ソフトウェア工学 61－1
（１９８８．７．２７）

要求仕様に基づくプロトコル仕様の完全化について

田倉　昭　　市川　晴久
ＮＴＴソフトウェア研究所

要求仕様にない動作をする可能性のある不完全なプロトコル仕様を完全化する手法を提案した．要求をメッセージシーケンス，その実現仕様であるプロトコル仕様を状態遷移機械で表現したとき，システムがプロトコル仕様上で要求外動作に伴なう異常状態に陥らないように必要なイベントを付加することにより，完全化を行う．イベントの付加による完全化は，次の二種類に分類される．
(1) 要求仕様を変更しない範囲で，プロトコル仕様を完全化する．
(2) 要求仕様，プロトコル仕様ともに変更して，すべての要求外動作に対してプロトコル仕様を完全化する．
本手法を通信ソフトウェア作成支援環境ＳＤＥに組込み，実用性の高い設計支援を行うことができる．

Completing Protocol according to Requirements

Akira TAKURA　Haruhisa ICHIKAWA
NTT Software Laboratories
3-9-11, Midori-cho, Musashino-shi, Tokyo, 180 Japan

This paper presents a method to complete protocol specifications which may behave beyond requirements specifications. The method supports the insertion of events in a given protocol specification necessary to recover the specified system from undesired states, where protocol and requirements specifications are described as a set of state transition machines and as a set of message sequences between the machines, respectively. The event insertion is categorized into two types:
(1) The protocol specifications modified, but the requirements specifications not modified.
(2) Both the protocol and requirements specifications modified.
The method can be incorporated into the communications software development support system called "SDE (Systems Design Environment)".

1．まえがき

交換ソフトウェア，通信端末ソフトウェアなどの通信ソフトウェアにおけるサービス実現部は，複数の並行動作するプロセスを用いてモデル化されることが多く，その設計において，プロセス間の通信手順（プロトコル）は重要な設計項目である．

プロトコル設計に関し，Zafiropulo等はプロトコル合成法を提案している[1].この手法では，デッドロックなどを含む不完全なプロトコル仕様を与えられたとき，適当なメッセージ送受信イベントを挿入し，過不足のないメッセージ通信を行うプロトコル仕様を合成する．これをプロトコル仕様の完全化と呼んでいる．イベントの自動挿入は，初期仕様の意図と矛盾する可能性もあるため，Zafiropulo等は合成アルゴリズムのインタラクティブな適用を想定している．

通信サービスソフトウェアの設計は次の手順を踏むことが多い．

(1) システムを，並行動作するプロセスでモデル化する．

(2) システムが実現すべき機能，状況毎にプロセスの動作をメッセージシーケンス（ＭＳＱ）と呼ばれる形式で記述する．

(3) 列挙されたメッセージシーケンスを実現するように，プロセスの動作を状態遷移機械（ＳＴＭ）として記述する．

(4) プログラムを開発する．

通信ソフトウェア開発環境ＳＤＥ（Systems Design Environment）では，ＭＳＱを“サービス要求”，ＳＴＭを“実現仕様”と捉える．ＳＤＥはサービス要求の集合（要求）の実現仕様への統合変換，プロセスプログラムの生成を自動化支援する[2].この支援は，新サービスの追加に際して，ＭＳＱで記述した新サービスと既存サービスの両方を実現するように既存ＳＴＭを自動更新する．このとき，要求の実現性，既存仕様との両立性，要求外動作の有無を検証する．ここで，要求外動作とは，変換後のＳＴＭでは起り得るが，要求されたＭＳＱの集合には規定されていない動作のことである．

ＳＤＥが検出する要求外動作には，チャネルにメッセージが残ったり，プロセスがデッドロック状態になったりするものがある．このため，上記のZafiropulo等のプロトコル合成を適用することにより，要求外動作の自動排除の可能性が考えられる[3].ＳＤＥでは，ＳＴＭのいわば意味を表わすＭＳＱがあるために，Zafiropulo等が考えたよりもより踏み込んだ自動化が可能である．本稿では，要求外動作によりシステムが異常状態に陥ることがないように自動的にＳＴＭを変換する方法を論じる（以下では，この変換もＳＴＭの完全化という）．

ＭＳＱとＳＴＭを通信ソフトウェア仕様の対等な２表現形式とし，相互変換法が検討されている[4].ＳＴＭ形式の仕様に対し，Zafiropulo等の完全化手法が適用されている．この方法では，ＭＳＱとＳＴＭの間に要求仕様と実現仕様の関係がないため，プロトコル完全化についてZafiropulo等の手法と同じ問題点を持っている．また，ここで用いられているＳＴＭ形式で記述されたプロセスは，通常，そのままでは，プログラムにならない．ＳＤＥでは，プログラム生成を可能にするため，ＳＴＭ形式に制限をつけている[2,5].

以下では，要求を変更せずにＳＴＭを完全化できるための要求外動作の性質及び要求の変更も含むＳＴＭ完全化手法（要求外動作排除アルゴリズム）を示す．本手法のＳＤＥへの具体的適用についても考察する．

2．要求仕様と要求外動作

2．1．要求仕様

通信ソフトウェアに対する要求仕様をＭＳＱで書くことができる．図１は，二つのプロセスｐ，ｑからなるcommunicate という名前のシステムを，通信ソフトウェア作成環境ＳＤＥが支援する仕様記述言語ＳＡＬで記述している．

```
object communicate() {
  p = { +(tel:) .voice()
      | +(fax:) .image() }; }
macro voice() {
  p = -q() { q = +p() -p() .talk(); }
      +q() .talk() +(onhk:) .rel_path()
      -q() { q = +p(); }; }
macro image () {
  p = -q() { q = +p() .recv(); } .send()
      +(over:) .rel_chnl() -q() { q = +p(); };}
```

図１．ＳＡＬによる仕様記述例

図１において，“object communicate()”は，シス

テムの名前がcommunicate であること，“macro voic e() ”は，voice() がマクロであることを宣言している．図２は，ＭＳＱ表現である図１から得られるＳＴＭ表現の実現仕様である．図１及び図２で，-p(m) はプロセスｐへの信号ｍの送信，+p(m) はプロセスｐからの信号ｍの受信を表わすイベントである．イベント +(tel:) のようにシステムの外部との通信には，ｐなどのプロセス名指定をつけない．また，.talk() は，ｐ，ｑ間の通話を可能にする手続きの呼出しを表わすイベントである．実現仕様から最終的にプログラムを得るために，どのプロセスも決定的に動作すると仮定する．すなわち，送信イベント，手続きの呼出しイベントは一つの状態に単独で存在しなければならない．図２において，状態◎は初期状態を表わす．また，“数字：”の形の信号は，ＳＤＥが自動的に割り付ける同期信号である．

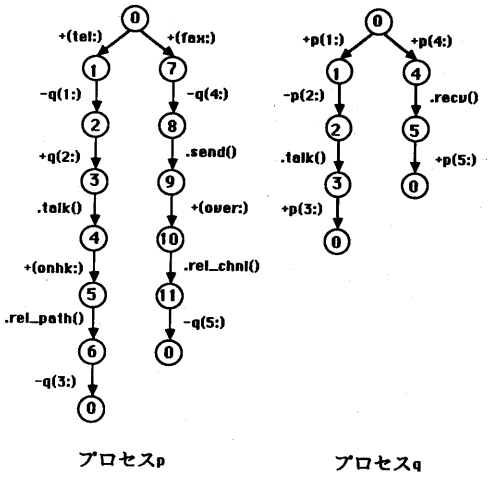

図２．communicate の実現仕様

２．２．要求外動作

システム内の各プロセスにおいて，そのプロセスの動作を，初期状態から再び初期状態に戻るまでに実行するイベント系列の任意の接頭列として定義する．システムの動作を実現仕様上で起り得る各プロセスの動作の組合せとして定義する．図２の実現仕様では，次の二つの動作b_1，b_2が起り得る．

```
b1 = { p = +(tel:) -q(1:) +q(2:) .talk()
           +(onhk:) .rel_path() -q(3:);
       q = +p(1:) -p(2:) .talk() +p(3:); }
b2 = { p = +(fax:) -q(4:) .send() +(over:)
           .rel_chnl() -q(5:);
       q = +p(4:) .recv() +p(5:); }
```

communicate に，新たに次のサービスを追加する．

```
object communicate() {
  q = +(copy:) -p() { p = +q() -q(); }
      +p() .loop() { +(start:) .copy() }
      +(end:) -p() { p = +q(); }; }
```

そのときのcommunicate の実現仕様を図３に示す．

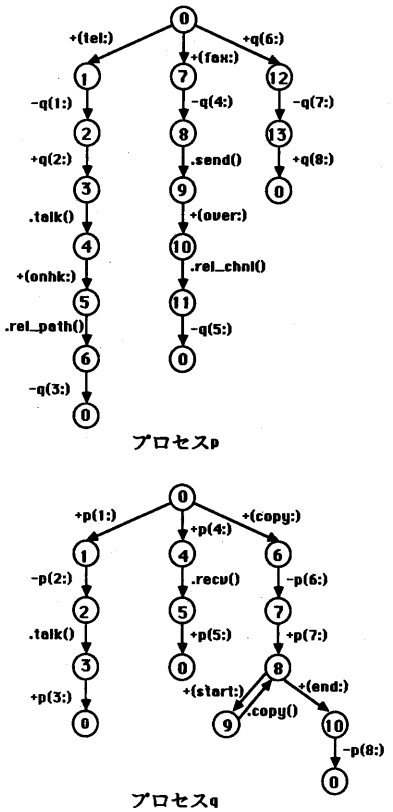

図３．更新後のcommunicate の実現仕様

新サービス追加後のcommunicate の実現仕様において，要求にはない動作b_3，b_4が起り得る．

```
b3 = { p = +(tel:) -q(1:);
       q = +(copy:) -p(7:); }
b4 = { p = +(fax:) -q(4:) .send()
           +(over:) .rel_chnl() -q(5:);
       q = +(copy:) -p(7:); }
```

例えば，b_3はプロセスｐにtel:，プロセスｑにcopy: が同時に入力された場合に起る動作である．このよう

な，要求にはないが実現仕様では起り得る動作を要求外動作と呼ぶ．

要求外動作に陥った場合，システムは正常に動作を終了しない場合がある．例えば，要求外動作b_3の場合には，プロセスｐは状態②で，プロセスｑは状態⑦でデッドロック状態になる．要求外動作b_4では，プロセスｐは初期状態⓪に戻ることができるが，プロセスｑは状態⑦でデッドロック状態になる．

２．３．要求外動作の完全化

要求外動作b_3によるデッドロック状態については，

プロセスｐに　+q(6:): ②→⓪，

プロセスｑに　+p(1:): ⑦→⓪

という遷移を実現仕様に付け加えることにより回避できる．このような初期状態に戻らない動作に対して，その動作に関わる各プロセスがチャネル上に信号を残さず初期状態に戻るようにすることを動作の完全化と呼ぶ．ところが，要求外動作に陥ったことを検出できなければ，要求外動作に対する処理を行うことができない．そこで，要求外動作の完全化を以下のように定義する．

［定義］　ある要求外動作に対して，その動作に関わるすべてのプロセスがチャネル上に信号を残さず，初期状態に戻るようにし，かつどのプロセスも要求外動作に陥ったことを検出するように遷移を付加することを要求外動作の完全化と定義する．更に，実現仕様で起り得るすべての要求外動作を完全化することをプロトコル仕様の完全化と定義する．また，完全化は一つの動作の範囲で行うこととする．　□

３．プロトコル仕様の完全化

３．１．要求に基づく完全化

要求外動作の完全化を行うとき，不足イベントを各プロセスのＳＴＭに追加する必要がある．ところが，ＳＴＭは決定的に動作するという仮定があるので，必ずしも不足イベントを付加できるとは限らない．不足イベントを付加するためには，既存サービスの変更が必要となることがある．はじめに，要求に変更を加えずに，完全化が可能な要求外動作が満たす性質を明らかにし，完全化する方法を述べる．

［E^+完全化手続き］

(1) 要求外動作を，実現仕様上でトレースしていき，プロセスが初期状態に戻るか，それ以上イベントの実行ができなくなる状態を求める．この求められた状態により，要求外動作を三種類に分類する．

E^+：初期状態⓪に戻ったプロセスがなく，かつどのプロセスもチャネルが空でない．

E^0：どのプロセスも初期状態にもどり，チャネルが空である．

E^-：E^+，E^0のどちらにも入らない．

(2) E^+の要求外動作に対しては，各プロセスに受信チャネルを空にする受信イベントを付加する．　□

上の手続きにおいて，一つの要求外動作を完全化したときに新たに要求外動作が生じることはないから，この完全化手続きは必ず停止する．

図３の実現仕様で起り得る要求外動作b_3はE^+に属する要求外動作であるから，上記のE^+完全化手続きにより，完全化することができる．完全化するに当り，各プロセスが要求外動作に陥ったことを検出したときに行う処理を .ex_i() (i=1,2)とすると，図４に示すようにb_3は完全化できる．

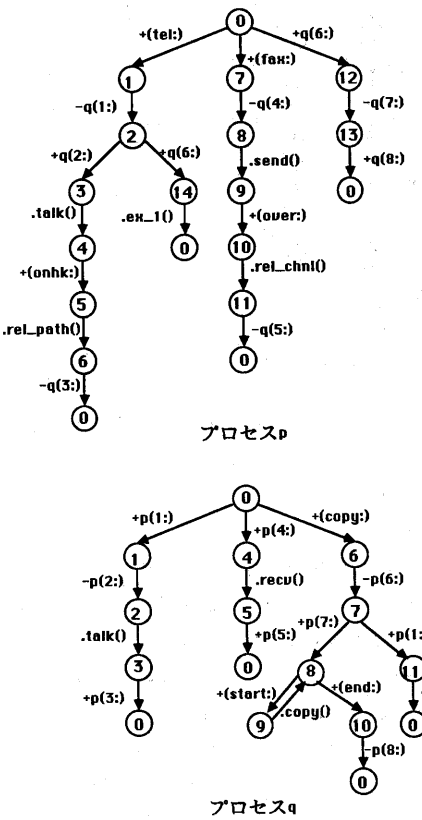

図４．E^+完全化手続きの適用例

E^+の要素は，上の手続きで完全化できる．E^0の要素は，チャネル上に信号が残ったりデッドロック状態に陥ることはないが，どのプロセスも要求外動作に陥ったことを検出できない．E^-の要素は，初期状態に戻ったプロセスのなかにチャネルが空でないプロセスがあるか，デッドロック状態にあるどのプロセスもチャネルが空であるかのいずれかである．従って，要求外動作に新たに遷移を付加する方法では，完全化することはできない．

３．２．要求の変更を伴なう完全化

E^0, E^-に属する要求外動作の完全化を考える．二つのプロセスｐ，ｑからなるオブジェクトexample において，E^0, E^-に含まれる要求外動作の完全化を考察する．オブジェクトexample に対する要求Ｒ＝{ r_1,r_2, r_3 }を次に示す．

```
r1 = { p = +(a:) -q(a1:) +q(a2:);
       q = +(a:) +p(a1:) -p(a2:); }
r2 = { p = +(b:) +q(b1:) -q(b2:);
       q = +(b:) -p(b1:) +p(b2:); }
r3 = { p = +(c:) -q(b2:) +q(b1:);
       q = +p(b2:) -p(b1:); }
```

要求Ｒに対する実現仕様を図５に示す．

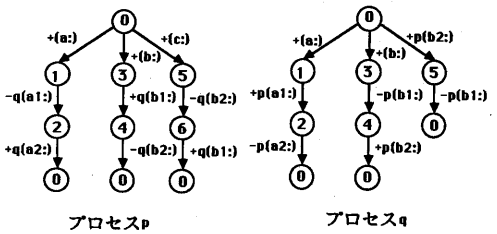

図５．example の実現仕様

この実現仕様には，次の４要求外動作が起り得る．

```
e1 = { p = +(a:) -q(a1:);
       q = +(b:) -p(b1:); }
e2 = { p = +(c:) -q(b2:);
       q = +(a:); }
e3 = { p = +(c:) -q(b2:) +q(b1:);
       q = +(b:) -p(b1:) +p(b2:);
e4 = { p = +(b:);
       q = +(a:); }
```

これらをE^+完全化手続きにより分類すると，

E^+ = { e_1, e_2 },
E^0 = { e_3 },
E^- = { e_4 },

となる．

E^0に属するe_3の完全化を考える．e_3は，二つの要求r_2, r_3から生じる要求外動作である．要求r_2, r_3に関わる各プロセスの動作を変更することにより，e_3の完全化を行う．これはプロセスｐのr_3の動作とe_3の動作を分離し，プロセスｑのr_2の動作とe_3の動作を分離することにより可能となる．すなわち，新しい信号n1:, n2:, n3:, n4:を用意し，r_2, r_3, e_3の動作の最後尾にイベントを次の順で付加する．①r_3のｐに+q(n1:)，r_3のｑに-p(n1:)，②e_3のｐに+q(n2:)，e_3のｑに-p(n2:)，r_2のｑに-p(n2:)，③r_2のｐに-q(n3:)，r_2のｑに+p(n3:)，④e_3のｐに-q(n4:)，e_3のｑに+p(n4:)，⑤ｐ，ｑが要求外動作を検出したときに行う処理を，.ex_i(i=1, 2)とすると，e_3のｐに.ex_1()，e_3のｑに.ex_2()．

```
r2 = {
  p = +(b:) +q(b1:) -q(b2:) -q(n3:);
  q = +(b:) -p(b1:) +p(b2:) -p(n2:) +p(n3:); }
r3 = {
  p = +(c:) -q(b2:) +q(b1:) +q(n1:);
  q = +p(b2:) -p(b1:) -p(n1:); }
e3 = {
  p = +(c:) -q(b2:) +q(b1:) +q(n2:)
      -q(n4:) .ex_1();
  q = +(b:) -p(b1:) +p(b2:) -p(n2:)
      +p(n4:) .ex_2(); }
```

上の手続きは，次のE^0完全化手続きとして一般化できる．ところで，各プロセスのイベント列において要求外動作を検出後に行うべき処理 .ex_i()等の挿入を記述するのは冗長であるから，以降の完全化手続きにおいては省略する．

［E^0完全化手続き］

(1) 要求の集合Ｒ＝{ r_1,･･･,r_m }から得られる実現仕様で起り得る要求外動作 e ($\in E^0$) を，

$e = \{ p_i = b_i \mid i = 1,\cdots,n \}$

とする．各$(p_i = b_i) \in e$ に対して，

$inc(b_i) = \{ r_j \mid b_i$ は，r_jにおけるプロセス p_i の動作の接頭列である$\}$

と定義すると，どの $(p_i=b_i) \in e$, $r_j \in inc(b_i)$ に対しても，$r_j \notin inc(b_k)$ であるような $(p_k=b_k) \in e$ が存在する．添字ｉ，ｊに対するこのようなｋの任意の一

つ（例えば，最小のｋ）を，<i,j> とおき，すべての $(p_i=b_i) \in e, r_j \in inc(b_i)$ に対して求める．

(2) 各 b_i (i=1,･･･,n)に対して，以下の処理を行い，プロセス p_i において，b_i と $inc(b_i)$ に含まれる要求の動作とを分離する．動作 $b_1,･･･,b_{i-1}$ に対する処理が終了した時点で，要求 $r_1,･･･,r_m$ ，要求外動作 e が

$r_j = \{ ･･･; p_k = a_{kj} ; ･･･ \}$, j=1,･･･,m

$e = \{ ･･･, p_k = c_k ; ･･･ \}$

となったとする．このとき，g_{ij}:, h_{ij}: (i=1,･･･,n; j=1,･･･,#(inc(c_i))) を新しい信号として，要求及び要求外動作に以下のようにイベントを付加する．ここで，#(inc(c_i))は inc(c_i) の要素数を表わす．$r_j \in inc(c_i)$, $r_k \in inc(c_{<i,j>})$ とすると，

$r_j = \{ ･･･; p_i = a_{ij} + p_{<i,j>}(g_{ij}:);･･･;$

$p_{<i,j>} = a_{<i,j>j} - p_i(g_{ij}:);･･･ \}$,

$r_k = \{ ･･･; p_i = a_{ik} + p_{<i,j>}(h_{ij}:);･･･;$

$p_{<i,j>} = a_{<i,j>k} - p_i(h_{ij}:);･･･ \}$

$e = \{ ･･･; p_i = c_i + p_{<i,j>}(h_{ij}:);･･･;$

$p_{<i,j>} = c_{<i,j>} - p_i(h_{ij}:);･･･ \}$

この結果，図６に示すようにプロセス p_i において要求外動作が検出できる．

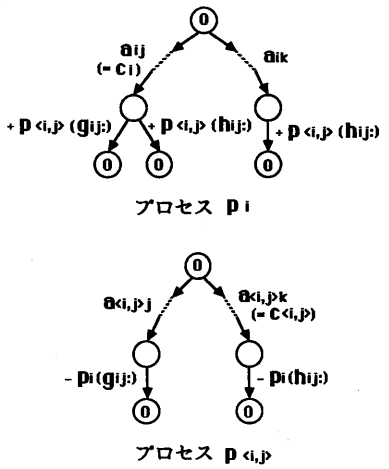

図６．要求外動作の完全化　　□

次に E^- に属する要求外動作の完全化を考える．E^o と同様に，要求の変更が必要となる．E^- に属する要求外動作において，プロセス p_i(i=1,･･･,n) の動作がたどり着いた状態を st_i,そのときのプロセス p_j からの受信チャネルの状態を ch_{ij} とおくと，① st_i＝◎，ch_{ij} ＝ φ (j=1,･･･,n)，② st_i＝◎，$ch_{ij} \neq \phi$ であるチャネルがある，③ $st_i \neq$ ◎，$ch_{ij} = \phi$ (j=1,･･･,n)，④ $st_i \neq$ ◎，$ch_{ij} \neq \phi$ であるチャネルがあるのいずれかであり，しかも②または③に属するプロセスが少なくとも一つはある．①のプロセスは，②または③のプロセスが要求外動作であることを検出すれば，そのプロセスを E^o 完全化手続きにおける $p_{<i,j>}$ として，E^o 完全化手続きが適用できる．④のプロセスは，E^+ 完全化手続きがそのまま適用できる．従って，②または③のプロセスを要求外動作が検出可能な形で $ch_{ij} = \phi$ である初期状態に戻すことができればよい．

［E^- 完全化手続き］

(1) はじめに，②及び③のプロセスの完全化を行う．要求の集合 R＝ { $r_1,･･･,r_m$ }から得られる実現仕様に含まれる要求外動作 e ($\in E^-$) を，

$e = \{ p_i = b_i \mid i=1,･･･,n \}$

とする．E^o 完全化手続きと同様に，②，③のプロセスに関して，すべての $(p_i = b_i) \in e$, $r_j \in inc(b_i)$ に対して，<i,j> を求める．

(2) (1) で<i,j> を求めた b_i(i=1,･･･,n)に対して以下の処理を行い，b_i と $inc(b_i)$ の要求とを分離する．動作 $b_1,･･･,b_{i-1}$ に対する処理が終了した時点で，要求 $r_1,･･･,r_m$,要求外動作 e が

$r_j = \{ ･･･; p_k = a_{kj}; ･･･ \}$, j = 1,･･･,m

$e = \{ ･･･; p_k = c_k;･･･ \}$,

となったとする．このとき，要求及び要求外動作に対して，g_{ij}:(i=1,･･･,n; j=1,･･･,#(inc(c_i)))を新しい信号として以下に示すようにイベントを付加する．

(i) p_i が②に属するとき，チャネルに入っている信号をすべて受取るイベント列を，d_i(= $+q_1(h_1:)･･･+q_f(h_f:)$)とすると，

$e = \{ ･･･; p_i = c_i d_i;･･･ \}$.

各 $r_j \in inc(c_i)$ に対して，r_j を実行したときに，プロセス p_i の動作が c_i になったときのプロセス $p_{<i,j>}$ の動作を pred($p_{<i,j>}, p_i, c_i; r_j$) ，プロセス $p_{<i,j>}$ の残りの動作を succ($p_{<i,j>}, p_i, c_i; r_j$) と書けば，

$r_j = \{ ･･･; p_i = c_i + p_{<i,j>}(g_{ij}:); ･･･;$

$p_{<i,j>} = pred(p_{<i,j>}, p_i, c_i; r_j) - p_i(g_{ij}:)$

$succ(p_{<i,j>}, p_i, c_i; r_j); ･･･ \}$.

(ii) p_i が③に属するとき，$r_j \in inc(c_i)$ とすると，

$e = \{ ･･･; p_i = c_i + p_{<i,j>}(g_{ij}:); ･･･;$

$p_{<i,j>} = pred(p_{<i,j>}, p_i, c_i; r_j) - p_i(g_{ij}:)$

$succ(p_{<i,j>}, p_i, c_i; r_j); ･･･ \}$.

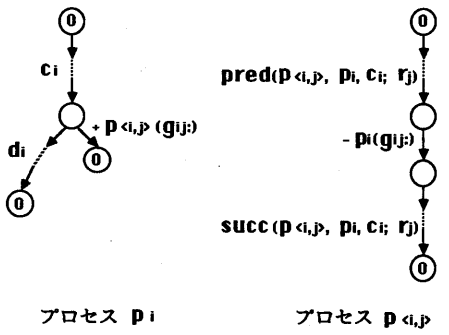

図７． $st_i = ◎$，$ch_{ij} \neq \phi$ の動作の完全化

$r_k \in inc(c_{<i,j>})$ に対して，

r_k = { ･･･; p_i = pred(p_i,$p_{<i,j>}$,$c_{<i,j>}$;r_k)
+$p_{<i,j>}$(g_i:)
succ(p_i,$p_{<i,j>}$,$c_{<i,j>}$;r_k); ･･･;
$p_{<i,j>}$ = pred($p_{<i,j>}$,p_i,c_i;r_k)
-p_i(g_i:) succ($p_{<i,j>}$,p_i,c_i;r_k); ･･･ }.

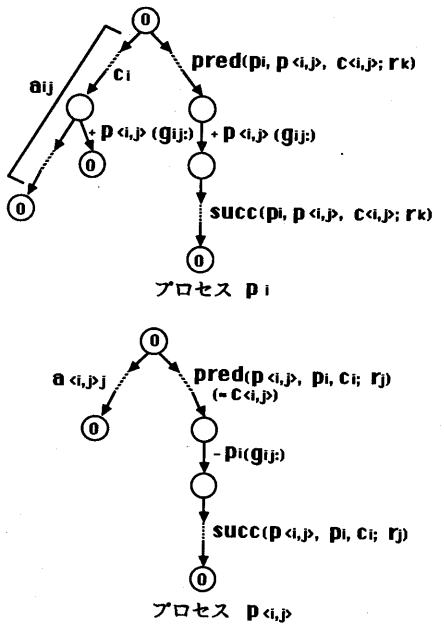

図８． $st_i \neq ◎$，$ch_{ij} = \phi$ の完全化

(3) ①のプロセスp_iに関して，(1),(2) で完全化したプロセスをE^0完全化手続きにおける$p_{<i,j>}$と対応させることにより，E^0完全化手続きを適用する．

(4) ④のプロセスはE^+完全化手続きを適用する．

□

上記のE^0，E^-完全化手続きともに，一つの要求外動作を完全化したときに新しい要求外動作が生じることはない．従って，どちらの手続きも停止する．

４．通信ソフトウェア作成環境ＳＤＥへの適用

４．１．通信ソフトウェア作成環境ＳＤＥ

通信ソフトウェア作成環境ＳＤＥはメッセージシーケンスを書く仕様記述言語ＳＡＬで書かれた要求を自動的にプログラムに変換する．この過程で既存の実現仕様を，既存要求，新規要求の両方を実現するように更新する．仕様の追加に当り仕様検証を行う．検証項目は，①追加仕様の実現性，②追加仕様と既存仕様の両立性，③追加仕様と既存仕様の組合せで生じる要求外動作の有無である．図９にＳＤＥにおける処理の流れを示す．

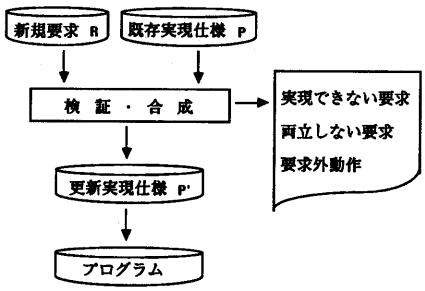

図９．ＳＤＥにおける処理の流れ

４．２．ＳＤＥにおける完全化

本稿では，実現可能で，かつ既存仕様と両立する要求を既存実現仕様に追加したときに起る要求外動作を完全化の対象とする．ＳＤＥに完全化手続きを組込むには，要求外動作を含む実現仕様を直接変換する方法と，完全化した要求外動作を再び要求としてＳＤＥに与える方法がある．ＳＤＥでは，すべての仕様をＳＡＬで管理している．従ってＳＤＥに完全化手続きを組込むには，後者の完全化した要求外動作をＳＡＬによる記述で出力し，ＳＤＥに入力するほうが望ましい．この方法では，完全化した要求外動作の既存の実現仕様への追加処理が余計にかかるが，これは完全化した要求外動作の長さに比例する時間で行えるので，時間計算量のオーダは変らない．ＳＤＥに完全化手続きを

組込んだときの処理を図１０に示す．

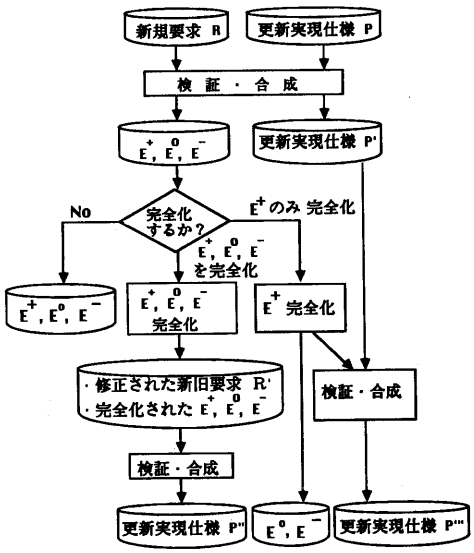

図１０．ＳＤＥにおける完全化

ＳＤＥにおける完全化の適用例を示す．二つのプロセスｐ，ｑからなるexceptに対する要求を次に示す．

```
object except () {
   p = +(tel:) -q() .talk() +q();
   q = +p() .talk() +(ofhk:) -p();

   p = +q() .recv() -q();
   q = +(pc:) -p() .send() +p(); }
```

オブジェクトexceptには，E+に含まれる要求外動作eが含まれる．完全化した結果がe'である．

```
 e  = { p = +(tel:) -q() .talk() +q();
        q = +(pc:) -p() .send() +p(); }
 e' = { p = +(tel:) -q() .talk() +q() .ex_1();
        q = +(pc:) -p() .send() +p() .ex_2(); }
```

e'をＳＤＥに入力した結果が図11であり，完全化後のプロセスｐの実現仕様である．

５．あとがき

プロトコル仕様を完全化する手法を提案した．本手法は，要求を与え，これの範囲でプロトコル仕様を完全化する方法と要求の変更を伴なう範囲で完全化する方法とからなり，どちらの場合にもプロトコル仕様をプログラム生成できる範囲に制限することを特徴とする．完全化の通信ソフトウェア作成環境ＳＤＥへの具体的適用法も示した．この適用例からも分るように実用性の高い設計支援ができる．

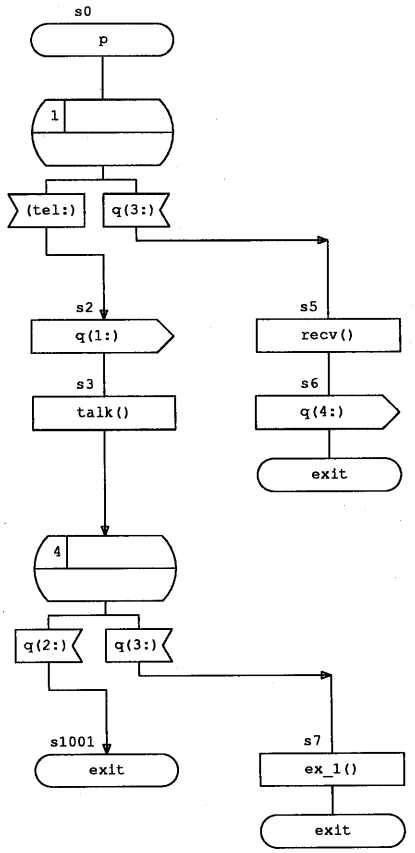

図１１．ＳＤＥにおける完全化の適用例

［参考文献］

[1] P.Zafiropulo, C.H.West他：Towards Analyzing and Synthesizing Protocols,IEEE Trans Commun., Vol.COM-28,1980

[2] 市川，伊藤，柴崎：Protocol-Oriented Service Specifications and their Transformation into CCITT Specification and Description Language, Trans. IECE E69, 1986

[3] 田倉，市川：要求仕様に基づくプロトコル仕様の完全化，情処学会第36回全国大会2M-5，1988

[4] 角田，若原：プロトコル仕様の時系列表現と状態遷移表現の相互変換，昭和62年信学会情シ部門全国大会526,1987

[5] CCITT ：Recommendation Z.100(Specification and Description Language), 1986