro}}) \cup \{\perp\}$, and for Ability $= (a, E)$, act(Ability) $= a$ and next(Ability) $= E$.

```
procedure interpret (E)
  var Ability;
begin
  Ability := $M$(E, $\tilde{\rho}$(), $\tilde{\delta}$());
  while (Ability $\neq \perp$) do
  begin
    execute(act(Ability));
    Ability := $M$(next(Ability), $\tilde{\rho}$(), $\tilde{\delta}$())
  end
end.
```

Figure 1: Framework of an Interpreter

The function $\tilde{\rho}$ is a 0-ary function returning an environment which determines what actions are executable. The function $\tilde{\delta}$ is a 0-ary function returning an oracle $\in (\omega \rightarrow \{0, 1\})$ which determines what alternative is chosen when there are several alternatives to be chosen nondeterministically. The function M is called the *interpretation function* of the interpreter, whose range is $(\mathbf{D}_{\mathrm{Act}} \times \mathcal{T}^\circ_{\mathrm{Pro}}) \cup \{\perp\}$; the mechanism computing M is called the interpretation mechanism.

Let $\mathbf{D}_{\mathrm{Port}} = (\{\tilde{?}, \tilde{!}\} \times \mathbf{D}_{\mathrm{Lab}}) \cup (\{\tilde{?}, \tilde{!}\} \times \mathbf{D}_{\mathrm{ILab}}) \cup \{\tau\}$. For $a \in \mathbf{D}_{\mathrm{Act}}$, port$(a) \in \mathbf{D}_{\mathrm{Port}}$ is defined by

$$\mathrm{port}(a) = \begin{cases} \gamma' & \text{if } a = (\gamma', v) \\ & \qquad \in (\{?, !\} \times \mathbf{D}_{\mathrm{ILab}}) \times \mathbf{D}_{\mathrm{Val}}, \\ a & \text{otherwise.} \end{cases}$$

We use δ ranging over the set of oracles, written $\mathbf{Orac} = (\omega \rightarrow \{0, 1\})$.

Moreover, we use variable ρ ranging over the set of environments, written $\mathbf{Env} = (\mathbf{D}_{\mathrm{Port}} \rightarrow \mathbf{D}_{\mathrm{Val}} \cup \{\bot\})$.

An action a is executable under an enviorenment ρ only if $\rho(a) \neq \bot$. Futhermore, in the case
$\exists \eta \in \mathbf{D}_{\mathrm{ILab}}[\mathrm{port}(a) = (?, \eta)]$,
ρ specifies $\rho((?, \eta))$ as the value read in the action.

To be precise, we define the notion *enabled* as follows.

Definition 4.1 Let $a \in \mathbf{D}_{\mathrm{Act}}$ and $\rho \in \mathbf{Env}$.

1. In the case $\exists \eta \in \mathbf{D}_{\mathrm{ILab}}[\mathrm{port}(a) = (?, \eta)]$, the action a is *enabled* under the environment ρ (written $\mathrm{enab}(a, \rho)$) iff
   $a = (\mathrm{port}(a), \rho(\mathrm{port}(a)))$.

   That is, the action with the port $(?, \eta)$ is enabled, iff $\rho(\mathrm{port}(a)) \neq \bot$ and the input value read in the action is $\rho(\mathrm{port}(a))$.

2. Otherwise, $\mathrm{enab}(a, \rho)$ iff $\rho(\mathrm{port}(a)) \neq \bot$.

We define the notions *soundness* and *completeness* of an interpreter by the property of the interpretation function M.

Definition 4.2 *(Soundness and Completeness)*
Let $\mathcal{U} \subseteq \mathcal{T}^{\circ}_{\mathrm{Pro}}$, and let
$M : \mathcal{U} \times \mathbf{Env} \times \mathbf{Orac} \rightarrow (\mathbf{D}_{\mathrm{Act}} \times \mathcal{U}) \cup \{\bot\}$.

(1) M is *sound* for the class $\mathcal{U}$ iff
$\forall E \in \mathcal{U}, \forall \rho \in \mathbf{Env}, \forall \delta \in \mathbf{Orac}$
$[\mathrm{enab}(\mathrm{act}(\mathrm{Ability}), \rho) \wedge$
$(\mathrm{act}(\mathrm{Ability}), [\![\mathrm{next}(\mathrm{Ability})]\!]) \in [\![E]\!]]$.
Here $\mathrm{Ability} = M(E, \rho, \delta)$.

(2) M is *deterministically complete* for the class $\mathcal{U}$ iff
$\forall E \in \mathcal{U}, \exists (a, p) \in [\![E]\!][\mathrm{enab}(\mathrm{port}(a), \rho)] \Rightarrow$
$\forall \delta [M(E, \rho, \delta) \neq \bot]$.

(3) M is *nondeterministically complete* for the class $\mathcal{U}$ iff
$\forall E \in \mathcal{U}, \forall (a, p) \in [\![E]\!][\mathrm{enab}(\mathrm{port}(a), \rho) \Rightarrow$
$\exists \delta [(\mathrm{act}(\mathrm{Ability}), [\![\mathrm{next}(\mathrm{Ability})]\!]) = (a, p)]]$.
Here $\mathrm{Ability} = M(E, \rho, \delta)$.

4.2 Guardedly Well Defined Terms

We define the class of *guardedly well-defined processes*, written $\mathbf{GWD} \subseteq \mathcal{T}^{\circ}_{\mathrm{Pro}}$, as a generalization of that in [Miln 80].

Definition 4.3 Let $\mathcal{U} \subseteq \mathcal{T}^{\circ}_{\mathrm{Pro}}$.

1. The class $\mathbf{B}(\mathcal{U})$ is the set of terms consisting of a guard (e.g., "? ξ") and an element of $\mathcal{U}$. That is, it is defined as follows.
   $\mathbf{B}(\mathcal{U}) =$
   $\{\mathrm{Stop}\} \cup$
   $\{(?\ \xi\ E), (!\ \xi\ E), (\tau\ E) : \xi \in \mathcal{T}^{\circ}_{\mathrm{Lab}} \wedge E \in \mathcal{U}\} \cup$
   $\{(!_1\ \eta\ V\ E) : \eta \in \mathcal{T}^{\circ}_{\mathrm{ILab}} \wedge V \in \mathcal{T}^{\circ}_{\mathrm{Val}} E \in \mathcal{U}\} \cup$
   $\{(?_1\ \eta\ F) :$
   $\eta \in \mathcal{T}^{\circ}_{\mathrm{ILab}} \wedge \forall V \in \mathcal{T}^{\circ}_{\mathrm{Val}}[(\mathrm{Apl}\ F\ V) \in \mathcal{U}]\}$.

2. The class $\mathbf{C}(\mathcal{U})$ is the set of terms consisting of elements of $\mathcal{U}$ and operations '+', '&', 'If', 'Apl', '$\mu_i(P_1 \cdots P_n)$', and '$\mu\ Q$'. That is, it is defined as follows.
   $\mathbf{C}(\mathcal{U}) =$
   $\mathcal{U} \cup \{(+\ E_1\ E_2) : E_1, E_2 \in \mathcal{U}\} \cup$
   $\{(\&\ E_1\ E_2) : E_1, E_2 \in \mathcal{U}\} \cup$
   $\{(\mathrm{If}\ V\ E_1\ E_2) : ([\![V]\!] \neq \epsilon \wedge E_1 \in \mathcal{U}) \vee$
   $([\![V]\!] = \epsilon \wedge E_2 \in \mathcal{U})\} \cup$
   $\{(\mathrm{Apl}\ (\lambda\ X\ E)\ V) : E[V/X] \in \mathcal{U}\} \cup$
   $\{(\mu_i(P_1 \cdots P_n)(E_1 \cdots E_n)) :$
   $E_i[E'_1/P_1, \cdots, E'_n/P_n] \in \mathcal{U}$, where
   $E'_j = (\mu_j\ (P_1 \cdots P_n)(E_1 \cdots E_n))$,
   $j \in [n]\} \cup$
   $\{(\mathrm{Apl}\ (\mu\ Q\ F)\ V) :$
   $(\mathrm{Apl}\ F[(\mu\ Q\ F)/Q]\ V) \in \mathcal{U}\}$.

3. The class $\mathbf{G}(U)$ is the smallest set of terms which contains $\mathbf{B}(\mathcal{U})$ and closed under the operations '+', '&', 'If', 'Apl', '$\mu_i(P_1 \cdots P_n)$', and '$\mu\ Q$'.

   That is, it is defined as follows.

   First, for $n \in \omega$, $\mathbf{G}_n(\mathcal{U})$ is defined by
   $\mathbf{G}_0(\mathcal{U}) = \mathbf{B}(\mathcal{U})$,
   $\mathbf{G}_{n+1}(\mathcal{U}) = \mathbf{C}(\mathbf{G}_n(\mathcal{U}))$.
   Then, let $\mathbf{G}(U) = \bigcup_{n \in \omega}[\mathbf{G}_n(\mathcal{U})]$.

4. The class $\mathbf{GWD}$ is the largest subset $\mathcal{W}$ of $\mathcal{T}^{\circ}_{\mathrm{Pro}}$ satisfying $\mathcal{W} \subseteq \mathbf{G}(\mathcal{W})$.

   That is, it is defined as follows.

   First, for $n \in \omega$, GWD_n is defined inductively by
   $\mathrm{GWD}_0 = \mathcal{T}^{\circ}_{\mathrm{Pro}}$,
   $\mathrm{GWD}_{n+1} = \mathbf{G}(\mathrm{GWD}_n)$.
   Then, let $\mathbf{GWD} = \bigcap_{n \in \omega}[\mathrm{GWD}_n]$.

Lemma 4.1 $\quad \mathbf{G}(\mathbf{GWD}) = \mathbf{GWD}$.

Proof. $\mathbf{G}(\mathbf{GWD}) = \mathbf{G}(\bigcap_{n \in \omega}[\mathbf{GWD}_n])$
$= \bigcap_{n \in \omega}[\mathbf{G}(\mathbf{GWD}_n)]$
$= \bigcap_{n \in \omega}[\mathbf{GWD}_{n+1}]$
$= \bigcap_{n \in \omega}[\mathbf{GWD}_n] = \mathbf{GWD}$. ■

Definition 4.4 For $E \in \mathbf{GWD}$, $\deg_{\mathbf{GWD}}(E)$ is defined by

$$\deg_{\mathbf{GWD}}(E) = \min(\{n : E \in \mathbf{G}_n(\mathbf{GWD})\}).$$

In [Miln 80], a process p_i defined by
$\{p_j \Leftarrow E_j(p_1, \cdots, p_n)\}_{j \in [n]}$
is said to be *guardedly well defined* iff there is no infinite sequence $(j(k))_{j \in \omega}$ such that

$$j(0) = i \wedge \forall k [p_{j(k+1)} \text{ is unguarded in } E_{j(k)}]. \qquad (2)$$

As the next lemma shows, our definition of guardedly well-definedness is a generalization of that in [Miln 80].

Lemma 4.2 *If p_i is guardedly well-defined in the sense of [Miln 80], then $p_i \in \mathbf{GWD}$ $(i \in [n])$.*

Proof. We can replace p_j's in E_i by E_j's so that all occurrences of p_j's is guarded in E_i', where E_i' is the result of replacement ($i \in [n]$). Hence,

$p_i \in \mathbf{G}(\{p_1, \cdots, p_n\})$. ($i \in [n]$).

So,

$\forall i \in [n][p_i \in \mathrm{GWD}_n] \Rightarrow$
$\forall i \in [n][p_i \in \mathrm{GWD}_{n+1}]$.

Thus, by induction,

$\forall n \in \omega[p_1 \in \mathrm{GWD}_n]$, i.e.,
$p_i \in \mathbf{GWD}$ ($i \in [n]$). ■

Remark 4.1 Note that for a process definition without a parameter, we can decide whether the Condition (2) holds or not. However, it seems that we cannot decide it for a process definition with a parameter. Moreover, the notion *guardedly well-definedness* is not syntactical here since the semantic notion $[\![V]\!]$ is involved in the definition, while it is syntactical in [Miln 80].

The next lemma follows from the definition of **GWD**.

Lemma 4.3

$$\begin{aligned}
&\forall E_1, E_2 \in \mathbf{GWD}\\
&[\tilde{\&}([\![E_1]\!], [\![E_2]\!]) =\\
&\{(\gamma, \tilde{\&}(p_1', p_2)) : (\gamma, p_1') \in [\![E_1]\!]\}\\
&\cup \{(\gamma, \tilde{\&}(p_2', p_1)) : (\gamma, p_2') \in [\![E_2]\!]\}\\
&\cup \{(\tau, \tilde{\&}(p_1', p_2')) : (\gamma_1, p_1') \in [\![E_1]\!]\\
&\qquad \wedge (\gamma_2, p_2') \in [\![E_2]\!] \wedge \tau \neq \gamma_1 = \ominus(\gamma_2)\}].
\end{aligned}$$

That is, there is no need to take closures for
$E_1, E_2 \in \mathbf{GWD}$ (cf. Definition 3.2).

4.3 Simple Interpretation Mechanism by Enumeration

To interpret a specification a notion "abilities" is introduced. It is defined as follows. Here, we omit IProc sort and the τ-operation for simplicity.

Definition 4.5 Let $E \in \mathbf{GWD}$. abilities(E) is defined by induction of $\deg_{\mathrm{GWD}}(E)$ as in Figure 2. In the figure, M-expression notation of [McCa 60] is used for defining a function recursively. We can compute abilities(E) by the usual recursive function call mechanism.

Then an interpretation mechanism M_0 of ITCS terms using abilities(E) is presented in Figure 3.

This simple interpretation mechanism can be extended so as to handle IProc sort also, but when it handles IProc sort it requires large capacity of memory to emumerate abilities as is shown in Section 4.5. We propose another interpretation mechanism which is more efficient than M_0. Moreover, its soundness and completeness is proved.

4.4 Interpretation Mechanism by Means of Lazy Evaluation

In this section, another interpretation mechanism M_1 is presented, and the soundness and completeness of M_1 is shown. We omit the τ-operation for simplicity, and an interpretation mechanism M_1 for the sub-language without τ is presented. The interpretation mechanism for the full ITCS language M_1^τ is more complex than M_1, but it has the same basic structure as M_1 and its soundness and completeness are shown similarly.

```
abilities[$E$] =
 [$E = (?\ \xi\ E') \Rightarrow \{((?\ \xi)\ E')\}$;
  $E = (!\ \xi\ E') \Rightarrow \{((!\ \xi)\ E')\}$;
  $E = (+\ E_1\ E_2) \Rightarrow$
      abilities($E_1$) $\cup$ abilities($E_2$);
  $E = (\&\ E_1\ E_2) \Rightarrow$
     $\{(\gamma\ (\&\ E''\ E_2))$ :
        $(\gamma\ E'') \in$ abilities($E_1$)$\} \cup$
     $\{(\gamma\ (\&\ E_1\ E''))$ :
        $(\gamma\ E'') \in$ abilities($E_2$)$\}$;
  $E = (\mu_i(P_1 \cdots P_n)\ (E_1 \cdots E_n)) \Rightarrow$
       abilities[$E_i[E_1'/P_1; \cdots; E_n'/P_n]$]
       (where
          $E_j' = (\mu_j\ (P_1 \cdots P_n)\ (E_1 \cdots E_n))$
          ($j \in [n]$))].
```

Figure 2: The Definition of abilities

```
function $M_0$(E, $\rho$, $\delta$)
   var Abilities, Ability;
begin
   $M_0 := \bot$;
   Abilities := abilities(E);
   while ((Abilities $\neq \emptyset$) and $M_0 \neq \bot$)
   do
   begin
      (Choose one of the abilities
        according to the oracle $\delta$,
        and assign it to Ability);
      if
         (Ability is enabled
           uder the environment $\rho$)
      then
         $M_0$ := Ability
   end
end.
```

Figure 3: Interpretation Mechanism M_0 by Enumeration

Definition 4.6 The mechanism M_1 is presented in Figure 4. For $(E, \rho, \delta) \in \mathbf{GWD} \times \mathbf{Env} \times \mathbf{Orac}$, M_1 returns an element of $\mathbf{D}_{\mathrm{Action}} \times \mathbf{GWD}$. We can show the termination of the computation by induction on $\deg_{\mathrm{GWD}}(E)$.

In the figure, M-expression notation is used as in Figure 2. The computation of $M_1[E, \rho, \delta]$ is executed by the usual recursive function call mechanism. For $n \in \omega$, $\mathrm{rest}[\delta; n] = (\delta[i+n])_{i \in \omega}$, $\delta^+ = \mathrm{rest}[\delta; 1]$, and count$[E; \rho; \delta]$ is the number of references to the oracle δ for computing $M_1[E; \rho; \delta]$. In an implementation of the mechanism M_1, an oracle is represented by a stream or a 0-ary function. If an oracle is represented by a stream, a reference to the oracle is represented by a read from the stream. If an oracle is represented by a 0-ary function, a reference to the oracle is represented by a call of the function.

Theorem 4.1 **(1)** M_1 *is sound for* **GWD.**
(2) M_1 *is deterministically complete for* **GWD.**
(3) M_1 *is nondeterministically complete for* **GWD.**

Proof. We prove only (3) here. The statement (1) and (2) are proved similarly.

It is sufficient to prove that the following formula holds for every $E \in \mathbf{GWD}$.

$$\begin{array}{l} \forall(a,p) \in \llbracket E \rrbracket[\text{enab}(\text{port}(a),\rho) \Rightarrow \\ \qquad \exists\delta[\text{act}(M_1(E,\rho,\delta)) = a \wedge \\ \qquad \llbracket \text{next}(M_1(E,\rho,\delta)) \rrbracket = p]]. \end{array} \tag{3}$$

We prove Formula (3), by induction on $\text{deg}_{\mathbf{GWD}}(E)$.
Step 1. If $\text{deg}_{\mathbf{GWD}}(E) = 0$, there are six cases, i.e., E has one of the following forms.

Stop, $(?\ \xi\ E')$, $(!\ \xi\ E')$, $(\tau\ E')$, $(!_1\ \eta\ V\ E')$, and $(?_1\ \eta\ F)$.

Here $V \in \mathcal{T}^{\circ}_{\text{Val}}$, $E' \in \mathbf{GWD}$, $\xi \in \mathcal{T}^{\circ}_{\text{Lab}}$, $\eta \in \mathcal{T}^{\circ}_{\text{ILab}}$, and $F \in \mathcal{T}^{\circ}_{\text{IPro}}$ such that $\forall V \in \mathcal{T}^{\circ}_{\text{Val}}[(\text{Apl}\ F\ V) \in \mathbf{GWD}]$.

We prove in the case $E = (?_1\ \eta\ F)$. In the other five cases, the statement is proved similarly.

In this case,

$$\llbracket E \rrbracket = \{(((?,\llbracket \eta \rrbracket),v),f(v)) : v \in \mathbf{D}_{\text{Val}}\},$$

where $f = \llbracket F \rrbracket$.
Let $(a,p) \in \llbracket E \rrbracket$. Then,

$$\exists v \in \mathbf{D}_{\text{Val}}[a = ((?,\llbracket \eta \rrbracket),v) \wedge p = f(v)].$$

It follows from the definition of 'enab' and the assumption $\text{enab}(\text{port}(a),\rho)$ that $v = \rho(\text{port}(a))$.
By the definition of M_1,

$$\begin{array}{l} M_1(E,\rho,\delta) \\ = (((?,\llbracket \eta \rrbracket),\rho(\text{port}(a))),(\text{Apl}\ F\ \mathcal{C}(\rho(\text{port}(a))))) \\ = (((?,\llbracket \eta \rrbracket),v),(\text{Apl}\ F\ \mathcal{C}(\rho(\text{port}(a))))). \end{array}$$

Hence,

$$\begin{array}{l} \text{act}(M_1(E,\rho,\delta)) = a \wedge \\ \llbracket \text{next}(M_1(E,\rho,\delta)) \rrbracket = p. \end{array}$$

Step 2. We assume, as an induction hypothesis, that Formula (3) holds for every $E' \in \mathbf{GWD}$ with $\text{deg}_{\mathbf{GWD}}(E') \leq n$. Let $E \in \mathbf{GWD}$ with $\text{deg}_{\mathbf{GWD}}(E) = n+1$. There are six cases, i.e., E has one of the following forms.

$(+\ E_1\ E_2), (\&\ E_1\ E_2), (\text{If}\ V\ E_1\ E_2), (\text{Apl}\ (\lambda\ X\ E')\ V)$, $(\mu_i(P_1 \cdots P_n)(E_1 \cdots E_n))$, and $(\text{Apl}\ (\mu\ Q\ F)\ V)$.

We prove in the case $E = (\&\ E_1\ E_2)$. In the other five cases, the statement is proved similarly.

When the τ-operation is omitted, it follows from the Lemma 4.3 that

$$\begin{array}{l} \llbracket E \rrbracket = \{(\gamma,\tilde{\&}(p'_1,p_2)) : (\gamma,p'_1) \in \llbracket E_1 \rrbracket\} \cup \\ \{(\gamma,\tilde{\&}(p'_2,p_1)) : (\gamma,p'_2) \in \llbracket E_2 \rrbracket\}. \end{array}$$

Let $(\gamma,p) \in \llbracket E \rrbracket$. Then there are two cases, i.e.,

$$\exists(\gamma,p'_1) \in \llbracket E_1 \rrbracket[p = \tilde{\&}(p'_1,\llbracket E_2 \rrbracket)] \text{ or}$$

$$\exists(\gamma,p'_2) \in \llbracket E_2 \rrbracket[p = \tilde{\&}(p'_2,\llbracket E_1 \rrbracket)].$$

We prove in the first case. The statement is proved similarly in the second case.

Suppose that $\rho(\text{port}(\gamma)) \neq \perp$. Then, by the induction hypothesis for E_1, it holds that

$$\exists\delta[M_1(E_1,\rho,\delta) = (\gamma,E'_1) \wedge \llbracket E'_1 \rrbracket = p'_1]$$

Let $\delta' = (0,\delta(0),\delta(1),\cdots)$. By Definition 4.6,

$$\begin{array}{l} M_1(E,\rho,\delta') = (\text{act}(A),(\&\ \text{next}(A)\ E_2)) \\ = (\gamma,(\&\ E'_1\ E_2)), \\ \text{where } A = M_1(E_1,\rho,\delta). \end{array}$$

Moreover

$$\llbracket(\&\ E'_1\ E_2)\rrbracket = \tilde{\&}(\llbracket E'_1 \rrbracket,\llbracket E_2 \rrbracket) = \tilde{\&}(p'_1,\llbracket E_2 \rrbracket) = p.$$

Hence, Formula (3) holds for $E = (\&\ E_1\ E_2)$.

Thus, by induction, Formula (3) holds for every $E \in \mathbf{GWD}$. ■

4.5 Comparison between the Two Mechanisms

The mechanism M_1 is more efficient than M_0 especially with respect to space.

For example, let F be a process schema defined by

$$\begin{array}{l} F \equiv \\ (\mu\ Q \\ \ (\lambda\ (X_0\ X_1\ X_2) \\ \quad (\text{If}\ (=\ X_1\ 0) \\ \qquad (!\ B\ X_2\ (\text{Apl}\ (Q\ X_0\ X_0\ \text{Nil}))) \\ \qquad (+\ (\text{Apl}\ Q\ (X_0\ (-\ X_1\ 1)\ (\text{cons}\ 0\ X_2))) \\ \qquad\quad (\text{Apl}\ Q\ (X_0\ (-\ X_1\ 1)\ (\text{cons}\ 1\ X_2))) \\ \qquad)))). \end{array}$$

Then,

$$\begin{array}{l} \text{abilities}[(\text{Apl}\ F\ (N\ N\ \text{Nil}))] = \\ \{(((!\ B)\ L)\ (\text{Apl}\ F\ (N\ N\ \text{Nil}))) : L \in ([N] \rightarrow \{0,1\})\}. \end{array}$$

Hence,

$$\sharp(\text{abilities}[(\text{Apl}\ F\ (N\ N\ \text{Nil}))]) = 2^N.$$

On the other hand, when M_1 is implemented by the usual recursive function call mechanism, the evaluation of

$$M_1[(\text{Apl}\ F\ (N\ N\ \text{Nil})),\rho,\delta]$$

is reduced to the evaluation of

$$\begin{array}{l} M_1[(\text{Apl}\ F\ (X_0\ (-\ X_1\ 1)\ (\text{cons}\ X_2\ 0))),\rho,\delta^+] \text{ or} \\ M_1[(\text{Apl}\ F\ (X_0\ (-\ X_1\ 1)\ (\text{cons}\ X_2\ 1))),\rho,\delta^+]. \end{array}$$

If the evaluation of $M_1[(\text{Apl}\ F\ (N\ N\ \text{Nil})),\rho,\delta]$ is reduced to the evaluation of the first form, then the second form is not evaluated at that time, but only stored for future evaluation. Hence, the memory capacity needed is proportional to N.

In general, for $E \in \mathbf{GWD}$, the memory capacity needed for evaluation of $M_0[E,\rho,\delta]$ is proportional to 2^d, while the capacity needed for evaluation of $M_1[E,\rho,\delta]$ is proportional to d, where $d = \text{deg}_{\mathbf{GWD}}(E)$.

$$
\begin{array}{l}
M_1[E;\rho;\delta] = \\
[E = \mathrm{Stop} \Rightarrow \bot; \\
\;\; E = (?\ \xi\ E') \Rightarrow [\rho[(?\ \llbracket\xi\rrbracket)] \neq \bot \Rightarrow ((?\ \llbracket\xi\rrbracket)\ E'); t \Rightarrow \bot]; \\
\;\; E = (!\ \xi\ E') \Rightarrow [\rho[(!\ \llbracket\xi\rrbracket)] \neq \bot \Rightarrow ((!\ \llbracket\xi\rrbracket)\ E'); t \Rightarrow \bot]; \\
\;\; E = (?_1\ \eta\ F) \Rightarrow [\rho[(?\ \llbracket\eta\rrbracket)] \neq \bot \Rightarrow (((?\ \llbracket\eta\rrbracket)\ \rho[(?\ \llbracket\eta\rrbracket)])\ (\mathrm{Apl}\ F\ \mathcal{C}[\rho[(?\ \llbracket\eta\rrbracket)]])); t \Rightarrow \bot]; \\
\;\; E = (!_1\ \eta\ V\ E') \Rightarrow [\rho[(!\ \llbracket\eta\rrbracket)] \neq \bot \Rightarrow (((!\ \llbracket\eta\rrbracket)\ \llbracket V\rrbracket)\ E'); t \Rightarrow \bot]; \\
\;\; E = (+\ E_1\ E_2) \Rightarrow \\
\qquad [\delta[0] = 0 \Rightarrow [M_1[E_1;\rho;\delta] \neq \bot \Rightarrow M_1[E_1;\rho;\delta]; t \Rightarrow M_1[E_2;\rho;\mathrm{rest}[\delta;\mathrm{count}[E_1;\rho;\delta^+]]]]; \\
\qquad\;\; \delta[0] = 1 \Rightarrow [M_1[E_2;\rho;\delta^+] \neq \bot \Rightarrow M_1[E_2;\rho;\delta^+]; \\
\qquad\qquad\quad t \Rightarrow M_1[E_2;\rho;\mathrm{rest}[\delta;\mathrm{count}[E_1;\rho;\delta^+]]]]]; \\
\;\; E = (\&\ E_1\ E_2) \Rightarrow \\
\qquad [\delta[0] = 0 \Rightarrow \\
\qquad\qquad\qquad [M_1[E_1;\rho;\delta^+] \neq \bot \Rightarrow (\mathrm{act}[M_1[E_1;\rho;\delta^+]]\ (\&\ \mathrm{next}[M_1[E_1;\rho;\delta^+]\ E_2])); \\
\qquad\qquad\qquad\;\; M_1[E_2;\rho;\mathrm{rest}[\delta^+;\mathrm{count}[E_1;\rho;\delta^+])] \neq \bot \Rightarrow \\
\qquad\qquad\qquad\qquad (\mathrm{act}[\mathrm{Ability}_2]\ (\&\ E_1\ \mathrm{next}[\mathrm{Ability}_2])) \\
\qquad\qquad\qquad\qquad (\mathrm{where\ Ability}_2 = M_1[E_2;\rho;\mathrm{rest}[\delta^+;\mathrm{count}[E_1;\rho;\delta^+]]]); \\
\qquad\qquad\qquad\;\; t \Rightarrow \bot]; \\
\qquad\;\; \delta[0] = 1 \Rightarrow \\
\qquad\qquad\qquad [M_1[E_2;\rho;\delta^+] \neq \bot \Rightarrow (\mathrm{act}[M_1[E_2;\rho;\delta^+]]\ (\&\ \mathrm{next}[M_1[E_2;\rho;\delta^+]\ E_1])); \\
\qquad\qquad\qquad\;\; M_1[E_1;\rho;\mathrm{rest}[\delta^+;\mathrm{count}[E_2;\rho;\delta^+]]] \neq \bot \Rightarrow \\
\qquad\qquad\qquad\qquad (\mathrm{act}[\mathrm{Ability}_1]\ (\&\ E_2\ \mathrm{next}[\mathrm{Ability}_1])) \\
\qquad\qquad\qquad\qquad (\mathrm{where\ Ability}_1 = M_1[E_1;\rho;\mathrm{rest}[\delta^+;\mathrm{count}[E_2;\rho;\delta^+]]]); \\
\qquad\qquad\qquad\;\; t \Rightarrow \bot]]; \\
\;\; E = (\mathrm{If}\ V\ E_1\ E_2) \Rightarrow \\
\qquad [\llbracket V\rrbracket \neq \epsilon \Rightarrow M_1[E_1;\rho;\delta]; t \Rightarrow M_1[E_2;\rho;\delta]]; \\
\;\; E = (\mathrm{Apl}\ (\lambda\ X\ E')\ V) \Rightarrow M_1[E'[V/X];\omega;\delta]; \\
\;\; E = (\mu_i(P_1 \cdots P_n)\ (E_1 \cdots E_n)) \Rightarrow M_1[E_i[E_1'/P_1;\cdots;E_n'/P_n];\rho,\delta] \\
\qquad (\mathrm{where}\ E_j' = (\mu_j\ (P_1 \cdots P_n)\ (E_1 \cdots E_n))\ (j \in [n])); \\
\;\; E = (\mathrm{Apl}\ (\mu\ Q\ F)\ V) \Rightarrow M_1[(\mathrm{Apl}\ F[(\mu\ Q\ F)/Q]\ V);\rho;\delta]].
\end{array}
$$

Figure 4: Interpretation Mechanism M_1 by Means of Lazy Evaluation

5 Interpretation System CLITCS

5.1 Outline of the Interpreter

The system CLITCS (Common Lisp ITCS), read "kliks", is an interpretation system of ITCS (a variant of Milner's CCS) implemented by Common Lisp. It can also be said that CLITCS is an extension of Common Lisp in the same sense as CLOS (Common Lisp Object oriented System) ([Keen 88]) is. It enables us to define and interpret communicating processes which pass values specified by Common Lisp functions.

It has the following features.

1. It supports the interpretation mechanism M_1^T described in the previous section, which is sound and complete for the class **GWD** (the class of well guardedly defined processes).

2. It can handle every object of every data type of Common Lisp as a.value passed between processes.

3. It supports a Lisp-like interactive environment for programming, debugging, and tracing.

5.2 The Syntax of CLITCS

The syntax of CLITCS is essentially the same as that of ITCS. However, there are several modifications as follows. Some of them are due to the restriction of the character set, and others are for readability and generalization. The syntactical classes `<ProcVar>` (variables of sort Proc), `<IProcVar>` (variables of sort IProc), `<ValVar>` (variables of sort Val), `<LabVar>` (variables of sort Lab), and `<ILabVar>` (variables of sort ILab) are defined by the following BNF's.

```
<ProcVar> ::= P<Identifier>.
<IProcVar> ::= Q<Identifier>.
<ValVar> ::= X<Identifier>.
<LabVar> ::= A<Identifier>.
<ILabVar> ::= B<Identifier>.
```

Moreover, 'λ' and 'μ' are represented by 'lambda' and 'mu' respectively.

We write "(mu P_i (P_1 E_1)$\cdots$(P_n E_n))" in CLITCS for "($\mu_i(P_1 \cdots P_n)$ ($E_1 \cdots E_n$))" in ITCS.

Moreover, we may write "(mu P_1 E_1)" instead of "(mu P_1 (P_1 E_1))" for short.

We write "(lambda (X) E)" in CLITCS for "(λ X E)" in ITCS for the generalization to permit process definitions with several parameters.

Moreover, we omit the symbol 'Apl' and write "$(F\ V)$" for "(Apl $F\ V$)".

In *Common Lisp Transparent Mode* (cf. Section 5.4), every form of Common Lisp which can be evaluated is an CLITCS term of sort Val. Otherwise the class `<ValTerm>` is defined, e.g., as follows.

```
<ValTerm> ::=  <Numeral> | <ValVar>
               | (+ <ValTerm> <ValTerm>)
               | (- <ValTerm> <ValTerm>)
               | (* <ValTerm> <ValTerm>) ... .
```

5.3 Process Definition and Interpretation

For naming process specifications, 'defproc' macro is used. For example.

```
    (defproc test-proc (mu P (! A P)))
```

You can interpret a process specification E by "**(interpret** E)".
Here, E may be process specification itself or a name of a process specification given by **defproc**.

The interpretation is executed along the following steps.

1. **Expanssion:** A Process name defined by *defproc* is expanded to its process specification.
2. **Semicompiling:** The expanded specification is converted to an intermediate form.
3. **Interpretation:** The intermediate form is interpreted by the mechanism M_1^T described in the previous section.

The system prompts for input by printing "Action:". For input through a label a of sort Lab, the user should type "a ↩", where '↩' represents a carriage return.

For input through a label b of sort ILab with arity n, the user should type "$(B\ V_1 \cdots V_n)$ ↩", where $V_i \in \mathcal{T}^{\circ}_{\mathrm{Val}}$ $(i \in [n])$.

5.4 Interpretation Mode

There are five parameters for mode, i.e., '*oracle*', '*input-mode*', '*read-time*', '*val-expr-flag*', and '*trace*'.

The variable '*oracle*' specifies the 0-ary function which is provided to the interpretation function M as the third argument δ (cf. Definition 4.2). The domain of '*oracle*' is

$\{$'left-most', 'right-most', 'random'$\}$.

In the case *oracle* = 'left-most', a constant function with value 0 is specified as the oracle. In the case *oracle* = 'random', a function which returns 0 or 1 at random is specified as the oracle.

The domain of the variable '*input-mode*' is

$\{$'spontaneous', 'user-driven'$\}$.

In the case *input-mode* = 'spontaneous', the system waits for input for a limited time (i.e., the value of '*read-time*') in each step of interpretation. If no input occurs in that interval, the system judges that there is no input in that step. Otherwise, the system gets input from the terminal, and it judges that there is no input, if only carriage return is typed.

The variable '*val-expr-flag*' specifies, whether the system is in *Common Lisp Transparent Mode* or not. In Common Lisp Transparent Mode, the system considers every Common Lisp form as a term of sort Val. In the case *val-expr-flag* is non-nil, the system is in Common Lisp Transparent Mode.

The variable '*trace*' specifies how to trace the CLITCS interpretation. The domain of '*trace*' is $\{\mathrm{nil}, 0, \cdots\}$. If *trace* is null, then the interpretation is not traced, otherwise it is traced in a certain way.

5.5 Usage Examples

The following example is taken from [Bolo 87].

Example 5.1 First, the definition of a process which represents a ***stack*** is presented in Figure 5. The interpretation in a trace mode with *trace* = nil is as follows.

```
>(interpret ProcZ)
**** Action: A
**** Accepted.
**** Action: A
**** Accepted.
**** Action: A
**** Accepted.
**** Action: Ab
**** Accepted.
**** Action: Ac
**** .
**** Action: Ab
**** Accepted.
**** Action: Ac
**** .
**** Action: ^C
```

Here, an output "." indicates silent action or τ-action (See [Miln 80]). To halt the interpretation, type '`^C`'.

6 Concluding Remarks

The following problems are remaining.

First, we have shown that for $E \in \mathbf{GWD}$ (the class of guardedly well-defined processes), the interpretation mechanism M_1 conforms to the denotational semantics presented in [Hori 88] completely. However, some process specifications defined recursively by ***unguarded expressions*** can not be interpreted according to the denotational semantics presented in [Hori 88]. The following example presents one of such specifications.

Example 6.1
By the denotational semantics in [Hori 88], the following

```
(defproc ProcC
  (mu PC
      (+ (/ (? A (& ($ PC (Ag Af) (Af Ag))
                    (? Ag PC)))
              Ag)
         (? Ab (! Af Stop)) )))
(defproc ProcZ
  (mu PZ
     (+ (? A
           (/ (& ($ ProcC (Af Ag) (Ag Af))
                 (? Ag PZ) )
              Ag))
        (? Az PZ) )))
```

Figure 5: Definition of Stack

two processes

"(+ (μ p p) (! α Stop))" and "(! α Stop)"

have the same denotation, but the first is interpreted differently from the second with some oracles.

Second, we have only discussed interpretation mechanism of the sort which reduce parallel execution to nondeterministic sequential execution. It remains for further study to investigate interpretation mechanisms of CCS-like languages in the framework of *multiprogramming* or *multiprocessing* or *distributed processing* ([Andr 83]).

There are related works for extending CCS. For example, in [Yuen 88], *dynamic communication naming* has been proposed. In ITCS, a similar facility can be supported, by introducing some function symbols with arity (Val) and sort Lab or ILab. In that case, the denotational semantics $\mathcal{M}$ and the interpretation mechanism M_1 defined in the same way.

There are also related works for implementing languages based on CCS. For example, in [Tomu 88], a language $\mathcal{A}$ has been proposed. In that paper, the notion *soundness* and *completeness* of an interpretation mechanism are defined in terms of a proof system, while they are defined in terms of the denotational semantics here.

Acknowledgements

The author would like to thank Dr. Masaki Itoh and Haruhisa Ichikawa for their helpful discussions and comments.

References

[Andr 83] ANDREWS, G. R., AND SCHNEIDER, F. B. (1983), *Concepts and notations for concurrent programming*, ACM Computing Surveys, Vol. 15, No. 1, pp. 3–43.

[Bakk 80] DE BAKKER, J.W. (1980), *"Mathematical Theory of Program Correctness"* Prentice-Hall International Inc.

[Bakk 82] DE BAKKER, J.W., AND ZUCKER, J.I. (1982), Processes and the denotational semantics of concurrency, *Inform. and Control* **54**, pp. 70–120.

[Bolo 87] BOLOGNESI, T. (1987), *Fundamental results for the verification of observation equivalence, a Survey,* IFIP TC6, Zurich, May 5–8, 1987.

[Hind 86] HINDLEY, J.R., AND SELDIN, J.P. (1986), *"Introduction to Combinators and λ-Calculus"*, Cambridge University Press.

[Hori 88] HORITA, E. (1988), *A fixed point theorem on the Bakker-Zucker process domain and its application to theory of communication processes,* IEICE Technical Report, Vol. 88, No. 33, COMP88-17.

[Keen 88] KEENE, S.E. (1988) *"Object-Oriented Programming in Common Lisp,* Addison-Wesley Publishing Company.

[McCa 60] MCCARTHY, J. (1960), *Recursive functions of symbolic expressions and their computation by machine, Part 1,* Communications of the ACM, 1960, Vol.3, No.4.

[Miln 80] MILNER, R. (1980), *"A Calculus of Communicating Systems"* Lecture Notes in Computer Science, Vol. 92, Springer Berlin / New York.

[Miln 83] MILNER, R. (1983), *Calculi for synchrony and asynchrony,* Theoretical Computer Science 25, pp. 267–310.

[Tomu 88] TOMURA, S., ISHIKAWA, Y., AND FUTATSUGI, K. (1988), *The proposal of a process algebraic language: $\mathcal{A}$,* 5-th Conf. Proc. Japan Soc. Softw. Sc. Tech, C6-1 (in Japanese).

[Yuen 88] YUEN, S., SAKABE, T., AND INAGAKI, Y. (1988), *Dynamic communication naming in CCS,* Preprints Work. Gr. for Software Foundation, IPSJ, No.26-6.