マルチメディア通信と分散処理 46－6
（１９９０．７．１２）

SAL: Semantics Analyzer for LOTOS Specifications

Atsushi Sato † Kenji Kawaguchi ††
Kaoru Takahashi †† Norio Shiratori †† Shoichi Noguchi ††

† Fujitsu Customer Engineering Ltd.
3-3-5, Ichibancho, Aoba-ku, Sendai, Japan 980

†† Research Institute of Electrical Communication, Tohoku University
2-1-1, Katahira, Aoba-ku, Sendai, Japan 980

The design of a semantics analyzer SAL to analyze the semantics of a system specification described in LOTOS is given. By using SAL, a transition system with variables (a TSV for short) can be obtained from a given LOTOS specification. The use of a TSV as a semantic model of a LOTOS specification yields advantages such as a compactness of semantics representation. A transition derivation system for obtaining a TSV is presented. SAL has been implemented on a workstation. In its execution, SAL provides the user with two modes, i.e., auto mode and interactive mode. In the auto mode, SAL uses the technique of a term rewriting system to evaluate terms and predicates while in the interactive mode evaluation of terms and predicates is left to the user. Some real running examples of SAL are demonstrated.

SAL：LOTOS仕様の意味解析支援系

佐藤 淳† 川口 研治†† 高橋 薫†† 白鳥 則郎†† 野口 正一††

† 富士通カストマエンジニアリング

†† 東北大学電気通信研究所

LOTOS仕様の動的意味解析を支援する処理系SALについて述べる。LOTOS仕様が与えられると、SALでは、意味解析の結果として変数付きの遷移システムを得る。変数付き遷移システムはLOTOS仕様のコンパクトな形での動的意味表現を支援し、変数およびそれに関連した述語(ガード、選択述語など)を残した形の遷移システムに対応する。変数付き遷移システムを得るための遷移導出体系が与えられ、また、その導出体系と本来の遷移システムの導出体系との関係が議論される。SALはワークステーション上にyaccおよびCを用いて実現されている。SALの実行を効果的に行うため、自動モードおよび対話モードの2つの使用形態がユーザに提供されている。SALの具体的な適用例をいくつか示す。

1. INTRODUCTION

In order to describe a system specification precisely and unambiguously, so-called FDTs are used. Estelle [3], LOTOS [1] [2] and SDL [4] are FDTs for formal specification of, in general, open distributed systems, and, in particular, OSI protocols and services [5]. Estelle and LOTOS have been developed and standardized by ISO while SDL has been developed by CCITT.

Among others, LOTOS, dealt with in this paper, is a FDT which has a well-defined mathematical foundation inherited from such theories as CCS [6] and ACT ONE [7]. Thus LOTOS provides a solid basis for specification analysis and for tool development. However it seems rather forbidding to learn, at least at first glance.

To help in interpretation of a given LOTOS specification and to ease the problem of low understandability of LOTOS, we develop an effective semantics analysis tool for LOTOS specifications, called **SAL** (Semantics Analyzer for LOTOS specifications). The main characteristic of SAL is that it obtains a transition system with variables as a semantic interpretation from a given LOTOS specification. A transition system with variables (a TSV for short) is a usual transition system [1] augmented by the addition of variables and predicates. The use of a TSV as a semantic model of a LOTOS specification yields

(a) a compactness of semantics representation, and
(b) an easiness of mechanical handling by (a).

SAL uses a dedicated transition derivation system tailored for obtaining the objective TSV from a given LOTOS specification. This transition derivation system is proposed in this paper, and the relation between the standard one [1] and this is considered in terms of transition systems.

TSV's obtained through SAL are available for different purposes, e.g., simulation of behaviour and verification.

SAL has been implemented on a workstation using yacc and C. It provides the user with two modes of execution, auto and interactive, to make the use of SAL flexible.

This paper is organized as follows. Section 2 gives the design of SAL, in detail. In Section 3, the implementation of SAL and some application examples of SAL are given.

We assume that the readers have knowledge about LOTOS.

2. DESIGN OF SAL

In this section, we describe the design of SAL. A design outline of SAL is given first. Then its components and the basic notion in their design are described.

2.1 OUTLINE

As described in Section 1, SAL aims at an effective analysis of LOTOS specifications, and the formation of foundation for other LOTOS-related tools such as simulator and verifier.

SAL reads a LOTOS specification, and outputs the corresponding TSV (transition system with variables) as a semantics representation of the specification. The structure of SAL is shown in **Fig.1**. It consists of the following two phases (components).

(A) Syntax & Static Semantics Analysis Phase
This phase receives a LOTOS specification, and performs lexical analysis, syntax analysis and static semantics analysis. After these analysis, the corresponding flattened LOTOS specification, consisting of process definitions part and data-type definitions part, is generated. The details are given in Subsection 2.2.

(B) Semantics Analysis Phase
Through this phase, the objective TSV is obtained from the flattened LOTOS specification. Subsection 2.3 gives the details of this phase and some basic ideas in the design of this phase.

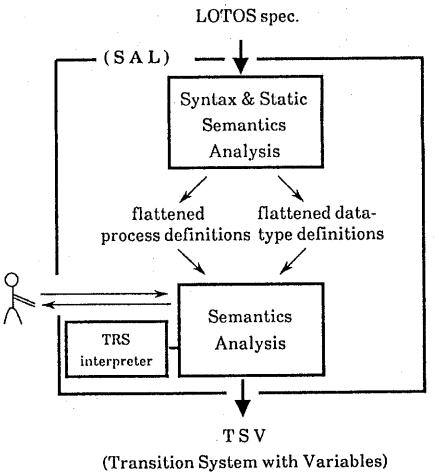

Fig.1 Structure of SAL

SAL provides the user with two modes of the usage of SAL, i.e., *auto mode* and *interactive mode*. The user can select his/her favorite mode at the beginning of execution. Terms and predicates, that occur in LOTOS behaviour-expressions, are evaluated with respect to their values by a TRS (Term Rewriting System) [8] interpreter in the auto mode or by the human user in the interactive mode (see **Fig.1**). We believe this idea makes the use of SAL flexible since a full mechanical interpretation of terms and predicates using a TRS interpreter may not necessarily succeed (for example, due to a non-termination property of TRS).

The TSV *Sys'* corresponding to a given behaviour-expression *B*, obtained through SAL (more precisely, through the transition derivation system used in SAL), is a compact form of the (usual) transition system *Sys*, obtained through the standard transition derivation system defined in [1]. The transition system generated by assigning concrete values to the variables in *Sys'*, is equivalent to *Sys* (see Subsection 2.3).

2.2 Syntax & Static Semantics Analysis

In this phase, a LOTOS specification is processed for lexical analysis, syntax analysis and static semantics analysis. The current version of SAL handles a subset of the full LOTOS syntax. With respect to process definitions, abbreviations such as *par-expression*, *local-definition-expression* and *sum-expression* are not used. And, with respect to data-type definitions, local data-type definitions and abbreviations such as *parameterization* and *renaming* are excluded. Incidentally we have newly introduced the keywords **constraint** and **endcons** into our LOTOS syntax. These keywords can be written at any places where the existing keywords **process** and **endproc** can be written. This provision helps in explicit indication of the use of the constraint-oriented style which is one of the useful specification styles [9].

In this phase, the following processing is mainly performed for a given LOTOS specification.

- identification and separation of lexical tokens
- detection of the defining and applied occurrences of identifiers
- derivation of the scopes of the defining occurrences of identifiers
- application of the rules for scopes
- check for the use of the applied occurrences of identifiers
- globalization of identifiers
- notification of syntax and static semantics errors to the user
- generation of the flattened specification

The flattened specification for a given specification consists of (a) flattened data-type definitions part and (b) flattened process definitions part (see **Fig.1**), and it is derived only for a specification which satisfies the prescribed syntax and static semantics rules (or requirements).

The flattened data-type definitions have the following characteristics. The all data-type definitions in the specification are integrated and expressed as a single data-type definition. Identifiers (operation, variable and sort identifiers) are unique every identifier class. An equation is expressed in a prefix notation. A variable identifier is expressed with its sort.

On the other hand, the flattened process definitions are represented as follows. The processes, including specification itself, in the specification are expressed separately. Identifiers (specification, process, gate, operation, variable and sort identifiers) are unique every identifier class. A behaviour-expression is represented in a prefix notation. This can be accomplished by the application of the following transformation rules. In the rules, ⟦B⟧ represents the expression of a behaviour-expression B in the prefix notation.

(a) ⟦ **hide** G **in** B ⟧ = **hide** G (⟦B⟧)
(b) ⟦ B >> B' ⟧ = >> (⟦B⟧ , ⟦B'⟧)
(c) ⟦ B [> B' ⟧ = [> (⟦B⟧ , ⟦B'⟧)
(d) ⟦ B || B' ⟧ = || (⟦B⟧ , ⟦B'⟧)
(e) ⟦ B ||| B' ⟧ = ||| (⟦B⟧ , ⟦B'⟧)
(f) ⟦ B |[G]| B' ⟧ = |[G]| (⟦B⟧ , ⟦B'⟧)
(g) ⟦ B [] B' ⟧ = [] (⟦B⟧ , ⟦B'⟧)
(h) ⟦ [P] -> B ⟧ = [P] (⟦B⟧)
(i) ⟦ a ; B ⟧ = a (⟦B⟧)
(j) ⟦ **stop** ⟧ = **stop**
(k) ⟦ **exit** ⟧ = **exit**
(l) ⟦ p[---](---) ⟧ = p[---](---)
(m) ⟦ (B) ⟧ = ⟦B⟧
(n) ⟦ (B)[R] ⟧ = [R] (⟦B⟧)

In the above rules, B and B' are *behaviour-expressions*, G is a *gate-identifier-list*, P is a *guard*, a is an *action-denotation*, p is a *process-identifier* and R is a *relabeling*.

2.3 Semantics Analysis

In this phase, the objective TSV (transition system with variables) is obtained (see **Fig.1**). Concretely speaking, given the behaviour-expression of the initial process definition (i.e., top level of behaviour-expression in the specification), this phase applies a transition derivation system to that behaviour-expression, and, as a result, generates the corresponding TSV. In the following, the transition derivation system which this phase of SAL is based on is first discussed. Next, the software design of this phase is given.

2.3.1 Derivation of TSV

We can get the corresponding transition system from a (closed) behaviour-expression by the application of the standard transition derivation system defined in [1].

The transition derivation system adopted in SAL is different from the standard one. It is a modified version of the standard one so that a TSV can be obtained. Generally, a TSV has the small number of states and transitions than the corresponding transition system because the size of the value domain of a variable can be reduced to one. As a simple example, when a behaviour-expression g?x:bool ; **stop** is given, we get a single transition in the TSV while two transitions in the transition system. Such an effect increases as the size of the value domain increases.

⟦**Definition 1**⟧ *transition system with variables*

A *transition system with variables* TSV is a 7-tuple
$<N, G, AS, V, CA, \rightarrow, init>$
where
(a) N is a set of states,
(b) G is a set of gates,
(c) AS is an algebraic specification $<S, OP, E>$ where $<S, OP>$ is a signature and E is a set of equations,
(d) V is a set of variables over S,
(e) CA is a set of *conditional actions*. Each conditional action ca $\in$ CA takes a form of $[cond_1] \cdots [cond_m]$ *act* or *act* where $cond_i$ $(1 \leqq i \leqq m)$ is an equation over $<S, OP>$ and V, and *act* is an action whose form is ① i or ② $g\ d_1 \cdots d_n\ [SP_1] \cdots [SP_h]$ or ③ i $(v_1 \cdots v_k\ [SP_1] \cdots [SP_h])$. g is an element in G. d_j $(1 \leqq j \leqq n)$ is either $!t_j$ or $?x_j{:}s_j$ where t_j is a term over $<S, OP>$ and V, and $x_j{:}s_j$ is a variable x_j in V whose sort is s_j. SP_j $(1 \leqq j \leqq h)$ is an equation over $<S, OP>$ and V. v_j $(1 \leqq j \leqq k)$ is $?x_j{:}s_j$ which is a variable x_j in V whose sort is s_j.
(f) $\rightarrow$ is a set of transitions $\rightarrow \subseteq N \times CA \times N$, and
(g) init $\in$ N is the initial state of TSV. □

In (e) in the definition above, ① represents an internal action and ② represents an observable action with (optional) selection predicates. $!t_j$ and $?x_j{:}s_j$ correspond to a value declaration and a variable declaration in LOTOS, respectively. ③ is same as ② except that it is hidden.

When (s_1, ca, s_2) belongs to $\rightarrow$ and ca is $[cond_1] \cdots [cond_m]$ act, this means that a transition from state s_1 to state s_2 is possible by the occurrence of action act provided that each equation $cond_i$ $(1 \leqq i \leqq m)$ holds. It is denoted as $s_1 - ca \rightarrow s_2$.

The essential difference between the transition derivation system SAL uses to obtain TSV's and the standard one [1] is in handling of variables which occur in a given behaviour-expression.

[Definition 2] *transition derivation system for TSV*

To obtain a TSV, we define the transition derivation system Ψ_{CLS} generated by a canonical LOTOS specification $CLS = <AS, BS>$ where AS is an algebraic specification and BS is a behaviour specification. (A canonical LOTOS specification is obtained by the application of the flattening function #.# to a given LOTOS specification [1].) Ψ_{CLS} is defined by the following axioms and inference rules (for lack of space, only those peculiar to handling of variables are given). Note that a behaviour-expression is represented in the prefix notation. The notation and functions used throughout this paper, including this definition, are listed in Table 1.

① action prefix (observable action without selection predicates)

- $g\ d_1 \cdots d_n\ (B) \quad - g\ v_1 \cdots v_n \rightarrow \quad [y_1'/y_1, \cdots, y_m'/y_m]\ B$ is an axiom

iff

notation / func.	meaning
$[t_1/z_1, \text{-----}, t_n/z_n]\ B$	simultaneous replacement of all occurrences of variables $z_1, \text{---}, z_n$ in a behaviour-expression B by terms $t_1, \text{---}, t_n$, avoiding naming clash.
name(ca)	gate identifier in conditional action *ca*
sort(t)	sort of term t
Ω_{AS}	derivation system generated by AS [1]
$[t]_{AS}$	AS-congruence class of a ground term t over signature $<S, OP>$, i.e., $\{ t' \mid \Omega_{AS} \vdash t = t' \}$
$Q_{s,AS}$	value domain of sort $s \in S$, i.e., $\{ [t]_{AS} \mid t$ is a ground term of sort $s \in S \}$
$A_{V,AS}$	set of all functions from $\{ (x_1, \cdots, x_n) \}$ to $Q_{sort(x_1),AS} \times \cdots \times Q_{sort(x_n),AS}$, i.e., all *value assignments* to the variables in V
$\sigma(r)$	term or equation obtained by replacing the variables in a term or equation r by the ground terms which represent the values $\sigma \in A_{V,AS}$ assigns
$x \leftarrow v$	assignment of a value $v \in Q_{sort(x),AS}$ to a variable $x \in V$
$\sigma_{[y_1 \leftarrow v_1, \cdots, y_m \leftarrow v_m]}$	value assignment which differs from $\sigma \in A_{V,AS}$, in only $y_1 \leftarrow v_1, \text{-----}, y_m \leftarrow v_m$

Note: In this table, the existence of an algebraic specification $AS = <S, OP, E>$ and a set of variables $V = \{x_1, \text{---}, x_n\}$ is assumed.

Table 1 Notation and functions

$v_i = ?y_j{:}s_i$ if $d_i = ?y_j{:}s_i$ $(1 \leqq i \leqq n, 1 \leqq j \leqq m)$,
$v_i = !\,t_i'$ if $d_i = !t_i$ $(1 \leqq i \leqq n)$ and t_i is a ground term,
$v_i = !\,t_i$ if $d_i = !t_i$ $(1 \leqq i \leqq n)$ and t_i is a non-ground term
where y_j' $(1 \leqq j \leqq m)$ is a *fresh* variable identifier, $\{y_1, \cdots, y_m\} = \{ x_i \mid d_i = ?x_i{:}s_i, 1 \leqq i \leqq n \}$ and $t_i' \in [t_i]_{AS}$.

② action prefix (observable action with selection predicates)

- $g\ d_1 \cdots d_n\ [SP_1] \cdots [SP_h]\ (B) \quad - g\ v_1 \cdots v_n\ [SSP_1] \cdots [SSP_k] \rightarrow \quad [y_1'/y_1, \cdots, y_m'/y_m]\ B$ is an axiom

iff

the condition in ① above plus the following condition hold.
$[SSP_1], \cdots, [SSP_k]$ are non-ground equations in $[y_1'/y_1, \cdots, y_m'/y_m]([SP_1] \cdots [SP_h]), \Omega_{AS} \vdash SP''_1, \cdots, \Omega_{AS} \vdash SP''_f$ where $SP''_1, \cdots, SP''_f$ are ground equations in $[y_1'/y_1, \cdots, y_m'/y_m]([SP_1] \cdots [SP_h])$ and $h = f + k$.

③ hiding

- $\dfrac{B - ca \rightarrow B' \quad , \quad name(ca) \notin \{g_1, \text{---}, g_n\}}{\text{hide } g_1, \text{---}, g_n\ (B) \quad - ca \rightarrow \quad \text{hide } g_1, \text{---}, g_n\ (B')}$ and

- $\dfrac{B - [cond_1] \cdots [cond_k]\ g\ v_1 \cdots v_m\ [SP_1] \cdots [SP_h] \rightarrow B' \ , \quad g \in \{g_1, \text{---}, g_n\}}{\text{hide } g_1, \text{---}, g_n\ (B) \quad - [cond_1] \cdots [cond_k]\ i\ (t_1 \cdots t_f\ [SP_1] \cdots [SP_h]) \rightarrow \quad \text{hide } g_1, \text{---}, g_n\ (B')}$

are inference rules

iff

$\{t_1, \cdots, t_f\} = \{ v_i \mid v_i = ?x_i{:}s_i, 1 \leq i \leq m \}$.

④ guarding

- $\dfrac{B - ca \rightarrow B'}{[P](B) - [P]\, ca \rightarrow B'}$ is an inference rule

iff P is a non-ground equation.

- $\dfrac{B - ca \rightarrow B'}{[P](B) - ca \rightarrow B'}$ is an inference rule

iff P is a ground equation and $\Omega_{AS} \vdash P$.

⑤ parallel composition

- $\dfrac{B1 - ca \rightarrow B1' \quad , \quad name(ca) \notin G \cup \{\delta\}}{|[G]|(B1, B2) - ca \rightarrow |[G]|(B1', B2)}$

- $\dfrac{B2 - ca \rightarrow B2' \quad , \quad name(ca) \notin G \cup \{\delta\}}{|[G]|(B1, B2) - ca \rightarrow |[G]|(B1, B2')}$

- $\dfrac{B1 - [cond_{11}] \cdots [cond_{1r}]\, \delta \rightarrow B1' \ , \quad B2 - [cond_{21}] \cdots [cond_{2s}]\, \delta \rightarrow B2'}{|[G]|(B1, B2) - [cond_{11}] \cdots [cond_{1r}]\, [cond_{21}] \cdots [cond_{2s}]\, \delta \rightarrow |[G]|(B1', B2')}$

- $\dfrac{B1 - [cond_{11}] \cdots [cond_{1r}]\, g\, v_{11} \cdots v_{1n}\, [SP_{11}] \cdots [SP_{1k}] \rightarrow B1' \ , \quad B2 - [cond_{21}] \cdots [cond_{2s}]\, g\, v_{21} \cdots v_{2n}\, [SP_{21}] \cdots [SP_{2h}] \rightarrow B2' , g \in G}{|[G]|(B1, B2) - [cond_{11}] \cdots [cond_{1r}]\, [cond_{21}] \cdots [cond_{2s}]\, g\, v_1 \cdots v_n\, [SP_1] \cdots [SP_m] \rightarrow |[G]|([\sigma 1]B1', [\sigma 2]B2')}$

are inference rules

iff

$v_i = ?x_i{:}s_i$ (x_i is a fresh variable identifier),

$\sigma_{1i} = x_i/x_{1i}$, $\sigma_{2i} = x_i/x_{2i}$
if $v_{1i} = ?x_{1i}{:}s_i$, $v_{2i} = ?x_{2i}{:}s_i$ $(1 \leq i \leq n)$

$v_i = !t_{2i}$, $\sigma_{1i} = t_{2i}/x_{1i}$
if $v_{1i} = ?x_{1i}{:}s_i$, $v_{2i} = !t_{2i}$, $sort(t_{2i}) = s_i$ $(1 \leq i \leq n)$

$v_i = !t_{1i}$, $\sigma_{2i} = t_{1i}/x_{2i}$
if $v_{1i} = !t_{1i}$, $v_{2i} = ?x_{2i}{:}s_i$, $sort(t_{1i}) = s_i$ $(1 \leq i \leq n)$

$v_i = !t_{1i}$ if $v_{1i} = !t_{1i}$, $v_{2i} = !t_{2i}$, $sort(t_{1i}) = sort(t_{2i})$, $=(t_{1i}, t_{2i})$ is a ground equation, $\Omega_{AS} \vdash =(t_{1i}, t_{2i})$ $(1 \leq i \leq n)$

$v_i = !t_{1i}$ if $v_{1i} = !t_{1i}$, $v_{2i} = !t_{2i}$, $sort(t_{1i}) = sort(t_{2i})$, $=(t_{1i}, t_{2i})$ is a non-ground equation $(1 \leq i \leq n)$

and

$[\sigma 1] = [\sigma_{11}, \cdots, \sigma_{1n}]$ and $[\sigma 2] = [\sigma_{21}, \cdots, \sigma_{2n}]$

and

$\{[SP_1], \cdots, [SP_m]\} = \{[SP] \mid [SP] = [\sigma 1]([SP_{1j}]), 1 \leq j \leq k\} \cup \{[SP] \mid [SP] = [\sigma 2]([SP_{2j}]), 1 \leq j \leq h\} \cup \{[SP] \mid [SP] = [=(t_{1i}, t_{2i})], v_{1i} = !t_{1i}, v_{2i} = !t_{2i}, sort(t_{1i}) = sort(t_{2i}), =(t_{1i}, t_{2i})$ is a non-ground equation, $1 \leq i \leq n\}$ □

The TSV, denoted as $TSV_{CLS}(B)$, corresponding to a behaviour-expression B relative to a canonical LOTOS specification $CLS = <AS, BS>$ is obtained using the transition derivation system Ψ_{CLS} generated by CLS.

「Definition 3」 *TSV corresponding to a behaviour-expression*

The TSV, denoted as $TSV_{CLS}(B)$, corresponding to a behaviour-expression B relative to a canonical LOTOS specification $CLS = <AS, BS>$ is a 7-tuple

$<DER_{CLS}(B), G_{CLS}(B), AS, V_{CLS}(B), CA_{CLS}(B), \rightarrow_{CLS}, B>$

where

$DER_{CLS}(B)$ is the smallest set satisfying:

(a) $B \in DER_{CLS}(B)$ and

(b) if $B' \in DER_{CLS}(B)$ and $\Psi_{CLS} \vdash B' - ca \rightarrow B''$ for some ca then $B'' \in DER_{CLS}(B)$.

$CA_{CLS}(B)$, $V_{CLS}(B)$, and $G_{CLS}(B)$ are a set of the conditional actions, a set of the variables and a set of the gates which occur in the process of derivation of $DER_{CLS}(B)$, respectively.

$\rightarrow_{CLS} = \{(B_1, ca, B_2) \mid B_1, B_2 \in DER_{CLS}(B), ca \in CA_{CLS}(B), \Psi_{CLS} \vdash B_1 - ca \rightarrow B_2\}$ □

The semantics of $TSV_{CLS}(B)$ is defined as a transition system obtained by assigning concrete values to the variables in $TSV_{CLS}(B)$ and by evaluating the related predicates (guards and selection predicates) under AS.

「Definition 4」 *reconstruction of transitions of TSV*

The *reconstructed transitions* (denoted as *RT(TSV)*) of a TSV $TSV = <N, G, <S, OP, E>, V, CA, \rightarrow, init>$ are a set of transitions

$RT(TSV) \subseteq P \times Act \times P$

defined by the following rules ①, ② and ③ where $P = N \times A_{V,<S,OP,E>}$ and $Act = \{i\} \cup \{g<v> \mid g \in G, v \in (\cup \{Q_{s,AS} \mid s \in S\})^*\}$. (See **Table 1** for the notation)

① there exists a transition such that $q - [cond_1] \cdots [cond_m]\, i \rightarrow q' \in \rightarrow$ and $\Omega_{<S,OP,E>} \vdash \sigma(cond_j)$ for all j $(1 \leq j \leq m)$.
$\Rightarrow ((q, \sigma), i, (q', \sigma)) \in RT(TSV)$

② there exists a transition such that $q - [cond_1] \cdots [cond_m]\, g\, d_1 \cdots d_n\, [SP_1] \cdots [SP_h] \rightarrow q' \in \rightarrow$ and $\Omega_{<S,OP,E>} \vdash \sigma(cond_j)$ for all j $(1 \leq j \leq m)$. And, $\Omega_{<S,OP,E>} \vdash \sigma[y_1 \leftarrow u_1, \cdots, y_k \leftarrow u_k](SP_w)$ for all w $(1 \leq w \leq h)$ where $\{y_1, \cdots, y_k\} = \{x_i \mid d_i = ?x_i{:}s_i, 1 \leq i \leq n\}$ and $u_z \in Q_{sort(y_z),<S,OP,E>}$ $(1 \leq z \leq k)$.
$\Rightarrow ((q, \sigma), g<v_1 \cdots v_n>, (q', \sigma[y_1 \leftarrow u_1, \cdots, y_k \leftarrow u_k])) \in RT(TSV)$

where

$v_i = [\sigma(t_i)]_{<S,OP,E>}$ if $d_i = !t_i$ $(1 \leq i \leq n)$

$v_i = u_z$ if $d_i = ?x_i{:}s_i$, $x_i = y_z$ $(1 \leq i \leq n, 1 \leq z \leq k)$

③ there exists a transition such that $q - [cond_1] \cdots [cond_m]\, i\, (?y_1{:}s_1 \cdots ?y_k{:}s_k\, [SP_1] \cdots [SP_h]) \rightarrow q' \in \rightarrow$ and $\Omega_{<S,OP,E>} \vdash \sigma(cond_j)$ for all j $(1 \leq j \leq m)$. And, $\Omega_{<S,OP,E>} \vdash \sigma[y_1 \leftarrow u_1, \cdots, y_k \leftarrow u_k](SP_w)$ for all w $(1 \leq w \leq h)$ where $u_z \in Q_{s_z,<S,OP,E>}$ $(1 \leq z \leq k)$.
$\Rightarrow ((q, \sigma), i, (q', \sigma[y_1 \leftarrow u_1, \cdots, y_k \leftarrow u_k])) \in RT(TSV)$ □

「Definition 5」 *semantics of a TSV*

The *semantics* of a TSV TSV=<N, G, <S, OP, E>, V, CA, →, init> *by an initial value assignment* $\iota \in A_{V,<S,OP,E>}$ is a transition system (denoted as $TS_\iota(TSV)$)

<States(RT_ι(TSV)), Actions(RT_ι(TSV)), RT_ι(TSV), (init, ι)>

where

$RT_\iota(TSV) = RT(TSV) - \{ t \mid t \in RT(TSV), t \text{ is unreachable from state } (init, \iota) \}$,

$States(RT_\iota(TSV)) = \{ q \mid q \in P, q \text{ is reachable from } (init, \iota) \text{ in } RT_\iota(TSV) \}$,

$Actions(RT_\iota(TSV)) = \{ a \mid a \in Act, a \text{ is contained in } RT_\iota(TSV) \}$. □

In the following, we consider a relation between the standard one [1] and the transition derivation system given in **Definition 2**. For convenience, the former is called *System A*, and the latter is called *System B*.

「Definition 6」 *bisimulation [10]*

Let $TS_1 = <S_1, Act, \rightarrow_1, s_{01}>$ and $TS_2 = <S_2, Act, \rightarrow_2, s_{02}>$ be transition systems and let $S = S_1 \cup S_2$. A relation $R \subseteq S \times S$ is a *bisimulation relation* if for every pair $(s_1, s_2) \in R$ and every action $a \in Act$:

① whenever $s_1 - a \rightarrow s_1'$:
$s_2 - a \rightarrow s_2'$ and $(s_1', s_2') \in R$ for some $s_2' \in S$

② whenever $s_2 - a \rightarrow s_2'$:
$s_1 - a \rightarrow s_1'$ and $(s_1', s_2') \in R$ for some $s_1' \in S$.

TS_1 and TS_2 are *bisimulation equivalent* if there exists a bisimulation relation R such that $(s_{01}, s_{02}) \in R$. □

If TS_1 and TS_2 are bisimulation equivalent then TS_1 and TS_2 are weak bisimulation equivalent [2].

The main difference between System A and System B is in handling of variables (in the form ?x:s). In System A, for a given closed behaviour-expression, the corresponding transitions (in the form of $-$action$\rightarrow$) are obtained with evaluation of variables by concrete values. On the other hand, in System B, the corresponding conditional transitions on the values of the variables (in the form of $-[cond_1] \cdots [cond_m]$ action $\rightarrow$) are obtained, remaining the variables (and consequently the related guards and selection predicates) unevaluated.

The difference between the Systems A and B in *action-prefix* appears if variables occur in an *action-denotation*. In System A, the variables are evaluated by the corresponding concrete values and then transitions are obtained. If needed, the evaluation of the selection predicates is involved. On the other hand, in System B, the variables (and the selection predicates containing the variables) remain unevaluated, and then transitions are obtained. It is clear that the transitions (reconstructed transitions) constructed, by evaluating the variables in the transitions obtained through System B by the corresponding concrete values and by simultaneously evaluating the related selection predicates, correspond to the transitions obtained through System A. And, the destination states of the transitions obtained through System B correspond to the destination states obtained through System A, by making the values in the evaluation participate in the identification of the destination states, as in **Definition 4**.

The difference between the Systems A and B in *guarding* is similar to that of action-prefix. By evaluating the variables by the concrete values and by evaluating guards, the reconstructed transitions and destination states of the transitions corresponding to those in System A are obtained.

In *hiding*, the hidden actions are recorded in the transitions, in case of variable declarations, in System B. As in **Definition 4**, by reconstructing the transitions and the destination states of the transitions, considering the recorded information, the reconstructed transitions and destination states correspond to those obtained through System A.

The essential difference between Systems A and B in *parallel-composition* is in derivation of a transition by a synchronization which involves two terms (value declarations) where at least one contains a variable. In System A, in such a case, a synchronization holds when the evaluation of the both terms is equal. Therefore in System B, a selection predicate which asserts that the evaluation of the both terms should be equal is added as a condition for transition. As in **Definition 4**, by reconstructing the transitions and destination states of the transitions with evaluation of the variables by the concrete values and evaluation of the selection predicate, the reconstructed transitions and destination states correspond to those obtained through System A.

From the above mentioned argument, for a closed behaviour-expression, there exists a reconstruction (see **Definitions 4** and **5**) of the transitions and destination states of the transitions obtained through System B, corresponding to those obtained through System A. Thus, the following theorem holds.

「Theorem 1」 *relation between Systems A and B*

Let B be a closed behaviour-expression relative to a canonical LOTOS specification CLS = <AS, BS>. Then

$TS_{CLS}(B)$ and $TS_\iota(TSV_{CLS}(B))$ are bisimulation equivalent

where

$TS_{CLS}(B)$ is the transition system corresponding to B, $TSV_{CLS}(B)$ is the TSV corresponding to B, ι is an arbitrary initial value assignment to the variables in $TSV_{CLS}(B)$, $TS_\iota(TSV_{CLS}(B))$ is the semantics of $TSV_{CLS}(B)$ by ι. □

By a similar argument to the above mentioned argument, the following theorem holds, for the behaviour-expression in the initial process definition of a canonical LOTOS specification.

「Theorem 2」

Let B be the behaviour-expression in the initial process definition of a canonical LOTOS specification CLS = <AS, BS>, let $x_1{:}s_1$, ---, $x_n{:}s_n$ be the formal value parameters of the initial process definition, and let t_i be a ground term of sort s_i for each i ($1 \leqq i \leqq n$). Then

$TS_{CLS}([t_1/x_1, \text{-----}, t_n/x_n]\,B)$ and $TS_\sigma(TSV_{CLS}(B))$ are bisimulation equivalent

where

$\sigma = \epsilon\,[x_1 \leftarrow [t_1]_{AS}, \text{---}, x_n \leftarrow [t_n]_{AS}]$,

ϵ is an arbitrary initial value assignment to the variables in $TSV_{CLS}(B)$. □

2.3.2 Software Design

This semantics analysis phase generates the objective TSV, through the application of the transition derivation system given above, from a flattened LOTOS specification obtained by the first phase.

In evaluation of terms and predicates occurring in behaviour-expressions, this phase provides the user with two modes of usage: *auto mode* and *interactive mode*. In the auto mode, the equations given in the data-type definition are regarded as term rewriting system (TRS) [8], from left hand side to right hand side. Terms and predicates, that occur in behaviour-expressions, are evaluated with respect to their values by a TRS interpreter. The strategy which this TRS interpreter adopts in the current version is leftmost outermost reduction, expecting TRS's such that linear, non-overlapping and left system. On the other hand, in the interactive mode, such evaluations are all left to the user. The user is always requested to input the reduced form (normal form) of the term to be evaluated or the truth value of the predicate to be evaluated. We believe that this idea, i.e., two modes of execution, makes the use of SAL flexible since a full mechanical evaluation of terms and predicates using a TRS interpreter may not necessarily succeed, e.g., due to a non-termination property and reduction strategy. The user is requested to carefully use SAL.

To effectively perform execution of this phase, a composition on *relabelings* and a cancel of an identity relabeling are done, i.e.,

(a) B [S] [R] → B [R • S]
(b) B [I] → B

where S and R are relabelings and I is an identity relabeling.

If the given process has infinite control structure, the execution of this phase never stops. An *unguarded* process [6] as shown below is an example of such processes:

```
process p[a]:noexit := p[a] [] a;stop endproc
```

Therefore, similarly to the use of the auto mode with respect to evaluation of terms, the user has to pay attention to the property of the specification.

3. Implementation and Application

SAL has been implemented on a Sun-3 workstation. In the implementation of the syntax & static semantics analysis phase, we have used yacc for a rapid development. About 600 lines of rules given to yacc for lexical and syntax analysis and about 1,300 lines of C program codes for static semantics analysis were created. On the other hand, in the implementation of the semantics analysis phase, about 3,300 lines of C program codes are created including about 600 lines for the TRS interpreter.

An input specification to SAL is given as a file which is created using a general purpose text editor such as emacs. If there are some errors such as lexical, syntax and static semantics errors, then the kinds and places of the detected errors are notified. If the specification contains no errors, then the corresponding flattened specification, consisting of the process definitions part and data-type definitions part, is generated in files. (If the input is a basic LOTOS specification, then the data-type definitions part is empty.) Next, a semantics analysis based on the transition derivation system described in the preceding section is carried out for this flattened specification. As described in the preceding section, the user has a choice between two modes of execution. In case of the auto mode, the evaluation of terms and predicates is left to the TRS interpreter. However, to avoid infinite loop of the evaluation due to a non-termination property of the TRS, SAL imposes some upper limit with respect to the number of times of term rewritings. When exceeding this limit, the TRS interpreter stops evaluating, and it asks the user to input the reduced form (normal form) of the term being evaluated or the truth value of the predicates being evaluated. The terms and predicates previously already evaluated are never evaluated again, in the both modes.

The TSV obtained through SAL is stored in a file. If necessary, it can be displayed on the screen, by using a tool for display.

Next, we show a real application example of SAL.

The example is given in Fig.2. The left window shows a part of the given specification. The use of the interactive mode in the application of SAL to this example yields interactions as shown in the top-right window. An input from the user is performed following the prompt message ">". In this figure, "m1" is input as the evaluation of the term "first(add(new_channel,m1))", and "true" is input as the evaluation of the predicate "=(eq(m1, m1), true)". After such interactions between the user and SAL, the TSV partly shown in the bottom-right window is obtained. In the auto mode, there are no such interactions between the user and SAL, and the same result is obtained.

4. Conclusion

In this paper, we have described SAL, a semantics analyzer for LOTOS specifications. Through SAL, a

transition system with variables (a TSV) is obtained from a given specification written in a large subset of LOTOS. A transition derivation system, which SAL is based on, has been formally presented to obain a TSV. The TSV provides the semantics of the specification in a compact form. TSV's obtained through SAL are available for different purposes such as a simulation, verification, implementation, etc. Consequently, as an effect of SAL, it is expected that the productivity of system developments based on LOTOS is enhanced.

The following further studies are expected:

a) an extensive application of SAL
b) a development of a new version of SAL that covers the full syntax of LOTOS
c) an extension of the proposed transition derivation system, for b)
d) a development of tools (e.g., simulator) based on TSV's obtained through SAL

REFERENCES

[1] ISO, "Information processing systems - open systems interconnection - LOTOS - a formal description technique based on the temporal ordering of observational behaviour," ISO 8807, 1989.

[2] T.Bolognesi and E.Brinksma, "Introduction to the ISO specification language LOTOS," Computer Networks and ISDN Systems, vol.14, pp.25-59, 1987.

[3] ISO, "Information processing systems - open systems interconnection - Estelle - a formal description technique based on an extended state transition model," ISO 9074, 1989.

[4] CCITT, "Specification and Description language SDL," CCITT Recommendation Z.100, 1988.

[5] ISO, "Information processing systems - open systems interconnection - basic reference model," ISO 7498, 1984.

[6] R.Milner, "A calculus of communicating systems," Lecture Notes in Computer Science, vol.92, 1980.

[7] H.Ehrig and B.Mabr, "Fundamentals of algebraic specification I," Springer Publishing Company, Berlin, 1985.

[8] G.Huet and D.C.Oppen, "Equations and rewrite rules: a survey," in "Formal language: perspectives and open problems," R.Book eds., Academic Press, pp.349-405, 1980.

[9] C.A.Vissers, G.Scollo and M.v.Sinderen, "Architecture and specification style in formal descriptions of distributed systems," Protocol Specification, Testing, and Verification VIII, pp.189-204, 1988.

[10] D.Park, "Concurrency and automata on infinite sequences," Proc. 5th GI Conf., Lecture Notes in Computer Science, vol.104, Springer-Verlag, 1981.

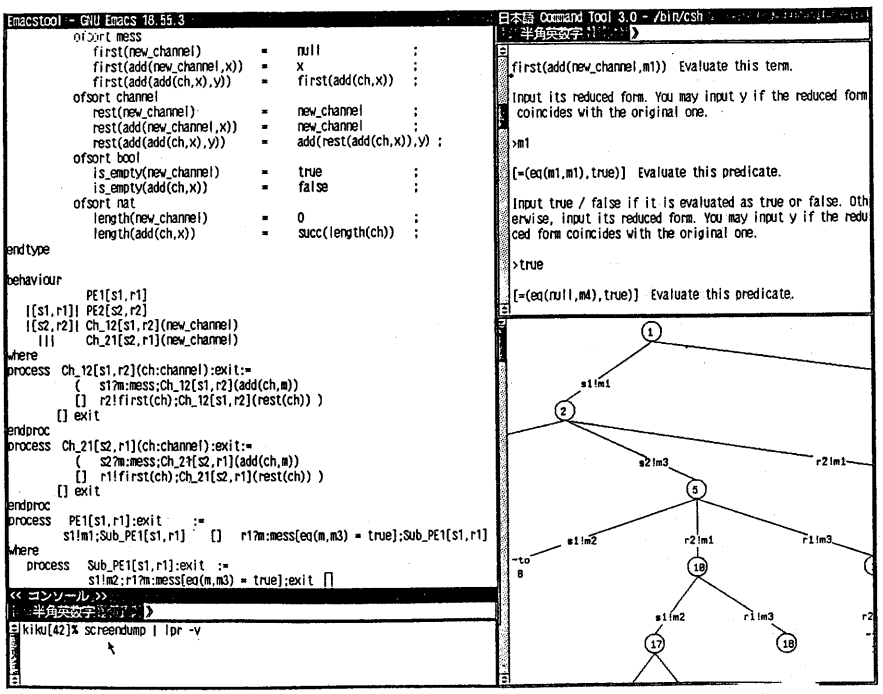

Fig.2 An example