プログラミング-言語・基礎・実践- 16－8
(1994. 3. 9)

プロセス合成のための支援環境に関する研究

臼井 伸幸　吉田 仙　木村 成伴　富樫 敦　白鳥 則郎

東北大学電気通信研究所

〒980 仙台市青葉区片平 2-1-1

E-mail:usui@shiratori.riec.tohoku.ac.jp

あらまし

プロセスの代数的計算システムとしてMilnerのCCSなどが代表的であり，通信プロセスや並行プログラムなどを形式的に記述するために利用される他，検証問題などにも応用できる．その反面，プロセス計算は読解性や記述のし易さに難点がある．そこで筆者らは，μ-calculus による論理式をプロセスの性質とみなし，与えられた性質からそれらを全て満たすプロセスを合成するアルゴリズムを提案した．本稿では，実用的なプロセス合成へのアプローチとして，このアルゴリズムを用いて実際にプロセスを合成するための支援環境を構築し，試作システムによって実際にプロセスを合成した例を示す．

和文キーワード　プロセス計算，μ-calculus，プロセス合成，支援環境

A Synthesizer for Algebraic Processes

Nobuyuki USUI, Sen YOSHIDA, Shigetomo KIMURA,

Atsushi TOGASHI and Norio SHIRATORI

Research Institute of Electrical Communication

Tohoku University

2-1-1, Katahira, Aoba-ku, Sendai, 980, Japan

E-mail:usui@shiratori.riec.tohoku.ac.jp

Abstract

Process calculi, e.g, Milner's CCS, are used to describe and verify the behavior of communicating concurrent processes formally. However, it needs skill and experience to write and understand specifications in the calculi. difficult. We have already proposed the synthesis algorithm for algebraic processes from their properties given as formulae in the μ-calculus . In this paper,we state the outline of a process synthesizer and its support environment based on the proposed algorithm. The prototype system is constructed in the practical point of view and we discribe running examples which are made with the help of the system.

英文 key words　process calculus, μ-calculus , process synthesis, support environment

1 はじめに

近年，プロセスを形式的かつ代数的に扱うことが可能なプロセス計算に関する研究 [1] が盛んに行われている．しかし，実際にプロセス計算で記述されたプロセスは複雑なものになりがちで，動作の理解や記述が困難であった．そこで筆者らは，μ-calculus [1] による論理式をプロセスの性質とみなし，与えられた性質からそれらを全て満たすプロセスを合成するアルゴリズム [3, 4] を提案した．このアルゴリズムは，論理式に用いる演算子を一部に限定しているが，再帰を含むプロセスを有限ステップで合成することができるものである．

本稿では，実用的なプロセス合成へのアプローチとして，このアルゴリズムを用いて実際にプロセスを合成するための支援環境を構築した．この支援環境では，具体的な論理式からプロセス (必要ならば再帰を含む) を合成し，ProCSuS のグラフィックインタフェース [7] を用いてその遷移関係を解析することができる．

2 準備

2.1 プロセスの代数的形式化

以下で述べるアクションの有限集合 $\mathcal{A}$ を仮定する．

定義 2.1 本システムで取り扱うプロセスを以下のように帰納的に定義する．

(1) プロセス変数 x はプロセスである．
(2) 無動作プロセス $\mathbf{0}$ はプロセスである．
(3) p がプロセスであるとき，アクション $a \in \mathcal{A}$ によるアクションプレフィクス $a.p$ はプロセスである．
(4) p_1, p_2 をプロセスとすると，これらによる和 $p_1 + p_2$ はプロセスである．
(5) c がプロセス定数であるとき，c はプロセスである．

□

プロセス定数の意味は，プロセス定義式によって与えられる．プロセス定義式を c, プロセスを p とするとき，プロセス定義式は $c \stackrel{\text{def}}{=} p$ と表現する．

これらのプロセスの意味は「ラベル付き遷移システム」[6] によって定められる．

定義 2.2 ラベル付き遷移システムは，3項組 $<S, \mathcal{A}, \rightarrow>$ である．ここで S は状態の集合，$\mathcal{A}$ はアクションの集合 ，$\rightarrow$ は遷移関係であり，$\rightarrow \subset S \times \mathcal{A} \times S$ として定義される． □

このプロセス計算で記述したプロセスの例として，$\{c_0 \stackrel{\text{def}}{=} a.c_1, c_1 \stackrel{\text{def}}{=} a.\mathbf{0} + b.c_1\}$ の遷移関係を以下に示す．

$$c_0 \stackrel{\text{def}}{=} a.c_1 \xrightarrow{a} c_1 \stackrel{\text{def}}{=} a.\mathbf{0} + b.c_1 \xrightarrow{a} \mathbf{0}$$

図 1: $\{c_0 \stackrel{\text{def}}{=} a.c_1, c_1 \stackrel{\text{def}}{=} a.\mathbf{0} + b.c_1\}$ の遷移関係

2.2 μ-calculus

本支援環境での入力に用いるプロセスの性質を表わすのに，μ-calculus [1] を用いる．

ただし，$\mathcal{X}$ は変数の無限集合とする．

定義 2.3 論理式は次のように帰納的に定義される．

(1) T は論理式である．
(2) 変数 $x \in \mathcal{X}$ は論理式である．
(3) f, f' が論理式ならば，$f \vee f', \neg f$ は論理式である．
(4) f が論理式ならば，$\langle a \rangle f$ は論理式である．ここで $a \in \mathcal{A}$ とする．
(5) x が変数で，f 中の自由な出現が正である (それぞれの x の自由な出現が偶数個の $\neg$ のスコープの中に含まれている) 論理式ならば，$\mu x.f$ は論理式である．

□

以下では，μ-calculus による論理式をプロセスの性質と考える．プロセス p が論理式 f を満たすことを $p \models f$ と書く．

定義 2.4 論理式の充足性を次のように定義する．

(1) 任意のプロセス p に対して，$p \models T$ である．

(2) $p \models A_1 \vee A_2$ とは，$p \models A_1$ であるか，$p \models A_2$ である示す．

(3) $p \models \neg f$ とは，$p \not\models f$ である．ここで，$p \not\models f$ はプロセス p が f を満たさないことを示す．

(4) $p \models \langle a \rangle f$ とは，ある q が存在して，$p \xrightarrow{a} q$ かつ $q \models f$ である．

(5) $p \models \mu x.f(x)$ とは，$p \models f(g) \supset g (\equiv \neg f(g) \vee g)$ を満たす任意の論理式 g において，$p \models g$ が成り立つことである．

□

定義 2.5 便宜的に以下の論理記号を導入する．

(1) $F \stackrel{\text{def}}{\equiv} \neg T$

(2) $A_1 \wedge A_2 \stackrel{\text{def}}{\equiv} \neg(\neg A_1 \vee \neg A_2)$.

(3) $[a]A \stackrel{\text{def}}{\equiv} \neg\langle a \rangle \neg A$.

(4) $\nu x.f(x) \stackrel{\text{def}}{\equiv} \neg\mu x.\neg f(\neg x)$.

□

定義 2.6 論理式 f 中の変数 x がガード的であるとは，x の任意の出現が $\langle a \rangle$ のスコープに入っていることである． □

補題 2.7 定義 *2.3* で定めた任意の論理式は，論理演算子 $T, F, \wedge, \vee, \langle a \rangle, [a], \mu, \nu$ だけを用いた論理式に等価変換できる． □

以下では，問題を簡単化するために，論理演算子を $T, F, \wedge, \langle a \rangle, [a], \nu$ に限定して考える．

これらの論理式を用いて記述したプロセスの性質の例を紹介する．

(1) $p \models \langle a \rangle T$:$p$ において a が動作可能

(2) $p \models [a]F$:p は a を実行することは不可能である．

(3) $p \models \nu x.\langle a \rangle x$:$p$ は a が無限回動作可能

2.3 事実の枚挙

プロセスの具体的な性質として考えた論理式から意図したプロセスと強等価なプロセスを合成する方法として，事実の枚挙 [4] がある．この方法は，意図するプロセスが満たすべき事実を入力すると，入力された事実の情報を使い，プロセスの骨格を徐々に構築していくというものである．アルゴリズムによって推測された現在のプロセスを p_c，この時点で入力された事実を A とすると，アルゴリズムは p_c が A を満たすように p_c を組織的に変形する．もちろん，変形され，新たに得られたプロセスは，A 以前に入力された事実も満たさなくてはならない．そして，更に新たな事実を読み込み，以上の操作を繰り返していく．

3 プロセス合成支援システム

事実の枚挙を用いると，新たな事実が入力されたときにプロセスの作り直しを行う場合がある．このような処理を行うのに適したプログラミング言語として，Prolog[2] が挙げられる．Prolog は記号処理，すなわち非数値演算用の論理型プログラミング言語であり，バックトラックを行うことができる．この言語は対象と対象間の関係 (ここでは論理式とプロセス) に関した問題解決に特に適している．また，Prolog はリストを用いたデータベースの操作を行うことができる．

これらの利点から本システムを Prolog 上に実装した．

3.1 システム構成

システム構成の概略を図 2 に示す．UI はユーザーインタフェース部分であり，事実が入力されると，リスト構造に変換しする．makeproc は実際に論理式からプロセスを合成する部分である．現在までに合成されているプロセスと新たに入力された論理式から，矛盾の有無を調べ，矛盾がなければプロセスを合成する．新しく合成されたプロセスを UI ,ProCSuS ,DataBase に出力する．DataBase では，合成されたプロセスを保存し，新たな事実の入力に備える．

3.2 プロセスの内部表現

本システムで合成するプロセスの意味はラベル付き遷移システムによって与えられる．さらに，入力された論理式の一部はそれぞれのプロセス定数に保存しておくことが必要になる．そ

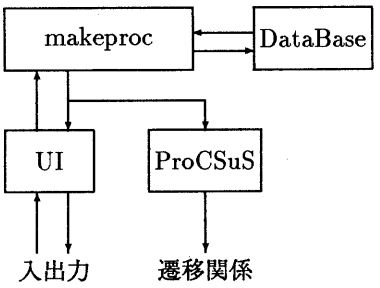

図 2: システム構成図

こで本システムでは，プロセスをプロセス定義式の集合として扱う．

定義 3.1 プロセス定義式は *3* 項組 $\langle c_i, C_i, T_i \rangle$ である．

c_i : プロセス定数

C_i : プロセス定数が満たすべき条件(論理式)

$T_i \stackrel{\text{def}}{=} \{(Act, c_j) | c_i \xrightarrow{Act} c_j\}$

□

補題 3.2 定義 *2.1* によって定めた任意のプロセスは，プロセス定数が満たすべき条件を T にすることによって，プロセス定義式の集合に等価変換できる． □

3.3 合成アルゴリズム

本システムでは，文献 [3] で提案されたアルゴリズムを用いている．ただし，一部 Prolog では実現不可能な箇所がある．それはループの作り直しをしていいかどうか判断する部分である．以下で述べるアルゴリズムではその部分は削除してあり，どこかで矛盾が生じた場合はある定められた回数だけループの作り直しを行うようにしてある．

c_i プロセス定数

D_i プロセス定義式 $(\equiv (c_i, C_i, T_i))$

S プロセス定義式の集合

c_0 プロセスの根 (初期状態)

入力 $A_1, A_2, \ldots$(満足すべき論理式の列)

出力 $S_1, S_2, \ldots$(入力された事実を満たすプロセス定義式の集合)

また，アルゴリズム内で，次のような略記法を用いる．

$S[(c_1, C_1, T_1), \ldots, (c_k, C_k, T_k)]$:$S$ 内の $D_1, \ldots, D_k$ をそれぞれ $(c_1, C_1, T_1), \ldots, (c_k, C_k, T_k)$ で置き換えたもの

$S\{x/y\}$:S 中の自由変数 y を全て x に置き換えたもの

アルゴリズム 3.3 合成アルゴリズム

mpstart :- $D_0 \equiv (c_0, \{T\}, \phi)$ % 初期状態はプロセス **0**

$mp(D_0)$.

$mp(S)$:- % S: プロセス定義式の集合

read-formula(A), % 論理式を入力.

makeproc(c_0, S, A, X), % 論理式からプロセスを合成.

write-process(X), % 結果を出力.

$mp(X)$. % 次の入力へ.

% *makeproc*(c, S, A, X)

% c: 着目しているプロセス定数

% S: プロセス定義式の集合

% A: 適用する論理式

% X: 合成された結果 (プロセス定義式の集合)

% T : 何もしないで元の値を返す

makeproc(c_i, S, T, S).

% F : 矛盾したことを意味するので，バックトラックする

% x_j : $\nu x_j.f(x_j)$ の変数

makeproc(c_i, S, x_i, S). % 既にループになっているなら何もしない.

makeproc(c_i, S, x_j, X) :- % $i \neq j$ のとき

$S' \leftarrow (S[(c_j, C_j, T_j \cup T_i)]$

$-\{(c_i, C_i, T_i)\})\{x_j/x_i\}\{c_j/c_i\}$, % c_i を c_j に融合する

makeproc(c_j, S', C_i, X). % C_i を c_j に適用する

% $\langle a \rangle B$:

makeproc$(c_i, S, \langle a \rangle B, X)$:-

%c_j が B を満たすようにする

$\exists(c_i \xrightarrow{a} c_j)$ ならば $makeproc(c_j, S, B, X)$.
$makeproc(c_i, S, \langle a\rangle B, X)$:-
　% 新しいプロセス定数 c_j を得る
　$get\text{-}new\text{-}process\text{-}constant(c_j)$,
　$makeproc(c_j, S[(c_i, C_i, T_i \cup (a, c_j)), (c_j, \{T\}, \phi)]$,
　$B \wedge \{f_a{:}[a]f_a \in C_i\}, X)$. % ただし $\wedge\phi \stackrel{\text{def}}{\equiv} T$
% $[a]B$: a で遷移可能な全てのプロセス定数に B を適用する
$makeproc(c_i, S, [a]B, X)$:-
　$c_i \xrightarrow{a} c_j$ % を満たす全ての c_j に対して
　$makeproc(c_j, S[(c_i, C_i \wedge [a]B, T_i], B, X)$.
% $B_1 \wedge B_2$: B_1 と B_2 を逐次適用する
$makeproc(c_i, S, B_1 \wedge B_2, X)$:-
$makeproc(c_i, S, B_1, Y)$, $makeproc(c_i, Y, B_2, X)$.
% $\nu x.f(x)$: 束縛変数 x を x_i に合わせて c_i にする
$makeproc(c_i, S, \nu x.f(x), X)$:-
　$makeproc(c_i, S, f(x_i), X)$.
% ループの作り直し
$makeproc(c_i, S, \nu x.f(x), X)$:-
　% $\nu x.f(x)$ の代わりに $f(\nu x.f(x))$ を適用
　$makeproc(c_i, S, f(\nu x.f(x)), X)$.

□

3.4 インタフェース

入力は，事実 (論理式) をキーボードもしくはファイルから逐次的に読み込む．μ-calculus の演算子の中には Prolog 上で扱いにくい記号がある．そのような記号は，以下の記号で代用する．

'ν' → '$', '$\langle\ \rangle$' → '<>', '.' → ':'

例えば，'$\nu x.\langle a\rangle\langle b\rangle x$' は'$ x:< a >< b > x' のように入力する．事実を 1 つ読み込む度に，それまでに入力された全ての事実を満たすプロセスを 1 つだけ出力する．

出力はプロセスを式として表示するだけではなく，ProCSuS[7] に出力を渡すことで遷移関係をグラフィカルに表示することができる．

4 実行例

4.1 例 1

簡単なプロセスを生成してみた．まず最初に，論理式 $A_1 = \langle a\rangle T$ を与えた．初期状態のプロセスは $p_0 = \{c_0 \stackrel{\text{def}}{=} \mathbf{0}\}$ なので，プロセス定数 c_0 に $\langle a\rangle T$ が適用されると，c_0 からアクション a によって遷移可能な新しいプロセス定数が得られる．その新しいプロセス定数に適用される論理式は T であり，プロセスはそのままでも論理式を満たすのでプロセス p_1 を表示し，次の入力待ちとなる．

次に論理式 $A_2 = \nu x.\langle a\rangle\langle b\rangle x$($\nu$ 記号の代用として $ を用いる) を与えると，p_2 のようなループを生成する．これは，c_1 に $\langle b\rangle x_0$ を適用すると，新しいプロセス定数を作らずに，アクション b によって c_0 に遷移するようなパスを作るためである．

3 番目に論理式 $A_3 = \langle c\rangle T$ を与えると，最初と同様に c_0 からアクション c によって遷移可能なプロセス定数を得る．

最後に論理式 $A_4 = [a][b][c]F$ を与える．この論理式の意味は，連続したアクション列 $\xrightarrow{a}\xrightarrow{b}\xrightarrow{c}$ による遷移は不可能である．ということである．現在のプロセス p_3 は，このアクション列による遷移が可能である．そこで，この矛盾をなくすために，ループの作り直しを行っている．具体的には，論理式 A_2 を与える直前と同じ状態にまでバックトラックを行い，c_0 に A_2 を適用する代わりに $\langle a\rangle\langle b\rangle\nu x.\langle a\rangle\langle b\rangle x$ を適用する．その後にもう一度，論理式 A_3, A_4 を与える．その結果として，プロセス p_4 が生成される．

4.2 例 2

次に，もう少し現実的なプロセスの合成例を紹介する．以下では，基本的な留守番電話の性質 (機能) から，プロセスを合成した．この留守番電話が備えている機能は次の 4 つである．

(1) 留守番機能がセットされているときは，電話が掛かってくると，メッセージを再生してから，録音を開始する．録音終了後は最初の状態に戻る．
(2) 留守番機能が解除されているときは，電話が掛かってくると，伝言を再生せずに，受話器が取られるのを待つ．受話器が取られ

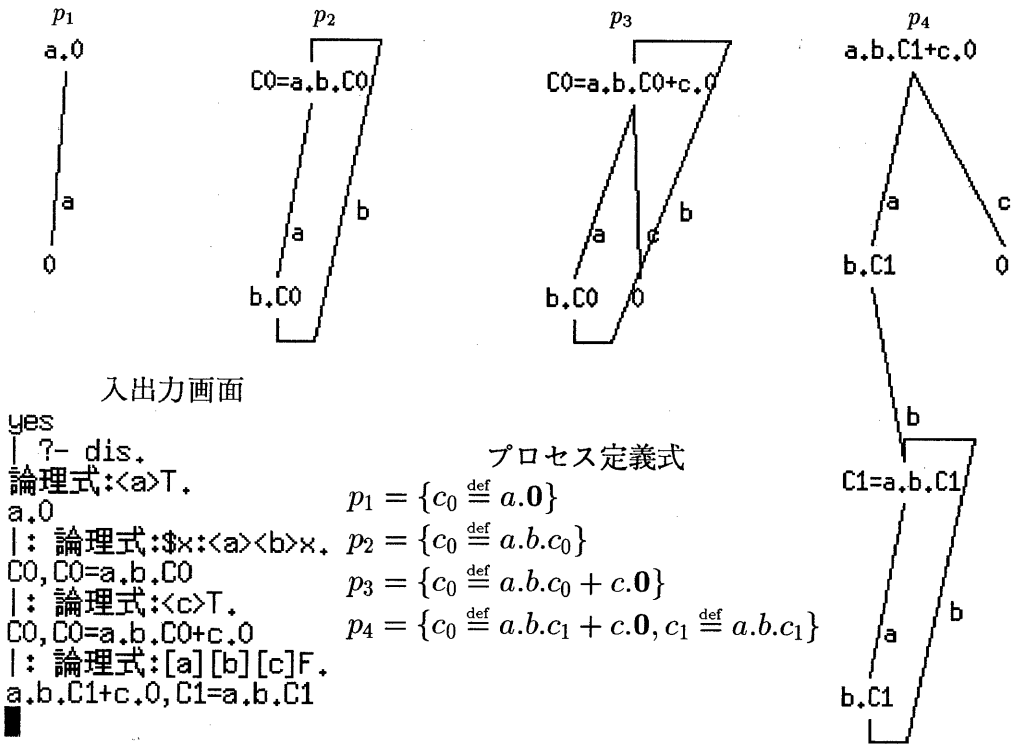

図 3: 実行例 1

た後は，会話が可能になり，受話器を置くことによって最初の状態に戻る．

(3) 伝言を再生中に受話器が取られると，会話可能になり，その後，受話器が置かれることによって最初の状態に戻る．

(4) 録音中に受話器が取られると，会話可能になり，その後，受話器が置かれることによって最初の状態に戻る．

これらの機能を論理式で表わすために，まず以下のアクションを定義する．

定義 4.1 基本的な留守番電話のアクション

ring: 電話が掛かる．

message: メッセージを再生する．

record: 録音する．

pickup: 受話器を取る．

talk: 会話をする．

hangup: 受話器を置く．

wait: 受話器が取られるのを待つ．

□

定義 4.1 で定義したアクションを用いて，この留守番電話の機能を表わすと次のようになる．

(1) $\nu x.\langle ring\rangle\langle message\rangle\langle record\rangle x$

(2) $\nu x.\langle ring\rangle\nu y.([message]F \wedge \langle wait\rangle y \wedge \langle pickup\rangle\nu z.(\langle talk\rangle z \wedge \langle hangup\rangle x))$

(3) $\nu x.\langle ring\rangle\langle message\rangle\langle pickup\rangle \nu y.(\langle talk\rangle y \wedge \langle talk\rangle\langle hangup\rangle x)$

(4) $\nu x.\langle ring\rangle\langle message\rangle\langle record\rangle(\langle pickup\rangle \nu y.(\langle talk\rangle y \wedge \langle talk\rangle\langle hangup\rangle x) \wedge [ring]F)$

この論理式を与えることによって 図 4 のようなプロセスが合成された．このプロセスは，与えた論理式を全て満たしてはいるものの，冗長な部分が見られる．例えば，c_2, c_3, c_4 はひとつのプロセス定数としておくことができる．これも今後の課題である．

$$c_0 \stackrel{\text{def}}{=} ring.c_1 + ring.message.(pickup.c_3 + record.pickup.c_4 + record.c_0)$$
$$c_1 \stackrel{\text{def}}{=} wait.c_1 + pickup.c_2$$
$$c_2 \stackrel{\text{def}}{=} hangup.c_0 + talk.c_2$$
$$c_3 \stackrel{\text{def}}{=} hangup.c_0 + talk.c_3$$
$$c_4 \stackrel{\text{def}}{=} talk.c_4 + hangup.c_0$$

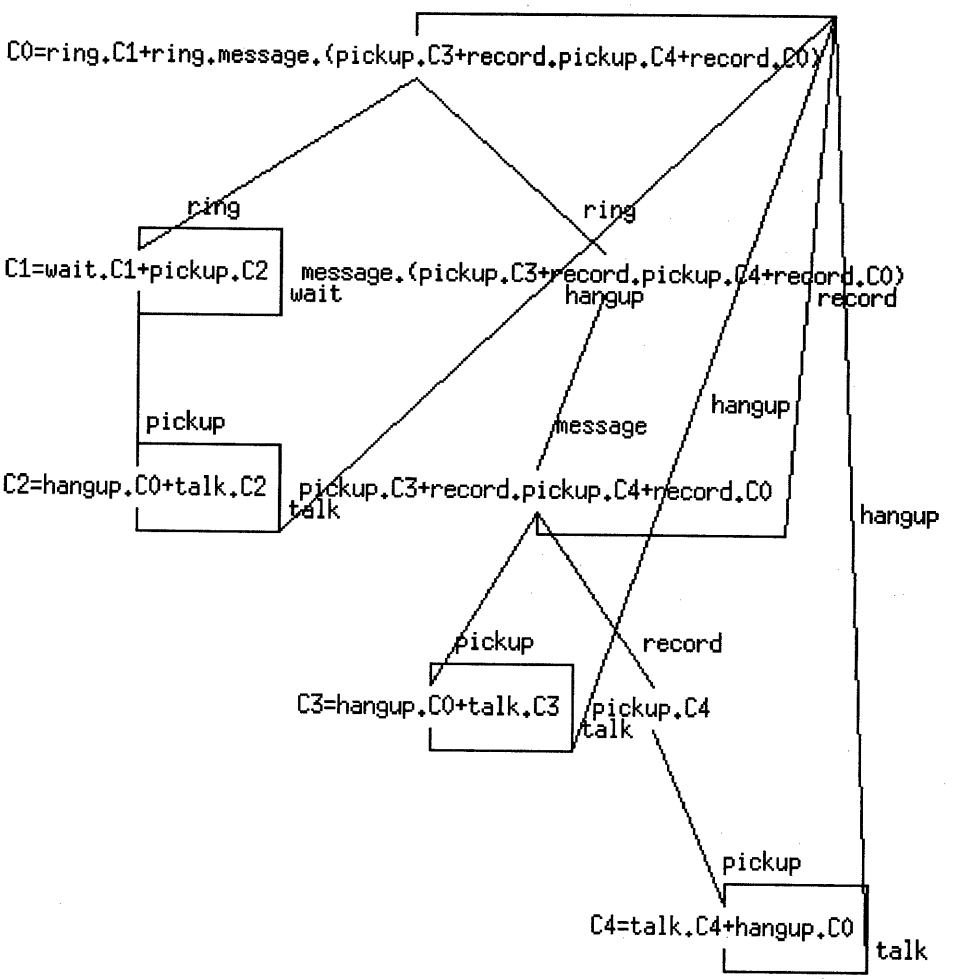

図 4: 留守番電話

5 まとめ

プロセスを合成するための支援環境を示し，実際に試作した．

今後の課題として，停止性の実現や μ-calculus の全ての演算子への対応などが挙げられる．

謝辞

本研究は一部、旭硝子財団、文部省科学研究補助金による援助を受けている。

参考文献

[1] Graf S., Sifakis J.: "A Logic for the Description of Non-deterministic Programs and Their Properties", Inf. and contr., **68**, pp.254–270(1986)

[2] Ivan Bratko : "Prolog への入門", 近代科学社 (1992).

[3] 木村成伴, 富樫敦, 白鳥則郎: "Synthesis Algorithm for Recursive Processes by μ-calculus ", 信学技報,**COMP93**,(1994-3).

[4] 木村成伴, 富樫敦, 野口正一 : "様相論理式による基本プロセスの合成アルゴリズム" , 信学論, **J75-D-I**, pp.1048–1061(1992).

[5] Milner R.: "Communication and Concurrency", Prentice-Hall(1989).

[6] 富樫敦: "代数的プロセスの計算モデル", 日本ソフトウェア科学会チュートリアルテキスト (1992).

[7] 吉田仙, 富樫敦, 白鳥則郎: "並行プロセス計算の開発・利用支援環境", 信学技報,**COMP93**,(1994-3).