〔公　開〕

ＴＲ－Ｃ－００８８

通信サービス仕様から通信ソフトウェアを生成する手法

河　田　慶　三
Keizo　KAWATA

１９９３　４．１４

ＡＴＲ通信システム研究所

通信サービス仕様から通信ソフトウエアを生成する手法

河田慶三

1993年4月14日

もくじ

図一覧

1　はじめに

本稿では、通信サービス仕様記述から通信ソフトウエアを自動的に生成するシステムについて記述する。

我々の通信ソフトウエア生成システムは、ＳＴＲとＳＴＲ／Ｄという２つの言語システムから構成されている。ＳＴＲは通信サービス開発者がサービスを記述するための宣言型言語であり、通信システムの外部から観測可能な事象のみを記述する。そのため、ＳＴＲは通信ネットワークに習熟していない人でも容易に理解できる。

ＳＴＲ／Ｄは、通信ソフトウエアを生成する上で必要な情報でＳＴＲ中には記述されていないものを記述するための宣言型言語である。通常、ＳＴＲ／Ｄは通信ネットワークに習熟した通信システム開発者が記述する。

ここでは、ＳＴＲからプロセス仕様を生成する２種類のアルゴリズムとＳＴＲ／Ｄ言語仕様について記述する。

2　通信システム

本研究において、通信システムは図1に示すように、端末層、ソフトウエア・プロセス層、通信媒体の３層構造をしていると仮定する。

プロセスは分散配置されており、各プロセスは識別子（ＰＩＤ）を持つ。プロセスは各時点において、特定の状態を持つ。状態中には、接続している周辺プロセスのＩＤが含まれる。プロセスは状態に応じて端末を制御する。システム中のすべてのプロセス上には、同一のソフトウエアが実装される。そのため、特別な機能を持ったプロセスは存在せず、プロセスはすべて均質である。

プロセスの集合ＰＤに対して、ＰＤ中のプロセスの状態及びそれらの間の関係をＰＤの広域状態と呼ぶ。また、システムを構成するすべてのプロセスに関する広域状態を、システムの状態と呼ぶ。

プロセス集合ＰＤの広域状態遷移とは、ＰＤ中の特定のプロセスが対応する端末から特定の入力を受けることによって、ＰＤの広域状態（始広域状態）が新しい広域状態（次広域状態）に遷移することである。

2.1　通信サービスの定義

通信システムに対して上記の仮定をおいた上で、通信サービスを次のように定義する。

- 通信サービスとは、広域状態遷移を与えるものである。

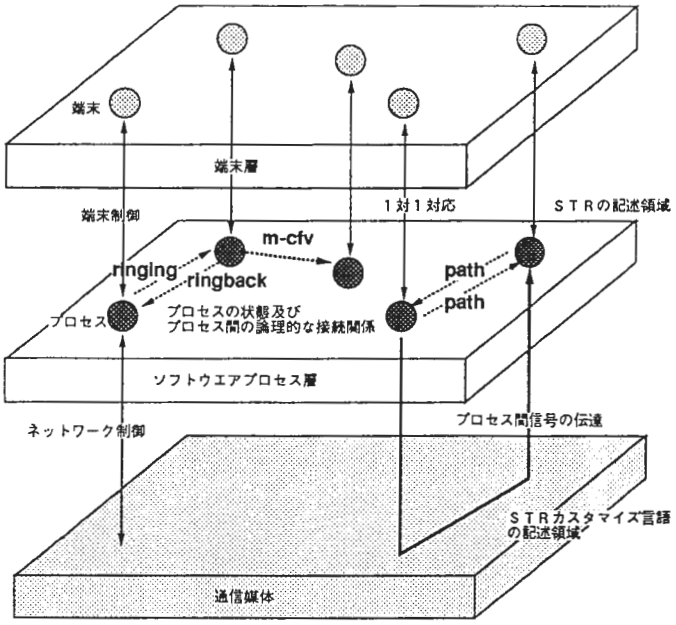

図 1: 通信システムのモデル

3 通信ソフトウエア生成システム

通信サービス仕様を記述するための言語ＳＴＲを導入する。ＳＴＲはシステムの外部から観測可能な事象を記述する言語であり、端末層とソフトウエアプロセス層の間の関係を記述する。

通信ソフトウエアを生成するためには、ネットワーク内部の情報も必要である。そのため、ＳＴＲを補間する言語としてＳＴＲ詳細化言語（ＳＴＲ／Ｄ）を導入する。

ＳＴＲはネットワークの専門家とは限らない通信サービス開発者が記述する言語であり、ＳＴＲ／Ｄはネットワークの専門家であるシステム開発者が記述する言語である。

図2に、通信ソフトウエア生成システムの概要を図示する。

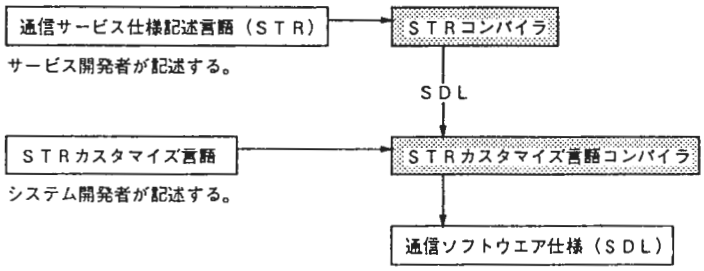

図 2: 通信ソフトウエア生成システム

この図に示すように、２段がまえの構成をとることによって、利用者によって定義された通信サービス仕様は多様なネットワークに対応することが可能になる。

4　通信サービス仕様記述言語ＳＴＲ

ＳＴＲでは広域状態を論理式で表現し、広域状態遷移を、プロダクション規則（ＳＴＲ規則）の集合で記述する。ＳＴＲ規則は、始広域状態、イベント（端末入力）、次広域状態の３項で表現される。広域状態は複数のプロセスの状態から構成される。プロセスの状態及びイベントは述語で表現される。

通信サービスの動作をＳＴＲ記述で記述したものを図3に示す。この規則は、ダイヤルトーン状態にあるプロセスＡから、空(idle)状態であるプロセスＢにダイヤルしたとき、プロセスＢが着信転送を設定(m-cfv(B,C))しており、その転送先であるプロセスＣが空状態の時には、Ｂへの呼びは、Ｃに転送され、Ａは呼返音(ringback-tone)受信状態へ遷移し、Ｃは呼出音(ringing-tone)の鳴っている状態に遷移することを規定している。

図3の規則で、dialtone,idle等はプロセスの状態を表現する述語であり、dialはイベントを表現する述語である。なお、一つのプロセスが同一の述語を２個以上持つことはないものとする。

```
dialtone(A), idle(B), m-cfv(B,C) , idle(C)    /* 始広域状態  */
dial(A,B) :                                   /* イベント */
ringback(A,C), ringing(C,A),                  /* 次広域状態  */
pingring(B,A), m-cfv(B,C).
```

図 3: 通信サービス規則例

ＳＴＲは広域状態遷移を記述する規則以外に、あり得ない状態を指定する禁止状態指定規則を持つ。禁止状態指定規則は、互いに共存できない状態記述プリミティブの集まりで表現される。図4の規則の場合、dialtoneとringingとringbackのどれか２つをプリミティブとして持つ状態を禁じる。

$$\{dialtone, ringing, ringback\}$$

図 4: 禁止状態指定規則例

4.1 グラフ文法

ＳＴＲ規則中の始広域状態記述とイベント記述は、あわせて１個のラベル付きの有向グラフとして表現できる。また、次広域状態記述も１個のラベルつき有向グラフとして表現できる。

図3の規則に対応するグラフ表現を図5に示す。図5において、規則中の ringback(A,C) という記述は、点 A から点 C への ringback というラベルを持つ有向辺として表現される。

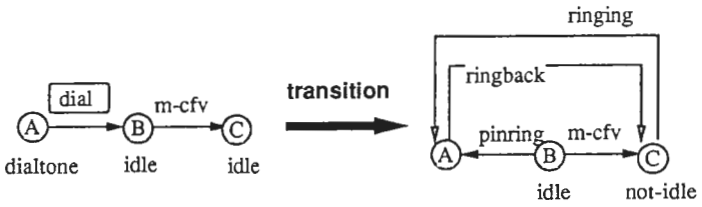

図 5: サービス規則に対するグラフ表現

このように、ＳＴＲ規則は、グラフの書き換え規則とみなすことができる。通信システム全体の状態も、グラフで表現できる。これをシステム・グラフと呼ぶことにすると、ＳＴＲ規則の意味するところは、次のように定義できる。

（ＳＴＲ規則の意味）ＳＴＲ規則は、システム・グラフ中の部分グラフで規則の始グラフと同形なものを、規則の次グラフと同形なものに置き換えることを指定する。

図6に、ＳＴＲ規則が適用される状況を図示する。

4.2 規則の優先順位

イベントが生起したとき、複数の規則が適用可能になる場合がある。このような規則間の競合を回避するため、規則間に優先順位をつける。ここでは、同一イベントを持つ規則の集合に対して、グラフの包含関係に基づいた次のような半順序を導入する。同一のイベントを持つ２つの規則 r1,r2 に対して、 r1 の始グラフが r2 の始グラフを含む時その時に限り、規則 r1 は規則 r2 よりも優先度が高い。この順序は全順序ではないため、規則の競合を完全に回避することはできない。上の半順序を用いても規則の競合を回避できない時、適用可能な規則のどれか１つが実行されるものとする。

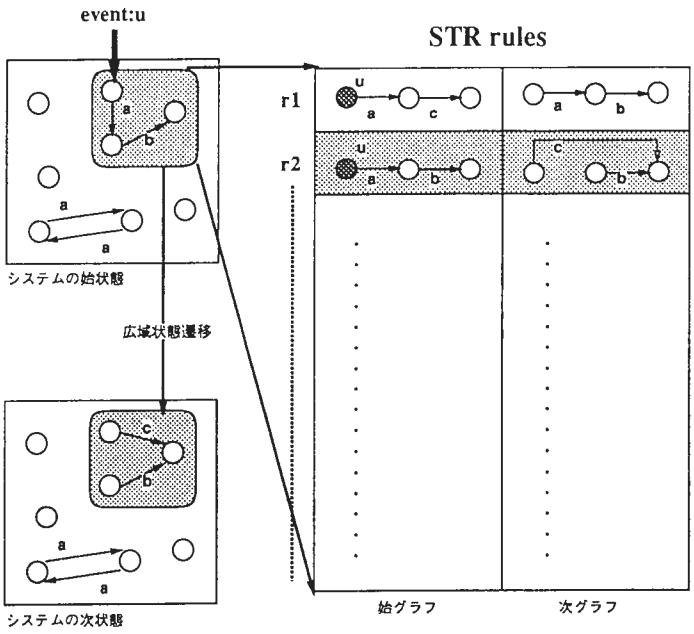

図 6: ＳＴＲ規則の適用

4.3 ＳＴＲ規則のクラス

連結な有効グラフを次の４つの種類に分類する。

(1)string 属性付き点が、イベント生起点を先頭にして一列に並んでいるグラフである。この列に逆行する辺は存在しない。

(2)trunk イベント生起点を先頭にして、すべての属性付き点を通るパスが存在するグラフである。

(3)tree サーキット（始点と終点が等しいパス）が存在しないグラフである。

(4)connected-graph 連結なグラフである。

上記４種類のグラフを図７に図示する。
ＳＴＲ規則は、規則中の始グラフの種類に応じて４つのクラスに分類できる。

5 クラス２ＳＴＲのプロセス仕様

5.1 プロセス間通信による広域状態遷移の実現

我々のシステムにおいて、プロセスは局所的な情報しか保持しない。プロセスは、通信することによって周辺プロセスと情報を交換し、広域的な状態を把握して、適用できるＳＴＲ

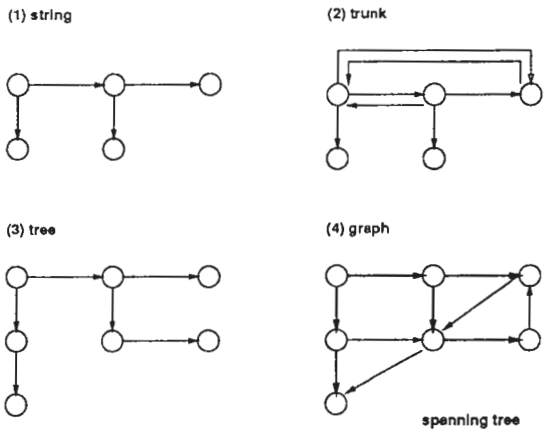

図 7: ＳＴＲ規則のクラス

規則を決定する。

プロセスの状態

各時点におけるシステム中のプロセスの状態は、変数を含まない素式の連言で記述できる。各素式の第１項には、そのプロセス自身の識別子（ＰＩＤ）が入る。たとえば、識別子 p_1 を持つプロセスで、ダイアルトーンが鳴っていて、識別子 p_2 を持つプロセスへ着信転送が設定されている状態は、次式で記述できる。

$$dialtone(p_1), m - cfv(p_1, p_2)$$

このように状態記述中にＰＩＤを含むので、プロセスの状態数はシステム中に存在するプロセスの個数に依存する。しかし、状態記述中のプロセスＩＤを無視すれば、プロセスの状態はシステムに依存せずにＳＴＲ記述だけから、有限個に分類できる。その分類番号を状態ＩＤと呼ぶことにすると、プロセスの状態は、状態ＩＤとＰＩＤの列で表現できる。

メッセージ

プロセス間で送受信するメッセージとして、探索依頼メッセージとそれに対する応答メッセージの２種類用意する。

探索依頼メッセージは、自分の状態を隣接プロセスに伝えることによって、隣接プロセスに適用できる規則の決定を依頼するために用いられるメッセージであり、応答メッセージは、決定された規則を通知するために用いられるメッセージである。応答メッセージの特殊なものとして、拒否メッセージがある。これは、適用できる規則が存在しないことを意味するメッセージである。

探索依頼メッセージは、適用できる可能性のある規則の集合を陰に表している。順次、探索依頼メッセージが伝えられていくに従って、適用できる可能性のある規則が絞り込まれ、最後に規則が一意に決定される。

メッセージはプロセスの状態を伝達する。そのため、メッセージ中にはメッセージＩＤとは別にメッセージを発信したプロセスの保持するプロセスのＩＤが含まれる。

プロセス間通信

プロセスＰ１でイベントが生起した場合、Ｐ１の状態だけで適用を決定できるＳＴＲ規則が存在するならば、その規則に応じて状態遷移を行なう。そのような規則が存在しないならば、Ｐ１は隣接プロセスの１つＰ２へ探索依頼メッセージを送信する。Ｐ２から応答メッセージを受信すると、応答メッセージの指示する規則に従って状態遷移を行なう。応答メッセージとして拒否メッセージを受信した場合は、別の隣接プロセスＰ３へ探索依頼メッセージを送信し同様な処理を続ける。

隣接プロセスＰ２がＰ１から探索依頼メッセージを受信した場合、Ｐ１からの情報とＰ２の状態で適用を決定できる規則が存在するならば、Ｐ１にその規則を意味する応答メッセージを返信し、Ｐ２はその規則に従って状態遷移を行なう。そのような規則が存在しないならば、隣接プロセスの１つＰ４へ探索依頼メッセージを送信する。この探索依頼メッセージには、Ｐ２の状態とＰ１の状態を含ませる。Ｐ４から応答メッセージを受信すると、応答メッセージの指示する規則に従って状態遷移を行なう。応答メッセージとして拒否メッセージを受信した場合は、別のプロセスＰ５へ探索依頼メッセージを送信し同様な処理を続ける。図8に、プロセス間通信の概略図を示す。

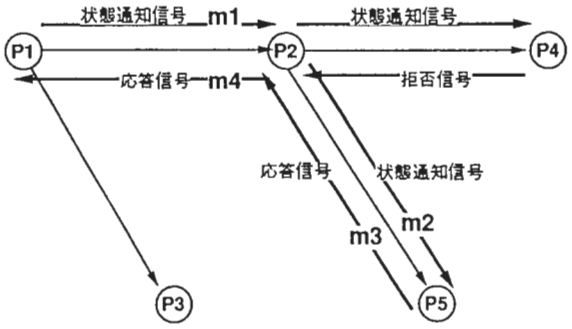

図 8: クラス２プロセス間通信例

プロセスＩＤの送信

図9は、図8中のプロセスＰ２が状態ｓ１でＰ５からメッセージｍ３を受信した後、ｍ３の指定する規則ｒ１に従って状態ｓ２へ遷移する状況を示している。

ｍ１中にはＰ１の保持するＰＩＤが含まれ、ｍ３中にはＰ５の保持するＰＩＤが含まれる。ｓ１からｓ２に遷移する際に、保持しているＰＩＤをｍ１とｍ３中のＰＩＤ情報を利用して再設定する。

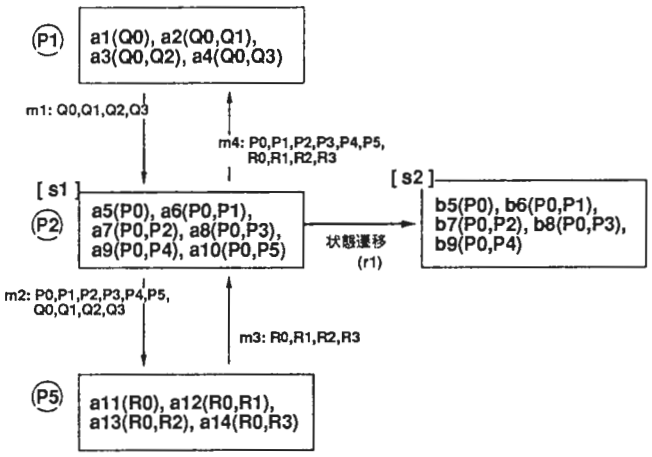

図 9: ＰＩＤの送信例

5.2 通信ソフトウエア

図10に、我々の通信ソフトウエアの動作概要をＳＤＬ形式で示す。図10において、(3)のＰＩＤの比較と(8)のＰＩＤの再設定を行なっている部分が、従来のオートマトン・モデルと異なる部分である。

6 クラス２ＳＴＲからＳＤＬを生成するアルゴリズム

プロセス動作仕様の生成過程は、次の３つのステップに分けることができる。

（ステップ１） 規則の分類
ＳＴＲ規則を規則中のイベント及び始グラフ中の点近傍によって分類し、規則の分類木を作成する。

（ステップ２） プロセス動作パターンの生成
分類木を探索することによって、プロセスの動作パターンを生成する。

（ステップ３） プロセス動作パターンの合成
プロセス動作パターンを用いて、アイドル状態から到達可能なすべての状態を生成し、各状

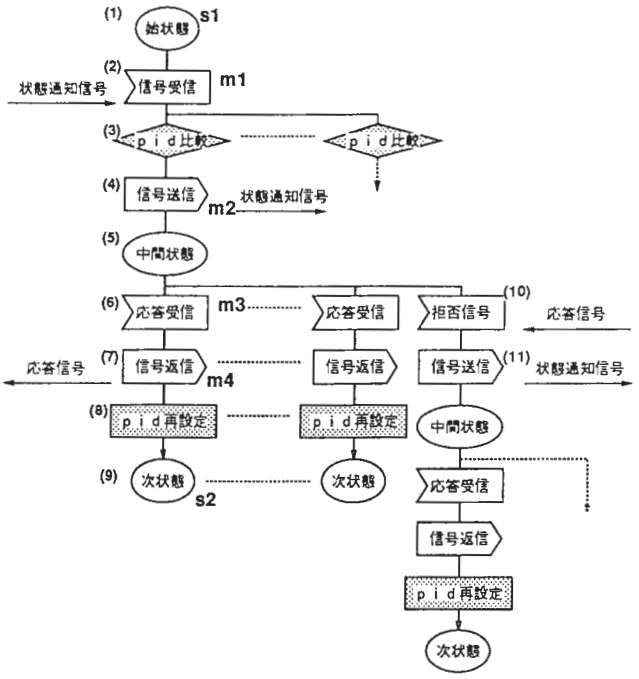

図 10: クラス 2 通信ソフトウエア

態におけるプロセスのふるまいを生成する。この時に、ＳＴＲ記述中の否定論理式を展開する。

6.1 規則グラフ中の点近傍

ＳＴＲ規則の始グラフにおいて、ラベルを持つ点はイベント生起点を先頭にして一列に並んでいる（幹）。この並びに応じて幹中の点の間に親子関係を設定できる。この親子関係に基づいて、始グラフ中の点の近傍を以下の 3 種類導入する。

単純近傍 ｖとｖから出ている辺及びそれらに付随するラベルで構成される近傍である。単純近傍はプロセスの状態を表現している。

祖先近傍 ｖの単純近傍の各辺に対して、その辺の終点ｗがｖの祖先を始点とする辺の終点になっている場合、祖先の種類（親、祖父等）と祖先を始点とする辺に付随しているラベルを単純近傍に付加したもの。ｗが複数の祖先に対して、それらを始点とする辺の終点となっている場合、もっとも関係の近い祖先から出ている辺を優先し、その情報だけを付加する。

長男近傍 ｖの祖先近傍中の辺の内、ｖの長男に向かう辺に長男への辺であることを示すラベルを付けたもの。

図11に、規則グラフとその点近傍の例を示す。この例において、点１の祖先近傍は単純近傍と同じである。また、点３の長男近傍は祖先近傍と同じである。

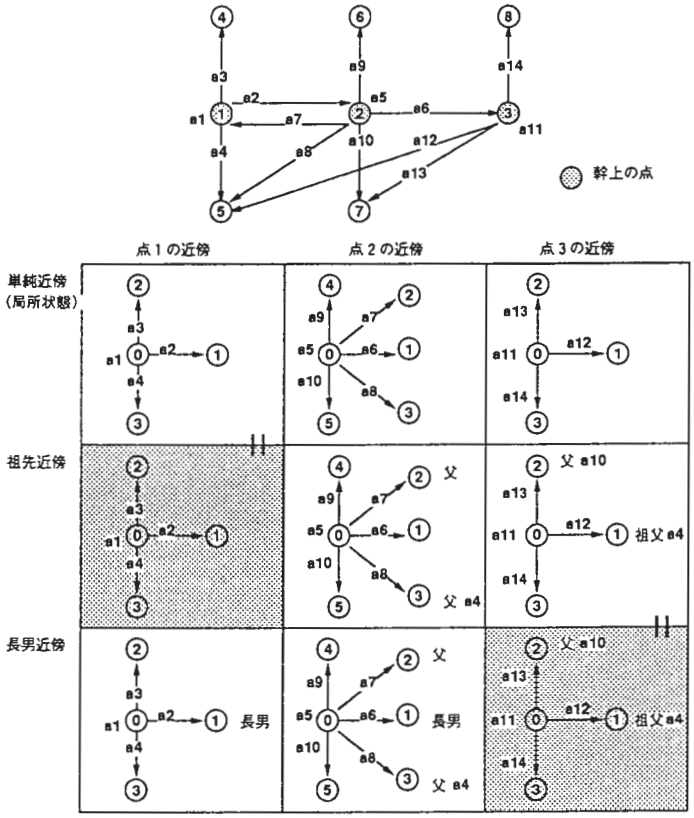

図 11: グラフ中の点近傍

6.2 分類木の生成

ＳＴＲ規則を規則中のイベント、幹中の点の単純近傍、祖先近傍、長男近傍をキーとして分類し、規則の分類木を作成する。分類木中のノードには、対応するＳＴＲ規則の集合及び分類キーとなったイベントまたは近傍が含まれる。分類木の葉には、分類された結果として１個の規則だけが含まれる。分類木は、文字列の集合中から特定の文字列を探索するために用いられるデータ構造 trie を拡張したものになっている。これは、ＳＴＲ規則の始グラフは幹という特徴を持つため可能になっている。図12に分類木の概略図を示す。

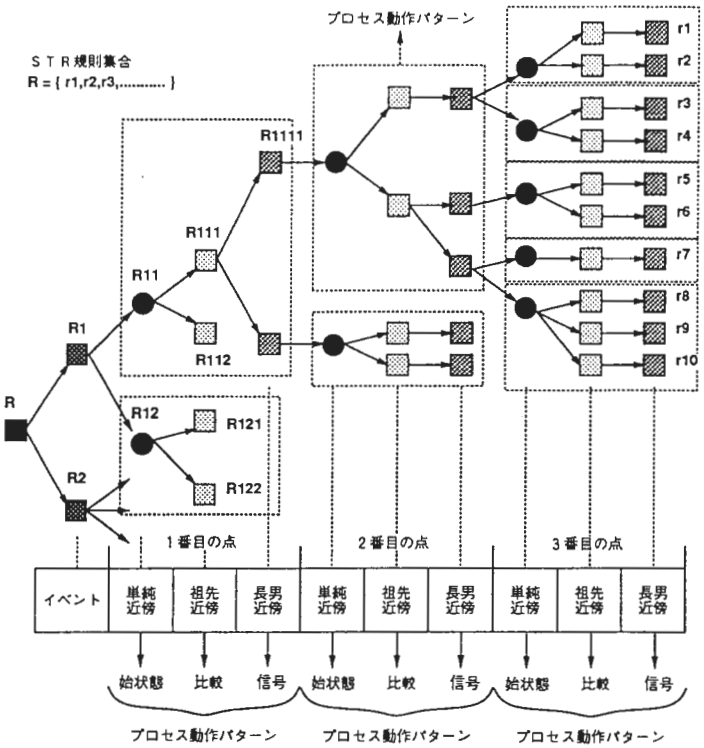

図 12: 分類木

6.3 プロセス動作パターンの生成

図12のＳＴＲ規則の分類木から、図10に示されているようなプロセス動作パターンを生成する。

分類木において、第i番目の点の近傍に関するノードすべてで構成される部分木に対して、１個プロセス動作を生成する。分類木を縦型に全探索することによって、順次プロセス動作を生成していく。

始状態、次状態の生成 分類木において、単純近傍に対応するノードからプロセスの状態を取り出す。

ＰＩＤ比較動作の生成 分類木中の祖先近傍に対応しているノードから祖先近傍を取り出す。取り出した祖先近傍中の祖先情報が付随している辺に対して、ＰＩＤ比較動作を生成する。

たとえば、ラベルa8を持つ辺が祖先情報として（父、a4）を持つ場合（図11の点２の祖先近傍）、a8という関係で接続しているプロセスのＩＤと、受信メッセージ中で親プロセスがa4という関係で接続しているプロセスのＩＤを比較する。

送信メッセージの生成 分類木中の長男近傍に対応するノードに対して、探索依頼メッセージを１個生成する。メッセージ中には、状態中のＰＩＤと受信した探索依頼メッセージ中のＰＩＤを含める。

生成されたメッセージは、このノードが保持しているＳＴＲ規則集合を暗に意味している。

ＰＩＤ再設定動作の生成 近傍抽出時の情報を利用して、ＰＩＤの再設定動作を生成する。

図13にＳＴＲ規則グラフから単純近傍（プロセスの状態）を抽出した例を示す。図のように、近傍抽出時に近傍中の各辺の終点と規則グラフ中の点の対応リストを作成しておく。状態中のＰＩＤは近傍中の辺の終点と対応しているので、このリストを用いて、状態中のＰＩＤと規則グラフ中の点との対応を確かめることができる。

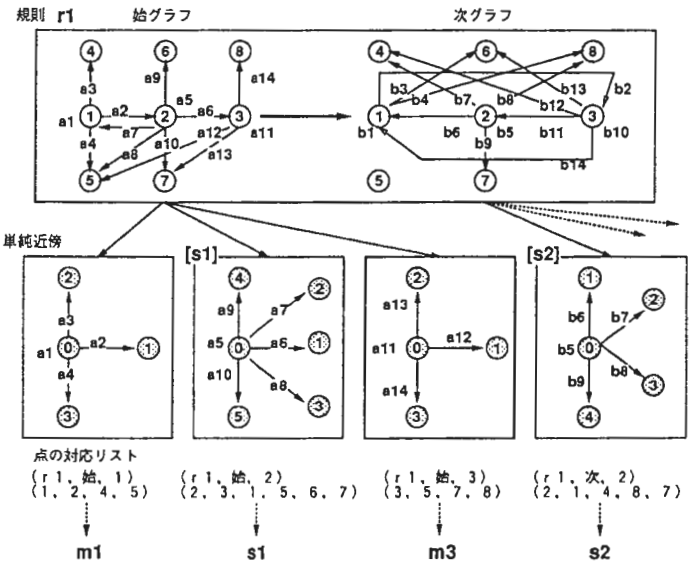

図 13: 近傍の抽出

図14に図13中の規則ｒ１を適用して、図9中のＰ２がｓ１からｓ２へ状態遷移する場合のＰＩＤ設定例を示す。

図13中の対応リストを用いて、状態ｓ１、ｓ２、メッセージｍ１、ｍ３中のＰＩＤと規則グラフ中の点の対応を知ることができる。図14は、これらのリストからｓ２中のＰＩＤをｓ１、ｍ１、ｍ３中から探す方法を例示している。

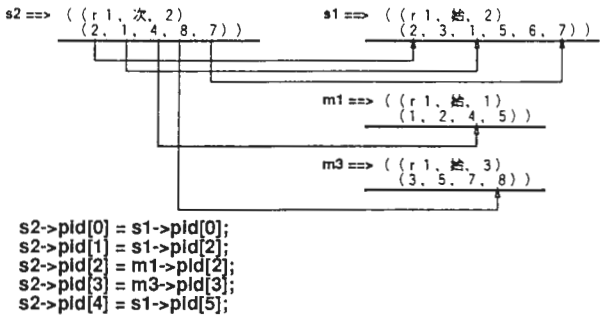

図 14: ＰＩＤ設定例

6.4　プロセス動作パターンの合成

生成されたプロセス動作パターンを用いて、アイドル状態から到達可能なすべての状態を生成し、各状態におけるプロセス動作を生成する。

（１）アイドル状態の展開。

ステップ２で生成されたプロセス動作パターンの内、idle を始状態として持つパターンを集め、アイドル状態時の動作を合成する。

（２）次状態の展開。

（１）で生成されたプロセス動作のすべての次状態ｓに対して、ステップ２で生成されたプロセス動作パターンの内、ｓに含まれる状態を始状態として持つものを集め、そのパターンと（１）で生成された動作パターンとを合成する。

図 15に次状態の展開を例示する。図中のパターン１は、アイドル状態から展開が進んで、状態ｓ１を始状態とするパターンが張りついているところである。図中のパターン２は、始状態ｓ３がｓ１に含まれ、パターン１と同じメッセージｍ１を受信した場合のプロセス動作パターンである。パターン１中には、拒否メッセージを受信して始状態に戻る動作が必ず含まれている。パターン１中の拒否メッセージ受信後の動作を、パターン２中のｍ１受信後の動作に置き換える。この時付加されるパターン２の次状態ｓ４をｓ５に書き換える。ｓ５はｓ４にｓ１とｓ３の差分（ｓ１－ｓ３）を付加した状態である。こうして、新しい状態が生成される。

合成されたパターン３を（１）にもどして、（２）の次状態を展開を新しい状態が生成されなくなるまで繰り返す。

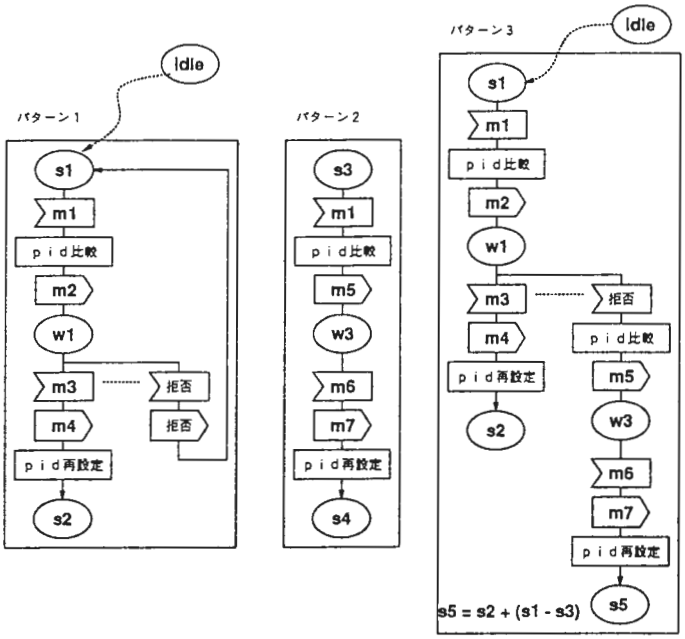

図 15: プロセス動作パターンの合成

6.5 禁止状態指定規則の適用

プロセス動作パターンの合成時に、禁止状態指定規則で指定された禁止されたプロセスの状態が生成される場合がある。

図 16において、(1),(2),(3) の 3 個のプロセス動作パターンを合成することを考える。(1) は状態 s1 から規則 r1 を適用して状態 s2 に遷移する動作であり、(2) は、状態 s3 から規則 r2 を適用して s4 に遷移するか、規則 r3 を適用して s4 に遷移する動作を表している。また、(3) は (2) から信号 m3 を状態 s6 で受信した場合の動作を表している。

s3 が s1 に含まれる場合 s1 において s3 と同様な動作が可能であるため、(1) と (2) を合成することができる。その合成結果を (4) とすると、(4) においてＳＴＲ中には陽に記述されていなかった状態 (s4+(s1-s3)) と (s5+(s1-s3)) が現れる。これらの状態は、それぞれ状態 s4 と s5 に s1 と s3 の状態記述プリミティブの差分を加えたものであり、プリミティブが増加したために、禁止状態指定規則に抵触する場合がある。

今、(s4+(s1-s3)) が禁止状態になったとする。(4) において、単純に (s4+(s1-s3)) への遷移を取り除くと、(7) に示すようにプロセスＢで規則 r2 を適用しプロセスＡでは何もしないという状況がおこりえる。これは異常である。このような動作の生成を防ぐために、(5) に示すように信号 m3 の代用をする新しい信号 m4 を生成する。信号 m3 は、規則 r2 と r3 の適用が可能であることを示す信号であったが、信号 m4 は規則 r3 のみが適用可能である。そのため、信号 m4 を受信したプロセスは決して規則 r2 を適用しない (6)。こうして、ＳＴ

Rの広域状態遷移規則と禁止状態指定規則の整合をとることができる。

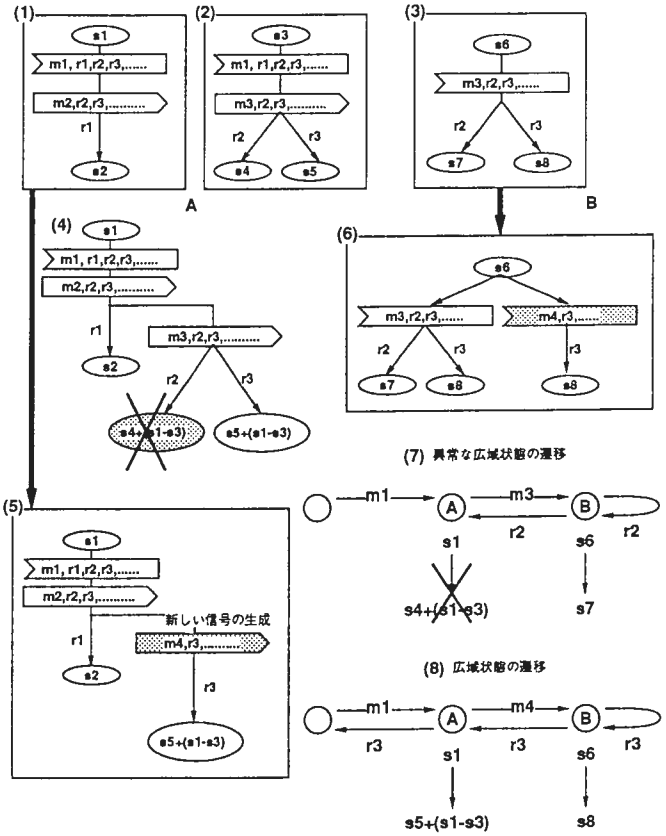

図 16: 状態生成の抑制

7　クラス４ＳＴＲのプロセス仕様

クラス１またはクラス２ＳＴＲでは、各プロセスは応答メッセージを受信した時点で適用できる規則を決定できるが、クラス３またはクラス４ＳＴＲではイベント生起プロセスが周辺の情報を確認するまで、適用できる規則を決定できない。そのため、以下の３種類のメッセージを用意する。

7.1　メッセージ

プロセス間通信に用いられるメッセージについて説明する。

探索依頼メッセージ

探索依頼メッセージは、周辺の広域状態を報告するように求めるメッセージである。探索依頼メッセージ中には、メッセージＩＤ(mid)が含まれる。このmidによって、探索すべき規則グラフの部分グラフの集合が指定される。探索依頼メッセージreqが指定するグラフ集合をG(req)と表記する。

応答メッセージ

探索依頼メッセージに対する応答メッセージである。応答メッセージ中には、メッセージＩＤと関連する周辺プロセスのＰＩＤ列が含まれる。

応答メッセージresは、発見された同形な部分グラフの集合G(res)とプロセスＩＤの集合属を含意する。

$$G(res) = \{g_1, ...g_k\},\ \ P(res) = \{P_1, ...P_k\},\ \ P_i = \{pid_{i0}, pid_{i1}, ...pid_{il}\}$$

ここでg_iは規則グラフの部分グラフである。P_iはg_iとシステムグラフの間の同形写像h_iを与える。

$$h_i(v_j) = pid_{ij},\quad v_j \in V(g_i)$$

状態遷移指示メッセージ

状態遷移指示メッセージ中には、メッセージＩＤが含まれる。状態遷移指示メッセージはＳＴＲ規則と規則グラフ中の点を含意する。このメッセージを受けとったプロセスは、指定された規則の指定された点の次状態に状態を遷移させる。

7.2 プロセス仕様の概略

プロセスＰ１が状態sの時イベントevが生起した場合、探索すべきグラフ集合$G(s, ev)$が決まる。また各隣接プロセスＰ２に対して、探索を依頼すべき部分グラフ集合$G(pid, s, ev)$が決まる。Ｐ１は各Ｐ２に対して、$G(p2, s, ev)$を含意する探索依頼メッセージreqを送信する。

Ｐ２からの応答めっせーじresは$G(res) \subset G(pid, s, ev)$を含意している。すべての応答メッセージを受信した後、同形なグラフの集合$G'(s, ev) \subset G(s, ev)$が決定される。そして、$G'(s, ev)$から任意に規則グラフを１個選択し、その規則を適用するために関連するプロセスに対して状態遷移指示メッセージを送信する。

プロセスＰ２が状態s'の時メッセージreqを受信した場合、探索すべきグラフ集合$G(s', req)$が求まる。また各隣接プロセスＰ３に対して、探索を依頼すべき部分グラフ集合$G(pid, s', req)$が決まる。Ｐ２はＰ３に対して$G(pid, s', req)$を含意する探索依頼メッセージreq'を送信する。

Ｐ３からの応答メッセージres' は $G(res') \subset G(pid, s', req)$ を含意している。すべての応答メッセージを受信した後、同形な部分グラフの集合 $G'(s', req) \subset G(s', req)$ が決定される。Ｐ３はＰ２に対して $G'(s', req)$ を含意する応答メッセージresを送信する。

プロセスは状態遷移指示メッセージを受信した場合、そのメッセージに従って状態遷移を実行する。

図17に、プロセス間通信によって適用すべきＳＴＲ規則が決定される様子を図示する。また、図18にプロセスの動作概要を図示する。

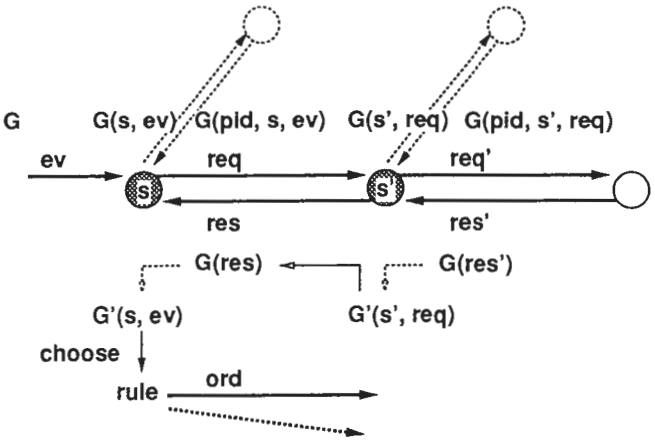

図 17: クラス４プロセス間通信

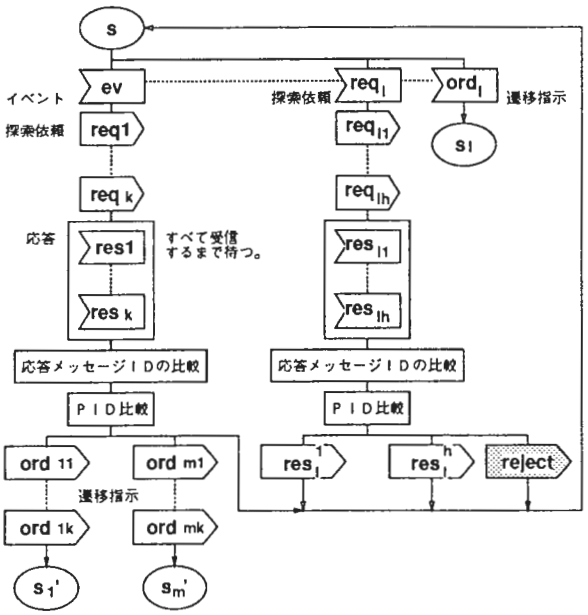

図 18: クラス４プロセスの動作概要

8 規則グラフの解析

プロセスは適用すべきＳＴＲ規則を決定するために、規則グラフの部分グラフを分担して探索する。本章では、各プロセスが探索すべき部分グラフの分担を決めるために、規則グラフの解析を行なう。

8.1 用語の定義

グラフを解析するために使用する用語を定義する。

(1) $g = (V, E, v^0)$: 根付きラベル付き有向グラフ。g の根 v^0 から g 中の任意の点へのパスが存在する。

(2) $spt(g)$: グラフ g の極大木。

(3) $st(v, t)$: 木 t の v を根とする部分木。

(4) $sg(v, g)$: g の部分グラフ st(v, spt(g)) から誘導される g の部分グラフ。

(5) $N(v)$: 点 v の近傍。

(6) $\Gamma(v)$: 点 v の隣接点の集合 $\{w|(v, w) \in E\}$ を表す。

(7) $g_1 \sqsubset g_2$: グラフ g_1 はグラフ g_2 の部分グラフであることを表す。

8.2 規則グラフの分解

規則グラフを分解して規則グラフの分解木を作成する。

分解木

ＳＴＲ規則の始グラフを g、g の極大木を spt(g) とする。g は次のように部分グラフの和に分解できる。

$$g = N(v_0) \cup sg(v_1, g) \cup ...sg(v_m, g)$$

ここで、v_0 は g の根であり、v_1,...v_m は v_0 の子である。また、$sg(v_i, g)$ は spt(g) の部分木 $st(v_i, spt(g))$ から誘導される g の部分グラフである。各 $sg(v_i, g)$ もまた同様にその部分グラフの和に分解される。

g の分解木 rt(g) は、以下の 3 種類のノードから構成される。

(1) イベントノード

イベントノードは分解木の根であり、規則のイベントを含む。イベントノードは子として近傍ノードを1個だけ持つ。

(2) 辺ノード

辺ノード edv は、極大木 spt(g) の辺 edge に対応し、以下の2種類の要素を含む。

$$edv = (LABEL(edv), sg(edv))$$

LABEL(edv) は対応する辺上のラベル集合であり、sg(edv) は edge の終点を根とする g の誘導部分グラフである。辺ノードの子は近傍ノードである。

(3) 近傍ノード

近傍ノード nv は g 中の点 v に対応し、以下の要素から構成される。近傍ノードの子は辺ノードである。

$$nv = (N(nv), sg(nv), F(nv))$$

ここで N(nv)=(V_0, E_0) は v の近傍であり、sg(nv)=(V, E) は g の誘導部分グラフ (sg(v, g)) であり、F(nv) は以下に示されるような V 上の写像の集合である。

点集合間の写像

$$\begin{aligned} F(nv) &= \{f(nv), f(edv_1), ...f(edv_k)\} \\ f(edv_i) &: V \longrightarrow (V_i \cup \{\lambda\}) \\ f(edv_i)(v) &= \begin{cases} v & (\text{if } v \in V_i), \\ \lambda & (\text{otherwise}) \end{cases} \end{aligned}$$

ここで edv_i は nv の子ノードである。edv_i に含まれるグラフ $sg(edv_i) = (V_i, E_i)$ は sg(v, g) の部分グラフである。

すべてのＳＴＲ規則に対して分解木を生成する。ＧをＳＴＲ規則の始グラフの集合とすると、分解木生成のために用いる極大木は以下の性質を満たす。

$$\forall g_i, g_j \in G,\ (g_i \sqsubset g_j \Longrightarrow spt(g_i) \sqsubset spt(g_j))$$

8.3 グラフの合成

すべてのＳＴＲ規則の分解木を合成して、合成木 SYT(G) を生成する。合成木は以下に示すように4種類のノードから構成される。

(1) 根ノード

根ノードはＳＴＲ規則の始グラフ全体からなる集合を含む。根ノードの子はイベントノードである。

(2) イベントノード

イベントノードevは、イベントと対応する規則グラフの集合G(ev)の2個の要素を含む。イベントノードの子は近傍ノードである。

$$ev = (event(ev), G(ev))$$

(3) 辺ノード

辺ノードedvは、辺上のラベルの集合と対応する部分グラフの集合を含む。辺ノードの子は近傍ノードである。

$$edv = (LABEL(edv), G(edv))$$

(4) 近傍ノード

近傍ノードnvは点近傍N(nv)と対応する部分グラフの集合G(nv)を含む。G(nv)中の各グラフには、分解木の場合と同様に点集合間の写像の集合が付随している。nvの構造は次のように表現できる。

$$\begin{aligned} nv &= (N(nv), G(nv)) \\ G(nv) &= \{SG_1(nv), ...SG_k(nv)\} \\ SG_i(nv) &= (sg_i(nv), F_i(nv)) \end{aligned}$$

ここでG(nv)中の各根付きグラフ$sg_i(nv) = (V_i, E_i)$の根の近傍は、nv中の近傍$N(nv) = (V_{i0}, E_{i0})$に等しい。$F_i(nv)$中の写像は、nvの子ノードに対応する。

$$F_i(nv) = \{f_i(nv), f_i(edv_1), ...f_i(edv_k)\}$$

ここでedv_jはnvの子ノードである。edv_j内のグラフ集合$G(edv_j)$中には、$sg_i(nv)$の部分グラフと同形なグラフ$sg_{ij} = (V_{ij}, E_{ij})$が存在する。写像$f_i(edv_j)$は以下のように定義される。

$$\begin{aligned} &f_i(edv_j) : V_i \longrightarrow (V_{ij} \cup \{\lambda\}) \\ &f_i(edv_j)(v_i) = \begin{cases} v_{ij} & \text{(if there is a corresponding vertex in } V_{ij}\text{)}, \\ \lambda & \text{(otherwise)} \end{cases} \end{aligned}$$

edv_j中のグラフsg_{ij}は$sg_i(nv)$とedv_jから決定できる。そのため、sg_{ij}を$sg(sg_i(nv), edv_j)$と表記することにする。図19に合成木を例示する。

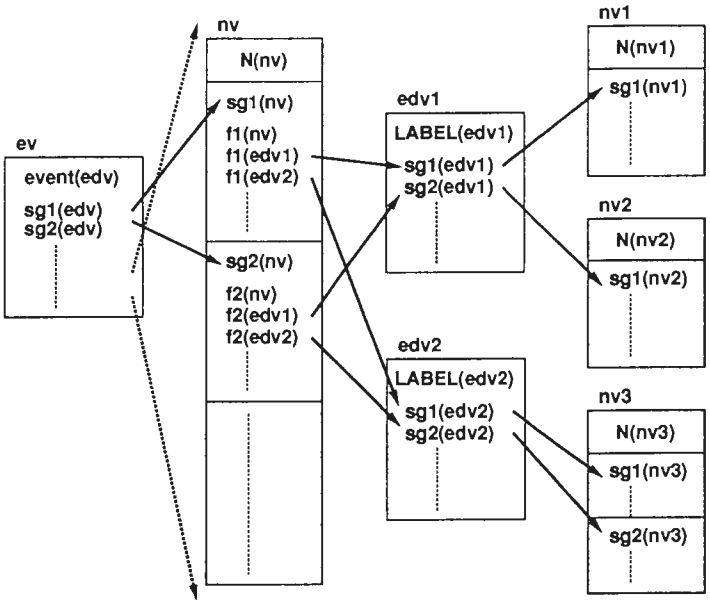

図 19: 合成木

8.4 合成木の生成

ＳＴＲ規則の始グラフ全体の集合をＧとし、$G_k = \{g_1, ...g_k\}$ をＧの部分集合とする。合成木SYT(G)は以下に示すように、帰納的に生成できる。

(I) $SYT(G_1)$

$SYT(G_1)$ は根ノードに分解木 $rt(g_1)$ を子として付加することによって生成される。

(II) $SYT(G_{k+1})$

$SYT(G_{k+1})$ は $SYT(G_k)$ と $rt(g_{k+1})$ から以下のように生成できる。ここで $G_{k+1} = G_k \cup \{g_{k+1}\}$ である。

ev、nv、edv を、それぞれ $SYT(G_k)$ 中のイベントノード、近傍ノード、辺ノードとする。また、ev_{k+1}、nv_{k+1}、edv_{k+1} を、それぞれ $rt(g_{k+1})$ 中のイベントノード、近傍ノード、辺ノードとする。

(1) イベントノードの合成

ev_{k+1} と ev 中のイベントを比較する。

もし、ev と ev_{k+1} が共通するイベントを含むならば、g_{k+1} を $G(ev)$ に追加する。そして、各 $nv_{k+1} \in \Gamma(ev_{k+1})$ と $nv \in \Gamma(ev)$ に対して、下記の (2) に示す方法で $st(nv_{k+1}, rt(g_{k+1}))$

と $st(nv, SYT(G_k))$ を合成する。

もし、ev と ev_{k+1} が共通するイベントを含まないならば、$rt(g_{k+1})$ を $root(G_k)$ に子として付加する。

(2) 近傍ノードの合成

nv_{k+1} と nv 中の近傍を比較する。

もし、$N(nv)$ と $N(nv_{k+1})$ が等しいならば、nv_{k+1} と nv を (2-1) に示すように合成する。

もし、$\Gamma(ev)$ 中の任意のノードが $N(nv_{k+1})$ と等しい近傍を含まないならば、$st(nv_{k+1}, rt(g_{k+1}))$ を ev に子として付加する。

(2-1) G(nv) の合成

$sg(nv)$ を $G(nv)$ の要素とする。$sg(nv_{k+1})$ と $sg(nv)$ を比較する。

もし、$sg(nv)$ が $sg(nv_{k+1})$ と同形ならば、$st(nv_{k+1}, rt(g_{k+1}))$ はすでに $st(nv, SYT(G_k))$ に部分木として含まれている。そのため、$st(nv, SYT(G_k))$ はこれ以上変更されない。

もし、$sg(nv)$ が $sg(nv_{k+1})$ と同形でないならば、$sg(nv_{k+1})$ を $G(nv)$ に追加する。$N(nv)$ は $N(nv_{k+1})$ と等しいので、$rt(g_{k+1}))$ 中の各イベントノード $edv_{k+1} \in \Gamma(nv_{k+1})$ に対して、$SYT(G_k))$ 中のイベントノード $edv \in \Gamma(nv)$ で $LABEL(edv) = LABEL(edv_{k+1})$ を満たすものが存在する。これらのイベントノードについて、$st(edv_{k+1}, rt(g_{k+1}))$ と $st(edv, SYT(G_k))$ を (3) に示すように合成する。

(3) 辺ノードの合成

$sg(edv)$ を $G(edv)$ の要素とする。$sg(edv_{k+1})$ と $sg(edv)$ を比較する。

もし、$sg(edv)$ が $sg(edv_{k+1})$ と同形ならば、$st(edv_{k+1}, rt(g_{k+1}))$ はすでに $st(edv, SYT(G_k))$ に部分木として含まれている。そのため、$st(edv, SYT(G_k))$ はこれ以上変更されない。

もし、$G(edv)$ 中のすべてのグラフが $sg(edv_{k+1})$ と同形でないならば、$sg(edv_{k+1})$ を $G(edv)$ に子として付加する。$rt(g_{k+1}))$ 中の各近傍ノード $nv_{k+1} \in \Gamma(edv_{k+1})$ と、$SYT(G_k))$ 中の各近傍ノード $nv \in \Gamma(edv)$ について、上記 (2) の手順に従ってそれらの部分木どうしを合成する。

9 クラス４ＳＴＲからＳＤＬを生成するアルゴリズム

プロセスが状態 s でメッセージ m を受信した後、次の安定状態に遷移するまでのふるまいを基本プロセス動作と呼び $bpb(s, m)$ と表記することにする。プロセス仕様は基本プロセ

ス動作の集合で表現できる。

プロセス仕様を生成するために、まずプロセスの状態とプロセス間通信に用いられるメッセージをすべて生成する。その後ですべての基本プロセス動作を生成する。

なお本稿では、プロセスは同一のラベルを一定個数（maxlabel）以上持たないと仮定する。

9.1 状態の生成

s をプロセスの状態、r をＳＴＲ規則、v を規則グラフ中の点とする。また、N(r, v) を v の始グラフ中の近傍、N'(r, v) を v の次グラフ中の近傍とする。

もし $N(r,v) \sqsubset s$ ならば、s 中の N(r, v) を N'(r, v) に置き換えたものを s' とする。もし s' が同一のラベルを maxlabel 以上持たないならば、s に対してＳＴＲ規則を適用できる可能性があり、この時 s' を next(s, r, v) と表記する。

プロセスの状態集合 S は以下のように再帰的に生成される。

$$(I) \quad S_0 = \{N(r,v) | r \in R, v \in V(r)\}$$
$$(II) \quad S_{i+1} = S_i \cup \{next(s,r,v) | s \in S_i, r \in R, v \in V(r)\}$$

ここで、V(r) はＳＴＲ規則 r の規則グラフ中の点集合である。S は有限集合になる。

9.2 合成木の修正

すべてのメッセージを生成する準備として、合成木を修正して規則グラフ間の包含関係を反映した構造にする。

SYT(G) 中のイベントノードまたは辺ノードを node、$\Gamma(node)$ 中のノード内の近傍全体の集合を S(node) とする。また、s を S(node) 中の状態、S(node, s) と NV(node, s) をそれぞれ以下のような集合とする。

$$S(node,s) = \{state | state \sqsubset s, state \in S(node)\}$$
$$NV(node,s) = \{nv \in \Gamma(node) | N(nv) \in S(node,s)\}$$

(1) もし S(node) が s を含むならば、$\Gamma(node)$ 中のノード nv(s) で、N(nv(s)) = s を満たすものが存在する。もし S(node) が s を含まないならば、新しく近傍ノード nv(s) を node の子として以下を満たすように作成する。

$$N(nv(s)) = s, \quad G(nv(s)) = \phi$$

(2) NV(node, s) 中の各ノード nv について、G(nv) を順に G(nv(s)) に付加し、$st(nv, SYT(G))$ と $st(nv(s), SYT(G))$ の合成を、前セクションで記述した G(nv) の合成方法を用いて実行する。

9.3 メッセージの生成

状態遷移指示メッセージは、規則グラフ中の各点に対して 1 個生成される。

合成木の辺ノード edv に対して探索依頼メッセージ req(edv) が生成される。req(edv) はグラフ集合 G(edv) を含意する。

合成木の近傍ノード nv に対して、部分集合属 GSET(nv) を以下のように構成する。

$$GSET(nv) = \{G_i | G_i \subset G(nv),\ (\forall g_j, g_k \in G_i, \neg(g_j \sqsubset g_k))\}$$

GSET(nv) の各要素 G_i に対して、応答メッセージ res(nv, G_i) が生成される。

9.4 プロセス仕様の生成

基本プロセス動作 bpb(s, m) は、次の 4 つの要素から構成されている。

(1) 探索依頼メッセージの送信　(2) 応答メッセージの受信

(3) 部分グラフ集合の決定　　(4) 応答メッセージの送信

node を SYT(G) 中のメッセージ m に対応するイベントノードまたは辺ノードとする。また、nv を node の子ノードで近傍 s を含む近傍ノードとする。bpb(s, m) は、以下のように生成される。

(1) 探索依頼メッセージの送信

edv_i を nv の子である辺ノードとする。edv_i に対して、隣接プロセスへの探索依頼メッセージ $req(edv_i)$ が生成される。メッセージ $req(edv_i)$ の送信先プロセスは、edv_i 中のラベルと同一のラベルで連結しているプロセスである。

(2) 応答メッセージの受信

$req(edv_i)$ に対する応答メッセージ $res(edv_i)$ は、edv_i の子ノードの集合から生成される。$res(edv_i)$ の含意するグラフ集合は、次の条件を満たす。

$$G(res(edv_i)) \in \bigcup_{nv_j \in \Gamma(edv_i)} GSET(nv_j)$$

(3) 部分グラフ集合の決定

(3-1) 応答メッセージ I D の比較

すべての $\Gamma(nv)$ の要素 edv_i と $G(nv)$ の要素 $sg_j = (V_j, E_j)$ について、応答メッセージの含意するグラフ集合 $G(res(edv_i))$ が $sg(sg_j(nv), edv_i) = (V_{ji}, E_{ji})$ を含んでいるかどうか調べる。そして、以下のグラフ集合 G_1 を作成する。

$$G_1 = \{sg_j \in G(nv) | \forall edv_i \in \Gamma(nv),\ sg(sg_j(nv), edv_i) \in G(res(edv_i))\}$$

(3-2) ＰＩＤの比較

$res(edv_i)$ 中のＰＩＤ集合で $sg(sg_j(nv), edv_i)$ 中の点集合に対応するものを P_{ji} とする。$PIDS$ をシステム中のＰＩＤ集合とすると、P_{ji} は写像 $h_{ji} : V_{ji} \longrightarrow PIDS$ を与える。近傍ノード nv は写像 $f_j(edv_i) : V_j \longrightarrow V_{ji}$ を含んでいる。h_{ji} と $f_j(edv_i)$ を用いて写像 $\bar{h_{ji}}$ を以下のように定義する。

$$\begin{aligned} &\bar{h_{ji}} : V_j \longrightarrow (PIDS \cup \{\lambda\}) \\ &\bar{h_{ji}}(v) = \begin{cases} h_{ji} \cdot f_j(edv_i)(v) & (\text{if } f_j(edv_i)(v) \neq \lambda), \\ \lambda & (\text{otherwise}) \end{cases} \end{aligned}$$

G_1 中のすべてのグラフ sg_j 中のすべての点 v と、$\Gamma(nv)$ 中のすべてのノードのペア edv_i、edv_k について、$\bar{h_{ji}}(v)$ と $\bar{h_{jk}}(v)$ が等しいかどうか調べる。そして、以下のグラフ集合 G_2 を作成する。

$$G_2 = \{sg_j \in G_1 | \forall v \in V_j,\ \forall edv_i, edv_k \in \Gamma(nv),\ \bar{h_{ji}}(v) = \bar{h_{jk}}(v)\}$$

(4) 応答メッセージの送信

G_2 中のグラフ sg_j において、各点 v に対して $\bar{h_{ji}}(v) \neq \lambda$ を満たす写像 $\bar{h_{ji}}$ が存在する。このことを利用して、写像 $h_j : V_j \longrightarrow PIDS$ を次のように定義する。

$$h_j(v) = \bar{h_{ji}}(v) \neq \lambda$$

h_j をもちいて以下の性質を満足する応答メッセージ $res(nv)$ が生成される。

$$\begin{aligned} &G(res(nv)) = G_2, \quad P(res(nv)) = \{P_1, P_2, ...P_k\}, \\ &P_j = \{h_j(v) | v \in V_j = V(sg_j),\ sg_j \in G_2\} \end{aligned}$$

10　生成されるプロセス仕様の効率

本稿では、２種類のプロセス仕様生成アルゴリズムを与えた。ここでは、これらのアルゴリズムを用いて生成されるプロセス仕様の効率について述べる。

クラス２ＳＴＲ記述から生成されるプロセス仕様の効率

各プロセスは応答メッセージを受信した時点で、適用すべき規則を決定でき状態遷移を実行できる。そのため状態遷移指示メッセージは不要になる。

しかし、隣接プロセスに対して１回ずつ探索依頼メッセージを送信するので、隣接プロセスから拒否メッセージが返信された場合、別のプロセスに探索依頼メッセージを送信しなおさなければならない。そのため、最悪の場合分類木中の長男近傍ノードの個数だけ探索依頼メッセージを送信してしまうが、通常こういうことはあり得ない。メッセージ送信のために必要な時間は、平均して分類木の深さに比例する。

クラス４ＳＴＲ記述から生成されるプロセス仕様の効率

プロセスは複数の隣接プロセスに対して並列に探索依頼メッセージを送信しているので、合成木の深さをｄとすると、メッセージ送信のために必要な時間は、最悪の場合でも 2d+1 で済む。

しかし、送信されるメッセージの総量は合成木中のノード数に比例する。また、多数の応答メッセージを用意する必要がある。応答メッセージ受信後の応答メッセージＩＤの比較のために要する時間は、$|\Gamma(nv)||G(nv)|^2$ に比例し、ＰＩＤ比較のために要する時間は、$|\Gamma(nv)||V_j||G(nv)|$ に比例する。このため、クラス２ＳＴＲに比べて多くの比較操作が必要である。

11　ＳＴＲカスタマイズ言語

ＳＴＲカスタマイズ言語は、通信システム開発者がＳＴＲ記述を補完するために記述する言語である。この言語記述によってネットワークの差異を吸収することができる。

ＳＴＲカスタマイズ言語は、ＳＴＲから生成されるＳＤＬにタスク挿入を指定する規則の集まりで構成される。これら規則は、ＳＤＬがグラフ表現可能なことに基づいて、ＳＤＬ中の位置指定とタスク指定のペアで与えられる。

位置指定　{タスク指定}

11.1　ＳＤＬ中の位置指定

ＳＴＲから生成されるＳＤＬは、構成要素としてプロセスの状態とプロセスの入力メッセージを含んでいる。ＳＤＬ中の位置を指定するために、この状態と入力信号を使用する。これらのＳＤＬ中に現れる状態及びメッセージは、ＳＴＲ記述中のＳＴＲ規則、状態プリミ

ティブ、イベントに依存して生成されるものである。従って、サービス提供者は与えられたＳＴＲ記述をもとに、ＳＴＲカスタマイズ言語を記述できる。

位置指定の方法には、次の６種類がある。

(1) 始点指定、(2) 終点指定、(3) 状態指定、(4) 入力指定、(5) 状態入力指定、(6) 状態遷移指定

(1) 始点指定は、プロセスの初期化動作を規定し、(2) 終点指定はプロセスの終了動作を規定するものである。

(3) 状態指定はＳＤＬ中に記述されているプロセスの状態を指定する。状態によって位置指定された場合、タスクは指定された状態に遷移する直前に挿入される。

(4) 入力指定は、ＳＤＬ中に記述されているプロセスのメッセージ入力を指定する。入力メッセージによって位置指定された場合、タスクはそのメッセージを受信した直後の動作として挿入される。

(5) 状態入力指定は、特定の状態において特定のメッセージを受信した部分を指定する。この場合も (4) と同様に、指定されたメッセージを受信した直後の動作としてタスクを挿入する。

(6) 状態遷移指定は、始状態と次状態を指定し、指定された始状態から指定された次状態へ遷移する直前の動作としてタスクを挿入する。

状態による位置指定

ＳＤＬ中の状態を１個特定して指定する場合と、ある条件を満たす状態すべてを指定する場合がある。ＳＤＬ中の状態は主状態プリミティブを用いて記述されているので、位置指定は以下のいずれかの表現をとる。

(1)*state* (ＳＴＲ中の状態プリミティブの列)

(2)*primitive* (ＳＴＲ中の状態プリミティブの列)

(1) は、指定されたプリミティブ列だけで表現される状態を１個指定する。(2) は、指定されたプリミティブ列を含む状態をすべて指定する。

入力メッセージによる位置指定

入力メッセージによる位置指定は、以下のように表現される。

(1)*event* (イベント指定)
(2)*request* (通知メッセージ指定)
(3)*response* (応答メッセージ指定)
(4)*norule*

(1) のイベント指定では、ＳＴＲ記述に用いられたイベントをそのまま書く。イベント指定は省略可能であり、その場合すべてのイベントを指定することになる。

(2) 通知メッセージは、対応しているＳＴＲの広域状態遷移規則の集合で指定する。通知メッセージ指定を省略した場合は、すべての通知メッセージを表す。

(3) 応答メッセージは対応している広域状態遷移規則で指定する。省略された場合は、すべての応答メッセージを指す。

特別な応答メッセージとして、適用できる規則が存在しないことを表すメッセージが存在する。(4)norule は、そのメッセージを指定するものである。

状態遷移指定

状態遷移による位置指定は、始状態と次状態についての状態中のプリミティブの差分を指定するすることによって、次のように記述する。

transition (<状態遷移指定式> *)

ここで、状態遷移指定式は以下の (1) または (2) の形態で表記する。

(1) + (ＳＴＲ中の状態プリミティブの列)
(2) − (ＳＴＲ中の状態プリミティブの列)

(1) は、次状態中のプリミティブで始状態中に存在しないものを表記する記法であり、(2) は始状態中のプリミティブで次状態中に存在しないものを表記する記法である。1 個の状態遷移指定記述によって、複数の状態のペアが指定される。以下の記述例は、 ringback を含む状態から path を含む状態への遷移を指定する。

transition (+(path) −(ringback))

11.2　タスク指定

タスク指定は、Ｃ言語の文、または条件分岐文で表現される。条件分岐文は、次の形をしている。

$$unless(\text{Ｃ言語の式}) < \text{Ｃ言語の文} >< \text{状態指定} >;$$

この条件分岐文において、式が成立しない場合の遷移先を指定する。

例１ 端末制御タスクの挿入

図20で示されるＳＴＲ規則があるものとする。このＳＴＲ規則からは、図21に示されるＳＤＬが生成される。

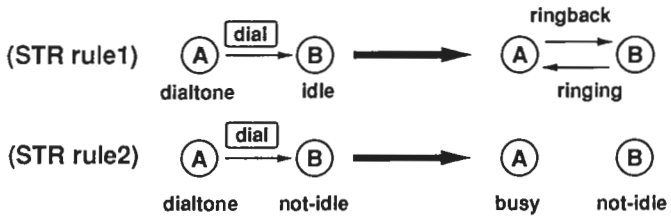

図 20: ＳＴＲ規則

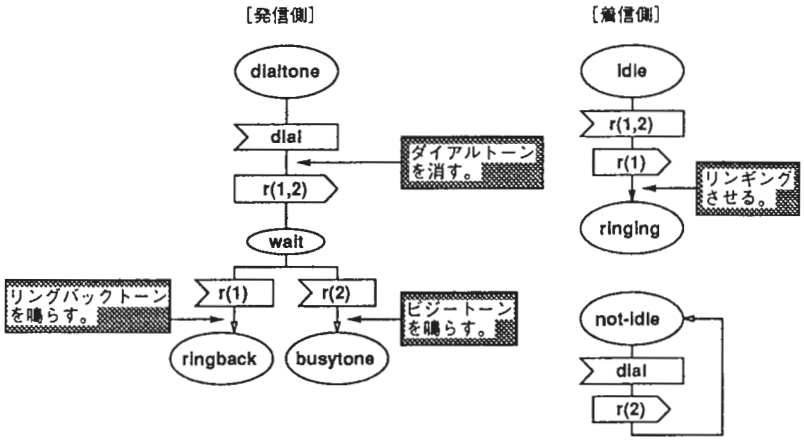

図 21: 端末制御タスクの挿入

図22に示す規則を書くことによって、図21のようにＳＤＬ中にタスクが挿入される。

```
event (dial) { stop_dialtone(); }                /* rule1 */
state (ringback) { start_ringbacktone(); }       /* rule2 */
state (busytone) { start_busytone(); }           /* rule3 */
```

図 22: 端末制御規則

例２ 通信路の確保

通信路を確保するタスクを実行する場合、通信路を確保できなかった時ビジー状態に遷移する（図 24）。この動作は次の規則 4 によって表現できる。

```
input (dial) {              /* rule4 */
   stop_dialtone();
   status = reserve_path();
   unless (status) {
     start_busytone();
   } state (busytone);
}
```

図 23: 回線確保規則

12 今後の課題

今後の課題として、ＳＴＲコンパイラに関しては、生成されるプロセス仕様の最適化、及びデッドロックを回避する手法の構築が上げられる。ＳＴＲ／Ｄに関しては、より使いやすい言語になるように言語仕様を改良する必要がある。

12.1 最適化

以下に生成されるプロセス仕様に関して、最適化すべき項目を列挙する。

（メッセージ中の不要なＰＩＤ） メッセージ中には通過したプロセスの保持するＰＩＤがすべて含まれるが、大部分のＰＩＤを使用されない。使用されるＰＩＤのみ送信すればよい。

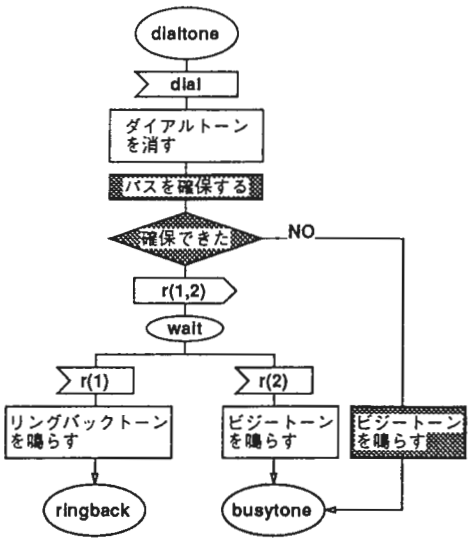

図 24: 通信路の確保

（ＰＩＤ比較の重複） 探索依頼メッセージを受信した後ＰＩＤを比較し、応答メッセージを受信した後もう一度ＰＩＤ比較を行なっている。ＰＩＤの比較は一度でよい。

（主状態に対するＰＩＤと副状態に対するＰＩＤの重複） 主状態に対するＰＩＤと副状態に対するＰＩＤを別々に管理している。１つのＰＩＤを主状態と副状態で重複して保持するのを避ける。

（応答メッセージ数を減らす） 複数規則に対して同一の次状態ならば、メッセージは１個ですむ。

（状態中の不要なＰＩＤ） プロセスは関係しているすべてのプロセスのＩＤを保持するが、使用されないＩＤもあり得る。そのようなＩＤを保持するのは無駄である。

（異種通信プロトコルの混合） クラス２ＳＴＲ記述から生成される通信プロトコルと、クラス４ＳＴＲ記述から生成される通信プロトコルは異なっている。これらを混合して最適なプロトコルを生成する。

13　おわりに

ＳＴＲからプロセス仕様を生成する２種類のアルゴリズムとＳＴＲ／Ｄ言語仕様について記述した。

ＳＴＲはプロダクションシステムの一種である。通常プロダクションシステムにおいて、システム中の様々な要素を表現するために変数が用いられる。しかし、ＳＴＲでは変数はプロセスを表現するためにしか用いられておらず、その意味でＳＴＲはプロダクションシステムのサブセットになっている。

本稿では、ＳＴＲ規則をグラフ表現したが、これはＳＴＲのこの特徴を生かしたものになっている。一般に、規則中の変数のユニフィケーションの実現方法は難しい。規則をグラフ表現することによって、いかに変数のユニフィケーションを実行するかという問題を、同形グラフの探索問題に置き換えている。問題を視覚化することによって、解決方法を見通すことが容易になっている。また、通信システムにおいて、複数のプロセスが連結しているかどうか明確にすることは本質的に重要である。ＳＴＲ規則のグラフ化はこの点にも貢献している。

なお本稿では、ＳＴＲコンパイラを作成する上で必要となる下記のアルゴリズムについては触れていない。巻末の参考文献並びに試作ソフトウエア仕様書、設計書を参照されたい。またＳＴＲ／Ｄコンパイラの内部ロジックにも触れていないが、ＳＴＲ／Ｄ言語は単純な言語でありインプリメントも容易である。

- ＳＴＲ規則中の否定記述を展開するアルゴリズム。
- ＳＴＲ規則中の副状態記述を展開するアルゴリズム。
- ＳＴＲ規則中の疑似イベントを処理するアルゴリズム。
- ＳＴＲ規則中の内部イベントを処理するアルゴリズム。
- ＳＴＲ規則中の時間制限付きプリミティブを処理するアルゴリズム。

参考文献

[1] 河田、平川、竹中“分散システムの記述とプロセス動作”情報処理学会全国大会, 4Q-5, 平成２年９月

[2] 河田、平川、竹中“通信サービス記述からプロセス動作仕様への変換の概要”電子情報通信学会全国大会, B-535, 平成３年３月

[3] 河田、平川、竹中“通信サービス仕様からプロセス動作仕様の生成”電子情報通信学会情報通信ネットワーキングアーキテクチャワークショップ, B-7, 平成３年１０月

[4] 河田、平川、竹中“通信サービス仕様からプロセス動作仕様の生成――通信サービス規則選択法――”電子情報通信学会全国大会, B-513, 平成４年３月

[5] K. Kawata, Y. Hirakawa “Efficient Process Generation from State Transition Based Telecommunication Service Specifications,” Proceedings of 5-th JC-CNSS, A1-1, pp. 3 - 8, Jul, 1992.

[6] 河田、田倉、太田“通信サービス仕様カスタマイズ言語の提案”電子情報通信学会全国大会, B-378, 平成４年９月

[7] 河田、田倉、太田“宣言型通信サービス仕様記述言語からＳＤＬへの変換法”情報処理学会全国大会, 5-375, 平成４年１０月

[8] 河田、田倉、太田“通信ソフトウエア生成システムにおけるネットワーク制御タスクの生成機構”電子情報通信学会 交換システム研究会, SSE92-46, 平成４年９月

[9] 河田、田倉、太田“宣言型通信サービス仕様記述言語からプロセス仕様を生成する手法”電子情報通信学会 人工知能と知識処理研究会, AI92-65, 平成４年９月

[10] 河田、田倉、太田“端末の動作記述からプロセス仕様を自動生成する手法について”電子情報通信学会 ソフトウエアサイエンス研究会 SS92-21, 平成５年１月

[11] 河田、田倉、太田“通信サービス仕様記述からプロセス仕様を生成する方法について”電子情報通信学会全国大会, B-579, 平成５年３月

[12] K. Kawata, A. Takura, T. Ohta “On a communication software generation method from communication service specifications described by a declarative language,” Proceedings of 5th International Conference on Computing and Information (ICCI’93), May, 1993. (to be published)

[13] K. Kawata, A. Takura, T. Ohta “Task generation mechanisms in a communication software generation system,” Proceedings of Asia-Pacific Conference on Communications (APCC’93), Aug, 1993. (to be published)

[14] K. Kawata, A. Takura, T. Ohta “Process specification generation from communication service specifications described by a graph grammar,” Proceedings of 7-th International Workshop on Distributed Algorithms (WDAG’93), Sep, 1993. (may be published)

[15] Y. Hirakawa, K. Kawata, T. Takenaka “Rule descriptions for telecommunication services and their transformation into standardized specification descriptions,” IEEE

8th International Conference on Software Engineering for Telecommunication Systems and Services, Apr, 1992.

[16] 田倉、河田、太田“端末動作に着目した通信サービス仕様記述とその実行環境について”電子情報通信学会 交換システム研究会, SSE92-102, 平成４年９月

[17] 田倉、河田、太田“端末動作記述によるサービス定義とその実行方式”電子情報通信学会全国大会, B-580, 平成５年３月

[18] 高見、平川、河田、竹中“ビジュアルプロトタイピングによる通信サービス仕様生成法”電子情報通信学会 交換システム研究会, SSE91-103, 平成３年１１月

[19] 平川“ＳＴＲ (State Transition Rule) 記述仕様書”ＡＴＲテクニカルレポート、TR-C-0073, 平成４年１月

A　プロセス仕様生成例

クラス４ＳＴＲコンパイラによって、ＳＴＲ規則の集合からプロセス仕様が生成される様子を、例を用いて説明する。なお、この例では分解木及び合成木中の点集合間の写像を点の対応表で実現している。

図25は通信サービス規則の記述例を示している。図26は通信サービス規則をグラフ表現した例を示している。図27は規則グラフの分解木の例を示している。図28は分解木を合成した合成木の例を示している。図29はプロセス仕様を構成する基本プロセス動作の例を示している。

```
[ rule1 ]
   a1(A), a2(B), a3(C), a4(A, B), a5(A, C), a6(B, D), a7(D)      /* 始広域状態 */
   ev(A) :                                                       /* イベント */
   b1(A), b2(B), b3(C), b4(D), b5(A, B), b6(A, D), b7(B, D).     /* 次広域状態 */

[ rule2 ]
   a1(A), a2(B), a3(C), a4(A, B), a5(A, C), a6(B, C)
   ev(A) :
   b1(A), b2(B), b3(C), b4(A, B), b5(B, C).
```

図 25:　ＳＴＲ規則例

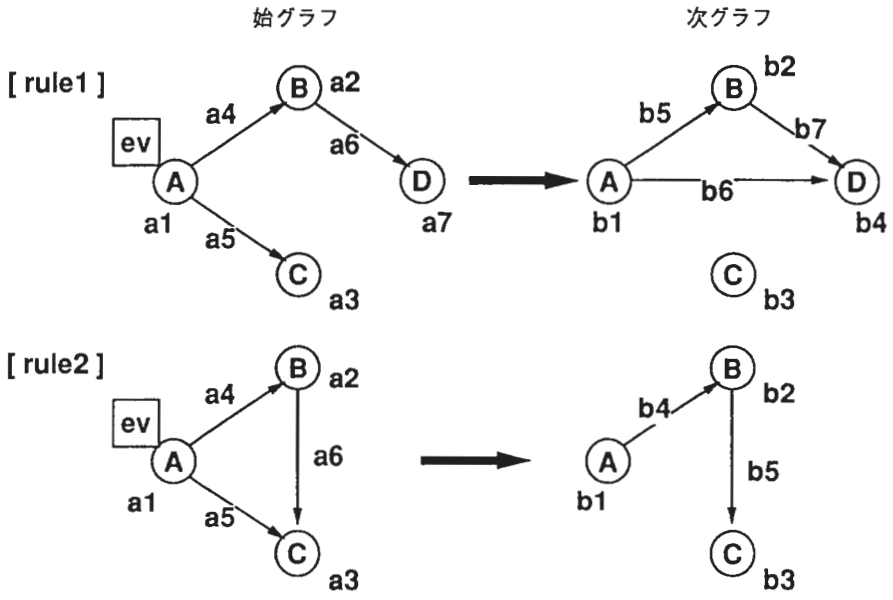

図 26: 規則グラフ例

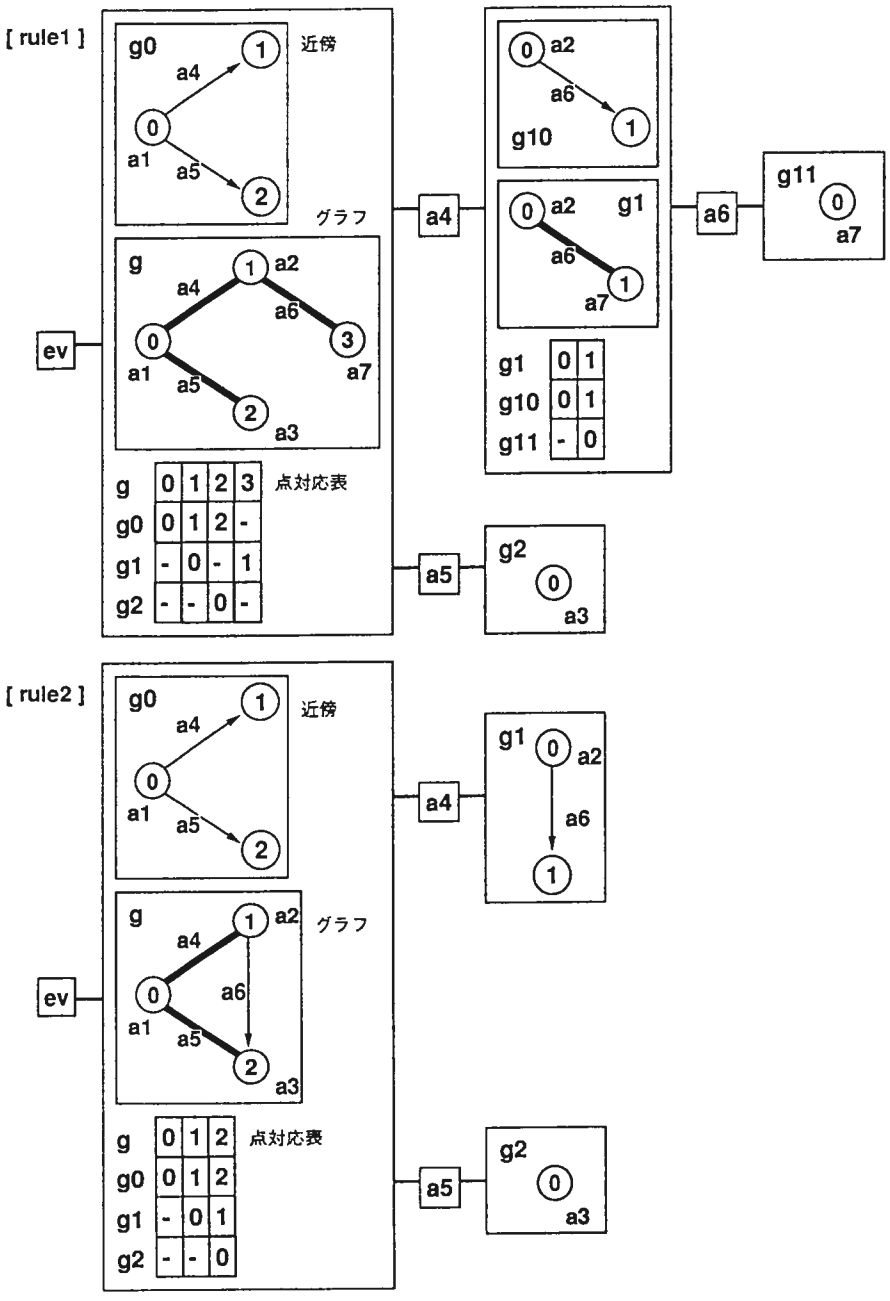
[rule1]
g0
近傍
a4
a1
a5
グラフ
g
a2
a6
a7
a3
ev
g 0 1 2 3
g0 0 1 2 -
g1 - 0 - 1
g2 - - 0 -
点対応表
a4
a5
a6
g10
g1
g11
g2
[rule2]
近傍
グラフ
点対応表

図 27: 分解木例

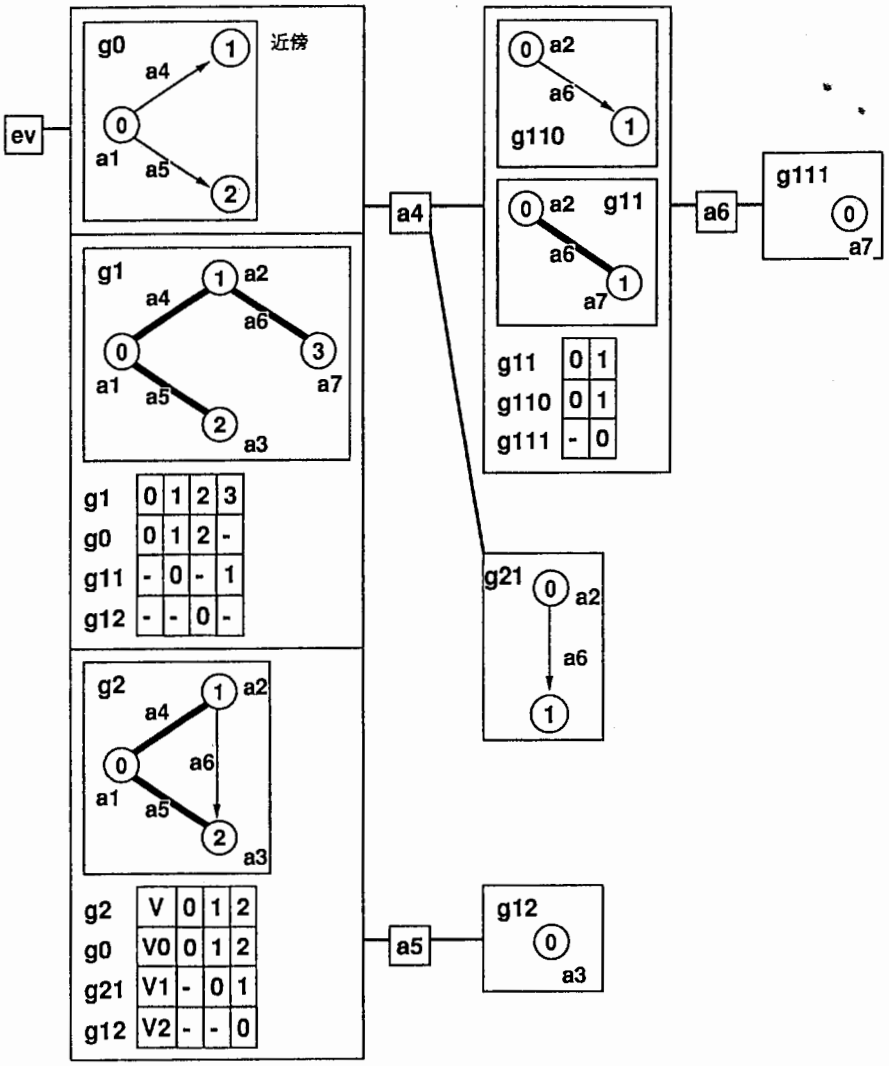
ev
g0
近傍
a4
a1
a5
g1
a2
a6
a7
a3
g1 0 1 2 3
g0 0 1 2 -
g11 - 0 - 1
g12 - - 0 -
g2
V
V0
V1
V2
g21
g12
g110
g11
g111
a6

図 28: 合成木例

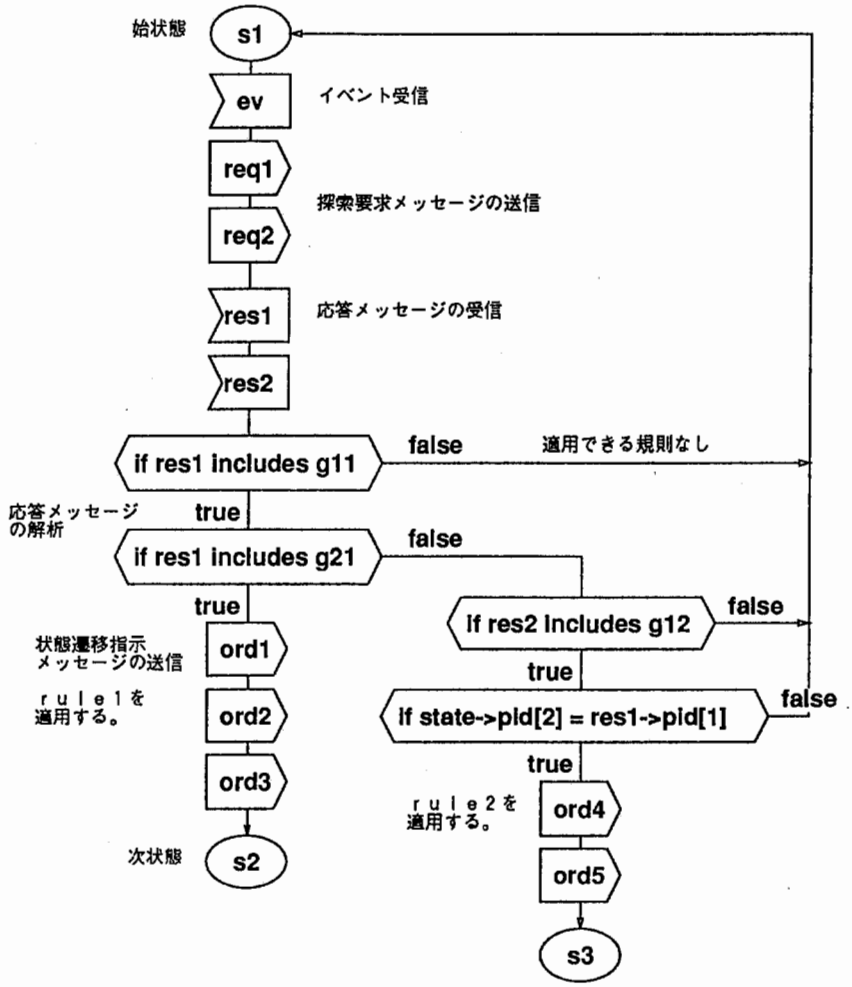
始状態
s1
ev
イベント受信
req1
探索要求メッセージの送信
req2
res1
応答メッセージの受信
res2
if res1 includes g11
false
適用できる規則なし
応答メッセージ
の解析
true
if res1 includes g21
false
true
if res2 includes g12
false
状態遷移指示
メッセージの送信
ord1
true
r u l e 1 を
適用する。
ord2
if state->pid[2] = res1->pid[1]
false
ord3
true
r u l e 2 を
適用する。
ord4
次状態
s2
ord5
s3

図 29: プロセス仕様例