〔公　開〕

ＴＲ－Ｃ－０１２０

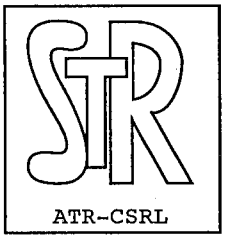
ＳＴＲ（State Transition Rule）記述仕様書

榎木　浩　Hiroshi ENOKI
若林　学　Manabu YAKABAYASHI
上田　佳寛　Yoshihiro UEDA
原田　良雄　Yoshio HARADA

１９９５　４．１２

ＡＴＲ通信システム研究所

STR (State Transition Rule) 記述仕様書



平成６年４月　STR-2.0

通信ソフトウェア研究室

目次

本仕様書に関する問い合わせ先
〒619-02　京都府相楽郡精華町光台2-2
ATR通信システム研究所　通信ソフトウェア研究室
Tel: 0774-95-1211 Fax: 0774-98-2054

1．はじめに

本稿は，通信サービスの動作記述のために開発されたＳＴＲ(StateTransition Rule)の記述方法について概説する．ＳＴＲは，ＡＴＲ通信システム研究所において，平成元年春より研究が開始された仕様記述言語であり，主に通信ソフトウェアの研究に用いられる．

ソフトウェア開発では，まずソフトウェアで実現すべきことを仕様として記述する．ソフトウェアで実現すべき機能を事前に記述することにより，早期段階での設計上の問題点を明らかにできる．

現在，仕様記述言語として用いられているもののほとんどは，手順的な記述，即ち時系列的な処理の流れを記述するものである．この手法には次の２つの問題がある．

(1) 通信サービスで，複数のサービスが絡んだ動作の簡潔な記述が困難

現状では，設計の初期工程では，単一サービスの動作を記述し，複数サービスが絡んだ場合の動作は自然言語で記述される程度で，複数サービスが絡んだ場合の動作は，設計の後工程になって初めて規定している．またその記述は，１つの端末に対して全ての動作を含み，極めて複雑でプログラム動作に近い詳細なアルゴリズムとなっている．

(2) 内部構造が異なるシステムでは，同じサービスの動作でも記述が異なる

通信システムの場合は，システムによるさまざまな構造を持つものがある．しかし，利用者から見た場合，システムの構造とは無関係に，その上で実現されるサービス動作は同じ場合がほとんどである．現在の仕様記述では，同じサービスを実現する場合でも，目的のシステムの構造が異なると，新たな仕様記述を行う必要がある．

そこでＳＴＲではこのような問題を解決すべく，これまでの手順的な記述を採用せず，宣言的な記述方法を用い，以下の点を狙いとしている．

(1) システムの内部構造によらず外部から認識できる動作が同じであるとき，その内部構成に依存しない外側からの記述で規定したい．

(2) 外側からの記述の段階で，サービス複合時の場合の動作も含め，形式的に動作を規定したい．

(3) 外側から記述された動作の段階で，サービス毎の論理的な衝突を検出し，サービスが組み合わされた状況に対する動作設計の支援をしたい．

(4) 外部からの記述を行った後，計算機の支援を受けながら，内部構造に依存したものに詳細化したい．

このように，ＳＴＲは，目的のシステムの内部構造には無関係な記述となるように，システムの内部構造はブラックボックスと捉え，利用者からの動作と利用者から認識できる対象物（端末，および人）の状態を基本としてサービス動作の規定を行う．

2．ＳＴＲ概論

2．1　概要

ＳＴＲは，サービス動作をシステムの内部構成によらず外部から認識できる動作で利用者の立場から規則として記述するものである．その規則の記述は，

「現状態」　「イベント」　「次状態」

の３つの要素で構成され，１つのイベントが生起した際に影響を受ける範囲を指定し，その

範囲に含まれる対象物の動作を規則として記述する手法である．サービス動作は，その規則の集合で規定される．

またＳＴＲでは，現状態とイベントは規則を適用する条件部とし，次状態は規則が適用された結果部となる．

２．２　状態

ＳＴＲでは，規則に記述する「現状態」「次状態」の他に，「実状態」という概念を使用している．また，「現状態」「次状態」は，状態の記述要素となる「状態記述要素」の集合として記述される．そこで，各状態の意味について説明する．

２．２．１　実状態

システムのある時点での，全ての対象物の状態を「実状態」という．（後の４章　ＳＴＲの適用規則を参照）．

２．２．２　現状態

ＳＴＲの規則の条件部で，規則を適用するための対象物の状態を表わすものを「現状態」という．

２．２．３　次状態

ＳＴＲの規則の結果部で，規則を適用した後の対象物の状態を表わすものを「次状態」という．

２．２．４　状態記述要素

規則を記述する際の現状態，次状態を構成する基本となる状態要素を「状態記述要素」といい，状態記述要素は，対象物の状態や，対象物と他の対象物の関係を表す状態をその名前と対象物の識別子により記述する．対象物の識別子には，端末識別子と人識別子があり，説明上，区別する必要がある場合は，それぞれ使い分ける．共通の説明については，識別子として表現する．

状態記述要素を用いたＳＴＲの状態表現には，次の４種類がある．

(1)　状態記述要素

状態記述要素には，対象物の状態そのものを表わす状態記述要素と，対象物でのサービスの登録を表わす状態記述要素がある．状態記述要素の引数は１つまたは２つである．第１引数は主体となる対象物を表わし，第２引数は主体と関連する対象物を表わす．

(2)　マクロ状態記述要素

複数の一般の状態記述要素を１つの状態記述要素で表現できる．この状態記述要素を「マクロ状態記述要素」という．常に状態記述要素のある組み合わせでしか現れないものや，組み合わせた方が概念的に理解し易いものについて，このマクロ状態記述要素を使用する．

(3)　時間制限の状態記述要素

サービスの仕様で，ある状態が一定時間経過したとき，自動的に次の状態に遷移するといった記述をしたいとき，予め時間制限の対象となる状態記述要素を定義し，タイムオーバイベント（次頁　イベントを参照）と共に用いる．この定義した状態記述要素を「時間制限の状態記述要素」といい，状態記述要素の集合で表される．

(4)　禁止状態記述要素の集合

規則の適用時，次状態に遷移しては困る場合に，規則以外の次状態に遷移することができ

る．これは，規則を適用後の状態に或る状態記述要素の集合が含まれていれば，その規則適用以外の状態遷移ができる仕組みのためである．この或る状態記述要素の集合を禁止状態記述要素の集合という．

(5) カウンタ付状態記述要素

サービス仕様で，ある状態が何度目かによってその回数をカウントしておきその値によって次の遷移が異なるといった記述をしたい（誤動作を何度かは許容するがある一定回数以上は認めないといったもの）とき，予め対象となる状態記述要素とその回数を定義する．この定義した状態記述要素を「カウンタ付状態記述要素」という．

カウンタ付状態記述要素を用いるには以下のような宣言と，以下のような定義に従った記述法が必要となる．

・対象とする状態記述要素と回数の宣言（３．３．５参照）
・カウンタの値の制御（４．５．２参照）を行う規則の記述法
・カウンタ値の値の判定（５．１．１．５参照）を行うための記述法
・カウンタ付状態記述要素がある規則の適用方法（４．５．１参照）

２．３　イベント

ＳＴＲでは，一般に対象物の動作を「イベント」といい，規則適用のトリガーとなる条件である．イベントは，対象物の動作を，そのイベント名と識別子で記述する．識別子の扱いとして，対象物の単独のイベントは１つの識別子，関係を表すイベントは自身の識別子と相手の識別子の２つの識別子を記述する．

イベントには次の４種類がある．

(1)　ユーザイベント

一般にイベントと呼ぶもので，ユーザが対象物（端末および人）に対して行う論理的な動作を「ユーザイベント」と言う．

(2)　タイムオーバイベント

サービスの仕様で，対象物のある状態が一定時間経過したとき，自動的に次の状態に遷移するといった記述のとき，そのときのイベントを「タイムオーバイベント」という．

(3)　疑似イベント

対象物がある状態になったことを契機として，規則が適用され，次状態に遷移するとき，そのイベントを「疑似イベント」という．

(4)　内部イベント

ある対象物でのイベントの発生により，内部的に発生するイベントを「内部イベント」という．

３．　記述要素の宣言

３．１　記述要素の宣言文字

ＳＴＲでは使用する記述要素を宣言文として定義しなければならない．ここでは，ＳＴＲで用いる英数字について説明する．

［記述に用いる英数字］

number　　：　0～9の数字

char　：a～zの英小文字と-（ハイフォン）
Var　：識別子を表わす先頭がＡ～Ｚの英大文字で，以降はnumber, char, Ａ～Ｚの任意の文字列
name-string　：先頭文字がnumberとcharの任意の文字で，以降はnumber, char, Ａ～Ｚの任意の文字列
internal-event-name　：先頭文字がnumberとcharの任意の文字で，以降はnumber, char, Ａ～Ｚの任意の文字列
Capital-starting-name　：Ａ～Ｚの英大文字先頭文字で，以降はnumber, char, Ａ～Ｚの任意の文字列
" "　：" "内の文字はそのまま記述に用いる固定文字
¦　：ORを表す記号
primitive　：状態記述要素でname-string(Var)またはname-string(Var, Var)で表現
primitive-string：状態記述要素をカンマで区切った状態記述要素列
event　：ユーザイベントでname-string(Var)またはname-string(Var, Var)で表現
event-string　：ユーザイベント，タイムオーバイベント，及び疑似イベント
event-strings　：event-stringをカンマで区切った列
internal-event　：内部イベントでinternal-event-name (Var)またはinternal-event-name (Var, Var)で表現
internal-event-string　：内部イベントをカンマで区切った列
inhibit-primitive-string : name-string($)又はname-string($, Var)で表現した状態記述要素の例

３．２　識別子の宣言

識別子には，端末と人の識別子があり，それぞれに使用する識別子を予め宣言する．識別子の宣言では，識別子の型（端末／人）と識別子に使用する先頭英大文字を記述する．但し，識別子の宣言を省略した場合は，使用する識別子の型は端末となる．

［記法］
識別子の型：端末(Terminals)
：人 (Users)
識別子の先頭文字の集合(var-string)：A¦..¦Zをカンマで区切った列
注）・A¦..¦ZはA～Zのいずれかの英大文字を示す

［宣言］
Terminals:
var-string
Users:
var-string

［例］
Terminals:
A,B,C
Users:
U,V

この例ではA,B,Cで始まる識別子は端末識別子で，U,Vで始まる識別子は人識別子となる．

３．３　状態記述要素の宣言

３．３．１　状態記述要素の宣言

ＳＴＲ記述で用いる状態記述要素は，予め宣言する必要がある．一般の状態記述要素の記法は，状態記述要素の名前を表すname-stringと識別子を表すVarにより先頭文字は英小文字として記述する．

［記法］

状態記述要素　：　name-string(Var)　または　name-string(Var, Var)

状態記述要素の宣言は，「Primitives：」の後に，使用する状態記述要素をカンマで区切り列挙する．（）内の第１引数の識別子が状態記述要素を保持する主体を表し，第２引数の識別子は，関連する対象物を表す．同一name-stringでも，識別子の型が異なる場合には，各々の状態記述要素を宣言する．

［宣言］

```
Primitives :
        primitive-string
```

［例］

```
Primitives :
        idle(A), dial-tone(A), busy(A), busy-dial(A,B),
        path(A,B), hangup(A), ringing(A,B),
        ringback(A,B), path-passive(A,B)
```

これは，通信サービスのＰＯＴＳ（基本電話サービス）で使用する状態記述要素の宣言例である．それぞれの状態記述要素とその意味について説明する．

idle(A)	：端末Ａが未使用の状態
dial-tone(A)	：端末Ａが発信音を受信している状態
busy(A)	：端末Ａがビジートーンを受信している状態
busy-dial(A,B)	：端末Ａがダイヤル後相手ビジー（idle以外）のときの音を受信している状態
path(A,B)	：端末Ａが端末Ｂに通話している状態
hangup(A)	：端末Ａの受話器が上がりっぱなしの状態
ringing(A,B)	：端末Ａに端末Ｂからの呼び出しベルが鳴っている状態
ringback(A,B)	：端末Ａが端末Ｂへの呼び出しをしている状態
path-passive(A,B)	：端末Ａが端末Ｂを保留している状態

また，サービスの登録を表わす状態記述要素は，m-を付けて端末の状態を表わす状態記述要素と区別して宣言している．その宣言例を示す．

m-cw(A)	：端末Ａが話中着信サービスの権利を登録している状態
m-regcfv(A,B)	：端末Ａが着信転送を端末Ｂに登録している状態

３．３．２　マクロ状態記述要素の宣言

マクロ状態記述要素は，複数の状態記述要素を１つの状態記述要素として表現するものである．マクロ状態記述要素には，状態記述要素のＡＮＤの集合を表すものと，状態記述要素のＯＲの集合を表すものがある．マクロ状態記述要素は，先頭を英大文字とすることで状態記述要素と区別する．

［記法］

マクロ状態記述要素　：　Capital-starting-name(Var)　　または
Capital-starting-name(Var, Var)

マクロ状態記述要素の宣言は，「Macro-Primitives：」の後に，使用するマクロ状態記述要素と，｛｝内にマクロ状態記述要素を構成する状態記述要素の集合を記述する．状態記述要素のＡＮＤの集合を表すマクロ状態記述要素は，状態記述要素の間をカンマで区切り，状態記述要素のＯＲの集合を表すマクロ状態記述要素は，状態記述要素の間を｜で区切り両端を（　）で括る．

［宣言］

Macro-Primitives :
macro-primitive ＝ {primitive-string}
macro-primitive ＝ {(primitive｜. .｜primitive)}

［例］

Macro-Primitives :
Calling(A,B) = { ringback(A,B), ringing(B,A) }
Busy(A,B)　= { (busy(A)｜busy-dial(A,B)) }

この例では，ringback(A,B)とringing(B,A)の２つの状態記述要素をまとめてCalling(A,B)として表している．また，Busy(A,B)はbusy(A)またはbusy-dial(A,B)のどちらかを示すマクロ状態記述要素である．

３．３．３　時間制限の状態記述要素の宣言

タイムオーバイベントのとき，時間制限の対象となる状態記述要素の集合を，時間制限の状態記述要素として宣言する．時間制限状態記述要素の宣言は，対象となる状態記述要素の集合とその制限時間を秒で記述する．また，時間制限の対象は，複数の対象物に対して（状態記述要素の第１引数の識別子が違う場合）可能であるが，状態記述要素の集合の中で，対象物の関係が辿れなければならない（５．３．２　制約６参照）．

［宣言］

Limited-Time-Primitives :
primitive-string　number "sec"

［例］

Limited-Time-Primitives :
busy(A)　30sec
busy(A), m-3wc(A), 3wc2(A,B)　30sec

これは，busy(A)或いはbusy(A), m-3wc(A), 3wc2(A,B)が現状態に含まれるとき，３０秒以

内に状態が変化しない場合，タイムオーバーとして規則に記述された次状態に遷移する．（５章のタイムオーバイベントの記述を参照）

３．３．４　禁止状態集合の宣言

規則の適用時，次状態に遷移しては困る場合に，規則以外の次状態に遷移することができる．これは，規則を適用するとき，次状態に或る１つの対象物の状態記述要素の集合が含まれていれば，その規則以外の状態遷移ができる仕組のためである．その状態記述要素の集合を定義するのが禁止状態集合の宣言である．

禁止状態集合の宣言は，「Inhibit-Primitive-Sets：」の後に，禁止対象状態として禁止対象となる状態記述要素の集合を｛ primitive-string ｝で記述し，次に禁止適用状態として，禁止適用後の状態記述要素の集合を(inhibit-primitive-string) で記述する．また，禁止適用状態を記述しないことも可能である．

［宣言］

Inhibit-Primitive-Sets :

{ primitive-string } (inhibit-primitive-string)

このとき，禁止対象状態の記述は，状態記述要素の第１引数の識別子が全て同じ（１つの対象物についての記述）でなければならない．また，禁止適用状態記述の状態記述要素の第１引数の識別子は必ず＄である．＄という識別子は禁止状態に遷移するようなイベントを生起した対象物を表す．また，禁止適用状態の状態記述要素に第２引数がある時は，その識別子は禁止対象状態に現われる識別子の何れかである必要がある．

［例］

Inhibit-Primitive-Sets :

{ cw-ringing(A,B), cw-ringing(A,C) } (busy($))

規則）　dial-tone(A), not[idle(B)], cond : m-cw(B), path(B,C)

dial(A,B) :　ringback(A,B), path(B,C), cw-ringing(B,A).

これは，上例の規則を適用しようとした時，次状態にcw-ringing(A,B), cw-ringing(A,C)が含まれた場合，その規則が適用されず，busy($) という次状態に遷移することを規定している．

３．３．５　カウンタ付状態記述要素の宣言

カウンタ付状態記述要素の宣言は以下のように行う．

［宣言］

Count-Primitives:

primitive　　number

［例］

Count-Primitives:

m-retry(A)　　5

これは，m-retry(A)がもつカウンタ値が５のときとそうでないとき回数判定記述（５．１．１．５参照）を用いることで違った状態遷移を記述できることを意味する．

３．３．６　対象物の初期状態の宣言

対象物には端末と人があり，各々の初期状態を表す状態記述要素の集合を宣言する．但し，初期状態は一つの対象物の状態記述要素の集合でなければならない．

［記法］

対象物の初期状態：primitive-string

［宣言］

Terminal-Initial-Primitive :
　　primitive-string
User-Initial-Primitive :
　　primitive-string

［例］

Terminal-Initial-Primitive :
　　idle(A)
User-Initial-Primitive :
　　m-upth(U,A)

この例では端末の初期状態はidle(A)，人の初期状態はm-upth(A,B)である．

３．４　イベントの宣言

３．４．１　イベントの宣言

ＳＴＲでは外部から認識できる動作をイベントとして自由に記述できるが，状態記述要素同様，使用するイベントを宣言文として定義しなければならない．イベントには，ユーザイベント，疑似イベント，タイムオーバイベント，内部イベントがある．

イベントの宣言文では，ユーザイベント，疑似イベント，タイムオーバーイベントを宣言し，「Events :」の後に，使用するイベントをカンマで区切り列挙する．

ユーザイベントは（）内の第１引数の識別子がイベントを発生する主体を表し，第２引数の識別子は，相手の対象物を表す．同一name-stringでも，識別子の型が異なる場合には，各々のイベントを宣言する．

疑似イベントは，複数の対象物に対して（状態記述要素の第１引数の識別子が違う場合）可能であるが，状態記述要素の集合の中で，対象物の関係が辿れなければならない（５．３．２　制約６参照）．

［記法］

・ユーザイベント　：　name-string(Var) または　name-string(Var, Var)
・疑似イベント　：　[primitive-string]
・タイムオーバーイベント：　timeover(primitive-string)

［宣言］

Events :
　　event-string

［例］

Events :
　　dial(A,B), flash(A), [idle(A)],timeover(dial-tone(A))

３．４．２　内部イベントの宣言

内部で発生する内部イベントは他のイベントと区別するために別に宣言する．内部イベントもユーザイベントと同様，同一name-stringでも識別子の型が異なる場合には，各々の内部イベントを宣言する．

［記法］

・内部イベント　　：　internal-event-name(Var, Var)

［宣言］

Internal-Events :

internal-event-string

［例］

Internal-Events :

sig-onhook(A,B)

３．４．３　内部イベント分配の宣言

ある対象物にイベントが発生したことにより，その対象物と関係があるすべての対象物に新たな内部イベントを発生させるため，予め内部イベントの分配を宣言することができる．その宣言は，「Internal-Event-Deliveration : 」の後に対象となるイベントと，「-->」の後に新たに発生させる内部イベントを記述する．このとき，内部イベントの第１引数にはイベント生起対象物の識別子，第２引数にはイベント生起対象物と関係のある全ての対象物の識別子として１つの識別子を記述する．但し，同一name-stringでも第２引数の識別子の型が異なる場合には，or記述を用いて各々を記述する．

［宣言］

Internal-Event-Deliveration :

event --> internal-event-name (Var, Var)

event --> internal-event-name (Var, Var)¦ internal-event-name (Var, Var)

［例 １］

Internal-Event-Deliveration :

onhook(A) --> sig-onhook(A, B)

オンフック動作の場合，広域的に記述しようとすると，規則数が関連端末の状態数の乗算的に増えてしまい，記述が大変である．そのため，この内部イベントの分配を定義し，使用することにより，オンでフックした際の動作は，比較的定形的に扱うことができる．この宣言では，sig-onhookという内部イベントの分配により，端末Aがオンフックした場合，その時に端末Aと関係のある端末全てにsig-onhookを送信することになる．

［例２］

Internal-Event-Deliveration :

onhook(A) --> sig-onhook(A, B)¦ sig-onhook(A,U)

この例ではsig-onhookという内部イベントを端末Aと関係のある端末と人に分配することとなる．

３．４．４　内部イベント分配制限の宣言

内部イベントをデフォルト宣言により分配するとき，その送信先を制限するため，制限の条件を追加することができる．その制限条件は，イベント生起端末（内部イベントの送信元）を第１引数に，送信の対象となる対象物を第２引数に持つ状態記述要素の集合を指定するものである．宣言は，「Delibery-range：」の後に，内部イベントと送信を制限する状態記述要素の集合を記述する．一般には，宣言した状態記述要素またはその集合が実状態にある場合に，内部イベントを分配しない．

［宣言］

Delibery-Range :

range(internal-event-name (Var, Var): primitive-string) = {　}

［宣言］

Delibery-Range :

range(sig-onhook (A,B): path-passive(A,B)) = { }

この例は，保留中の状態記述要素 path-passive(A,B) に対して，内部イベントの sig-onhook が送られないことを表している．

４．　ＳＴＲの規則適用

４．１　規則の基本適用方法

ＳＴＲでは，ある実状態（実際の対象物から構成される状態）において，イベントが発生したとき，規則の現状態が実状態に含まれ，かつ，発生イベントと規則イベントが等しいとき，その規則が適用され，実状態中の規則の現状態に相当する部分が次状態に相当する部分に書き変わる．

Ｇを実状態，ＧＥを実状態で生起したイベント，ＩＳｉを規則Ｒｉの現状態，Ｅｉを規則Ｒｉのイベント，Ｒｉを規則とすると，以下の関係が成り立つとき規則Ｒｉが適用される．

（Ｇ⊇ＩＳｉ）∧（ＧＥ＝Ｅｉ）

次のような状況を考えてみよう．通信システム内にＰ，Ｑ，Ｒ，Ｓという端末があり，それぞれの端末の状態が，以下の通りとする．

端末Ｐ　dial-tone(P)

端末Ｑ　dial-tone(Q)

端末Ｒ　idle(R)

端末Ｓ　idle(S)

そこで次の規則を考える．

dial-tone(A), idle(B)　dial(A,B)：　ringback(A,B), ringing(B,A).

このとき，端末Ｐが端末Ｒにダイヤルしたとき，Ａ＝Ｐ，Ｂ＝Ｒとみなすことで規則の適用条件が満足され，その結果規則が適用され端末ｐが端末ｒを呼び出している状態に遷移する．また，端末Ｑが端末Ｓにダイヤルしたときも，Ａ＝Ｑ，Ｂ＝Ｓとして規則が適用される．

４．２　規則の優先適用

複数の規則の条件部の関係に包含関係があるとき，即ち，ある規則の条件部より更に詳細な条件を記述する規則があった場合，その両方の規則が適用可能な状況では，一方を含んでいる規則を優先的に適用する．

Ｇを実状態，ＧＥを実状態で生起したイベント，ＩＳｉ，ＩＳｊを各々規則Ｒｉ，Ｒｊの現状態，Ｅｉ，Ｅｊを各々規則Ｒｉ，Ｒｊのイベント，Ｒｉ，Ｒｊを規則とすると，以下の関係が成り立つとき規則Ｒｉを優先的に適用する．

（（Ｇ⊇ＩＳｉ）　∧（ＧＥ＝Ｅｉ））∧
（（Ｇ⊇ＩＳｊ）　∧（ＧＥ＝Ｅｊ））∧
（ＩＳｉ⊃ＩＳｊ）∧（Ｅｉ＝Ｅｊ）

次に優先適用を具体例で示す．次の２つの規則を考える．

規則１　dial-tone(A), idle(B) dial(A,B) : ringback(A,B), ringing(B,A).

規則２　dial-tone(A), idle(B), cond : m-regcfv(B,C), idle(C)
　　dial(A,B) : ringback(A,C), pingring(B,A), ringing(C,A).

このとき，端末Ｐが dial-tone (P)，端末Ｑが idle (Q), m-regcfv (Q, R)，端末Ｒが idle (R) の状態とすると，端末ｐが端末ｑにダイヤルしたときには，規則１，規則２の両方が適用可能である．しかし，この場合，優先適用により，規則の現状態を包含している規則である規則２が適用される．

４．３　各種イベントの適用

４．３．１　タイムオーバイベントの適用

時間制限の対象となる状態記述要素の集合を記述するタイムオーバイベントでは一般に次のような適用となる．

(1) タイマの設定（開始）

ある規則を適用したことにより，時間制限の状態記述要素の集合が実状態に現われたとき，タイマが設定される．このとき，同時に設定可能な複数の時間制限の宣言の間に包含関係がある場合は，詳細な記述の方のタイマを設定する．

規則）　pn　en:　　p1,p2.
　Limited-Time-Primitives :
　　p1　　xsec
　　p1, p2　　ysec

このとき，p1, p2のタイマを優先的に適用する．

(2) タイマの解除

タイマのリセットは，宣言した状態記述要素の集合の一部でも実状態から消去された場合，包含関係にある別の時間制限の宣言にある状態記述要素の集合が現われた場合，また，タイムオーバーの規則が適用された場合に行われる．

(3) タイムオーバー

時間制限の対象として宣言された状態記述要素の集合がタイムオーバーイベントに指定されており，宣言した時間内に宣言された状態（状態記述要素の集合）が変化しない場合，宣言した時間をもって次状態に遷移する．他の扱いは，規則の記述と同様に，現状態記述を次状態記述に書き換える．

Limited-Time-Primitives :

dial-tone(A)　　30sec

dial-tone(A)　　timeover(dial-tone(A)) :　　busy(A).

この例では現状態がdial-tone(A)のとき，３０秒以内に状態が変化しない場合，timeover (dial-tone(A))イベントにより，次状態busy(A)に遷移する．

４．３．２　内部イベント分配制御の適用

内部イベントの分配制限では，宣言されている状態記述要素の第２引数である対象物に対して内部イベントの分配を制限する．従って，イベント生起端末の状態に分配制限されていない状態記述要素がある場合には，その第２引数の対象物には内部イベントを分配することとなる．

［例］

端末Aの状態　s1(A, B), s2(A, B)

端末Bの状態　s2(B, A)

内部イベントの分配制限の宣言

Delibery-Range :

range(internal-event : s1(A,B)) = {　}

規則

s2(A, B)　internal-event-name(B, A) :　s3(A).

この例では，端末Bには内部イベントがs2(B, A)に対して分配される．端末Bに内部イベントを分配しないためには，

Delibery-Range :

range(internal-event : s1(A, B)) = {　}

range(internal-event : s2(A, B)) = {　}

または

Delibery-Range :

range(internal-event : s1(A, B), s2(A, B)) = {　}

のいづれかの内部イベント分配制限の宣言が必要である．

４．３．３　疑似イベントの適用

疑似イベントに指定された対象物の状態が疑似イベントに指定された状態記述要素の集合を持つ状態になったとき，規則は適用可能となり，現状態を満足すれば，現状態の記述を次状態の記述に書き換える．これは，規則の適用およびそれに関する制約と同様である．同時に適用可能な場合の優先適用の扱いを以下に示す．

現状態と疑似イベントの状態記述要素の和集合をとり，その和集合が一方を包含している

方の規則を優先して適用する．

規則１）　p1　　[p3]　　p4.

規則２）　p1, p2, p3　　[p1, p2]　　p5.

規則１の和集合は＜p1, p3＞，規則２の和集合は＜p1, p2, p3＞となり後者が前者を包含しており，規則２を優先して適用する．

（注）疑似イベントの適用は他のイベントに優先した扱いとする（疑似イベントの規則が適用可能な場合は，疑似イベントの規則を適用する）．また，疑似イベントの規則適用後の状態に対して，さらに疑似イベントの規則が適用されることに対する制限はない．

４．４　禁止状態の適用

４．４．１　禁止状態の適用

禁止状態は，各種イベント（ユーザイベント，内部イベント，疑似イベント，タイムオーバーイベント）に対応する規則適用の後の状態を判定し，禁止対象状態に指定した状態記述要素の集合を含む場合，それらの規則適用以外の状態遷移を行う．この際のイベント生起対象物の次状態構成手順は次の通りである．

(1) 禁止状態で (inhibit-primitive-string) のあるときは，規則が適用される場合に現状態から取り除かれる状態記述要素を，現状態取り除き，その後「inhibit-primitive-string」を付加する．

(2) 禁止状態で「(inhibit-primitive-string)」の明記されていない場合には，状態は現状態のまま変化しない．

この禁止状態に関するチェックは，通常の適用規則決定のあとに行われる．従って，禁止状態に抵触したために適用規則が変更されることはない．

また，複数の禁止状態集合に包含関係がある場合，優先適用により詳細な記述の禁止状態集合が適用される．

禁止状態集合の記述例を以下に示す．

Inhibit-Primitive-Sets :

{ cw-ringing (A, B), cw-ringing (A, C) } (busy ($))

これは，１つの端末の実状態に上記の集合が含まれることを禁止するものである．

例えば，以下の記述を考えよう．

規則１）dial-tone (A), not [idle(B)], cond : m-cw (B), path (B,C)

dial (A,B) : ringback (A,B), path (B,C), cw-ringing (B,A).

規則１を適用しただけでは，端末Ａが話中着信を受けている状況で，更に他の着信があったとき，規則の条件が満足されるため，いくらでも呼びが着信してしまう．これを防ぐための記述が禁止状態集合である．

いったん話中着信を受け付けている状態で他の端末からダイヤルされた場合に，規則１を適用しようとすると，上記の禁止状態集合を持つ端末が現われてしまう．その場合，この規則の適用を行わず，その原因となったイベント生起端末をビジー音接続とする．

４．５　カウンタ付状態記述要素の適用

４．５．１　カウンタ付状態記述要素のある規則の適用方法

カウンタ付状態記述要素が現状態に状態記述要素としてのみある場合通常の状態記述要素と同じようにその状態が実状態に含まれていたら規則を適用できる．

カウンタ付状態記述要素が現状態に回数判定記述（５．１．１．５参照）とともにある場合その状態が実状態に含まれてかつカウンタ付状態記述要素が持っているカウンタ値が予め宣言した値と回数判定記述で定義した関係にあるとき規則を適用できる．

４．５．２　カウンタ付状態記述要素がもつカウンタ値の制御

カウンタ付状態記述要素がもつカウンタ値の制御は以下のように行う．（以下の規則例ではcs(A)がカウンタ付状態記述要素とする）

（１）カウンタ値の初期化

カウンタ値の初期化はカウンタ付状態記述要素が付与されるとき値を１として設定することで行う．つまり規則ではカウンタ付状態記述要素が現状態にはなく次状態のみにある場合に相当する．

（例）　s1(A,B)　e(A,B):　s2(A),cs(A).

（２）カウンタ値のインクリメント

カウンタ値のインクリメントは状態として参照されるが置き換え対象でないときカウンタ値を１だけプラスすることで行う．つまり規則ではカウンタ付状態記述要素が現状態にも次状態にもある場合に相当する．

（例）　s1(A,B),cs(A)　e(A,B):　s2(A),cs(A).

（３）カウンタ値のクリアー

カウンタ値のクリアーはカウンタ付状態記述要素が削除されることで必然的に行われる．つまり規則ではカウンタ付状態記述要素が現状態にはあるが次状態にない場合に相当する．

（例）　s1(A,B),cs(A)　e(A,B):　s2(A).

５．　ＳＴＲの記述

５．１　状態の記述

５．１．１　現状態

５．１．１．１　一般の現状態記述

現状態は，状態記述要素の集合として，状態記述要素，マクロ状態記述要素をカンマで区切り記述する．

［記法］

現状態　：　primitive-string

［例］

dial-tone (A), path(A,B), m-cw (A)

dial (A,B) : CW-ringing (A,B), path (A,B), m-cw(A).

上記規則の「dial-tone (A), path (A,B), m-cw (A)」が現状態の記述である．

５．１．１．２　ｃｏｎｄ記述

状態遷移に際して，現状態の条件として参照はするが，次状態への置き換えの対象にならない状態記述要素の集合については，現状態と次状態に重複して記述してもよいが，現状態のみに略記することができる．この略記をｃｏｎｄ記述という．ｃｏｎｄ記述は現状態記述にのみ使用できる．

［記法］

・状態記述要素が１つの場合　：　cond : primitive

・状態記述要素が複数の場合　：　cond : (primitive-string)

［例］

規則１）　dial-tone (A), path (A,B), m-cw(A)

dial (A,B) :　CW-ringing (A,B), path (A,B), m-cw(A).

規則２）　dial-tone (A), cond : path (A,B), cond : m-cw (A)

dial (A,B) :　CW-ringning (A,B).

規則１で，状態記述要素 path(A,B)とm-cw(A)は，現状態と次状態で状態が変化しないため，規則２では cond : path (A,B), cond : m-cw (A) として現状態のみに略記している．

５．１．１．３　ｎｏｔ記述

(1) notの記述

対象物の状態が，ある状態記述要素の集合を含まない状態であることをnot［状態記述要素の集合］で記述する．ｎｏｔ記述も現状態記述のみに使用でき，またｃｏｎｄ記述と併用はできない．

［記法］

ｎｏｔ記述　：　not [primitive-string]

［例］

not [idle(B)]

これは，端末Bがidle(B)という状態記述要素を含まない状態にあることを示している．

また次は複数の状態記述要素をnot記述した例では，論理式の展開に準じて以下の様に展開される．

［例］

not [p1,p2,p3] = not[p1] ¦ not[p2] ¦ not[p3]

(2) notの適用

not記述は指定した状態記述要素を含まないことである．そのため複数の適用可能な規則があり，その規則の現状態にnot[状態記述要素の集合]がある場合には，以下の取扱を行った後，規則の優先適用の判定を行う．

not[状態記述要素の集合]の取扱：

not[状態記述要素の集合]を，比較する他方の規則の同一対象物に付加する．但し，他方の規則の同一対象物に同一のnot[状態記述要素の集合]がある場合には付加は行わない．

［例］

規則１）p1(A) , not[p2(B)]　ev(A,B) :　p3(A).

規則２）p1(A), p4(B)　ev(A,B) :　p5(A,B), p6B,A).

端末P=p1(P)，端末Q=p4(Q)で，イベントev(P,Q)が発生したとき，規則１），規則２）の両方が適用可能であるが，この場合，規則２）の現状態をp1(P)，not[p2(Q)]，p4(Q)とみなすことで，規則の優先適用に従い，規則２）が適用される．

５．１．１．４　ｏｒ記述

異なる現状態で，同じイベントでかつ同じ次状態となる複数の規則に対して，一つの規則として現状態を併せて記述できる．これをｏｒ記述という．ｏｒ記述も現状態のみ使用でき，cond記述と併用はできない．

［記法］

ｏｒ記述　：　(p-string ¦ p-string)

注）p-string：＝現状態

［例］

規則１）　path(A,B)　onhook (A) :　idle (A).

規則２）　busy (A,B)　onhook (A) :　idle (A).

規則３）　(path (A,B) ¦ busy (A,B))　onhook (A) :　idle (A).

規則１）のpath(A,B)と規則２）のbusy (A,B)を併せて規則３）の(path (A,B) ¦ busy (A,B))として一つの規則で記述できる．

５．１．１．５　回数判定記述

カウンタ付状態記述要素が持っているカウンタ値と宣言した値の比較判定を規則中に記述するにはequal［カウンタ付状態記述要素］，not-equal［カウンタ付状態記述要素］を用いる．回数判定記述は現状態記述のみに使用でき，またｃｏｎｄ記述とは併用できるがｎｏｔ記述とは併用できない．

［記法］

回数判定記述：　equal[count-primitive-string]

not-equal[count-primitive-string]

注）count-primitive-string:=primitive（状態記述要素）

equa［カウンタ付状態記述要素］を記述した規則は，実状態にカウンタ付状態記述要素があり，そのカウンタ値が予め宣言した値と等しいときに適用される．not-equal［カウンタ付状態記述要素］を記述した規則は，実状態にカウンタ付状態記述要素があり，そのカウンタ値が予め宣言した値と等しくないときに適用される．

［例］　cs(A)がカウンタ付状態記述要素とする

s1(A,B),equal[cs(A)]　e(A,B):　s2(A).

この規則は実状態にs1(A,B)，cs(A)がありcs(A)のカウンタ値が宣言した値と同じとき適用される．

s1(A,B),not-equal[cs(A)]　e(A,B):　s2(A).

この規則は実状態にs1(A,B)，cs(A)がありcs(A)のカウンタ値が宣言した値と違うとき適用

される．

５．１．２　次状態

５．１．２．１　一般の次状態記述

次状態は，一般の状態記述要素に対しての記述は現状態と同様である．但し，現状態では許されているｃｏｎｄ記述，ｎｏｔ記述，及びｏｒ記述は，次状態では記述できない．

［記法］

次状態　：　primitive-string

［例］

dial-tone (A), path (A,B), m-cw (A)

dial (A,B) :　CW-ringing (A,B), path (A,B), m-cw (A).

上記規則の「 CW-ringing (A,B), path (A,B), m-cw (A) 」が次状態の記述である．

５．１．２．２　内部イベント記述

規則の適用時に，内部イベントを送信する動作の記述ができる．このとき送信する内部イベントは，次状態の中に，「＞内部イベント」として記述する．このときの第二引数は，ある特定の対象物を表わす識別子を記述する．

［記法］

次状態　：　>internal-event-name (Var, Var)

［例］

3wc (A), 3wc1(A,B), 3wc2 (A,C), path (A,B), path (A,C)

flash (A) :　path(A,B), >sig-onhook (A,C).

三者通話中に端末Ａのフラッシュにより，第３者として呼び出していた端末Ｃとの通話を切断し，元の端末Ｂとの二者通話に戻る時の規則であり，次状態path (A,B) に遷移する際に，端末Ｃにsig-onhookを送信することを意味している．

５．１．２．３　ｅｍｐｔｙ記述

次状態に状態記述要素の記述，かつ内部イベントの記述がない場合には，空白ではなくｅｍｐｔｙと記述する．

［記法］

次状態　：　empty

［例］

cw-ringing (A,B)　sig-onhook (B,A) :　empty.

現状態のcw-ringing (A,B)が，内部イベントsig-onhook (B,A)の受信により，次状態では現状態に関わる状態記述がなくなったことを意味している．

５．２　イベントの記述

５．２．１　ユーザイベントの記述

ユーザイベントの記述はイベント名の後ろに括弧で括り，第１引数は，イベントを生起させた識別子，第２引数は相手先の識別子を記述する．

［記法］

ユーザイベント：　name-string (Var) :　または　name-string (Var,Var) :

［例］

dial-tone (A), path(A,B), m-cw(A)

dial (A,B) : CW-ringing (A,B), path (A,B), path (A,B), m-cw (A).

上記規則の「dial (A,B) :」がイベントの記述である．

5．2．2　タイムオーバイベントの記述

タイムオーバイベントは，時間制限で宣言した状態記述要素の集合をイベントとして記述することになる．タイムオーバイベントは「timeover」の後に状態記述要素の並びをカッコで括り記述する．このとき，状態記述要素第１引数は異なる識別子でもよいが，識別子の関係が宣言した状態記述要素で辿れなければならない．（5.3.2　制約６参照）

［記法］

タイムオーバイベント：　timeover (primitive-string) :

［例］

Limited-Time-Primitives :

dial-tone(A)　　30sec

dial-tone (A)　timeover (dial-tone (A)) :　busy (A).

時間制限の状態記述要素として宣言されているdial-tone (A)が現状態のとき，３０秒以内に状態が変化しない場合，timeover (dial-tone (A))というイベントにより次状態に遷移する．

5．2．3　疑似イベントの記述

疑似イベントの記述も，タイムオーバイベント同様，状態記述要素の集合をイベントとして記述する．疑似イベントは，状態記述要素の集合を［　］で括り記述する．

［記法］

疑似イベント：　［primitive-string］：

［例］

m-CCBS (A,B), confirmation (A), m-CCBSed (B,A), idle(B)

[idle (B)] :　ringback (A,B), ringing (B,A).

これはＣＣＢＳの規則で，端末Ａが端末Ｂにダイヤルし，相手話中であった場合，ＣＣＢＳを要求し，そのまま待機していると，相手（端末Ｂ）が空き状態 idle (B)になった時直ちに相手呼び出しの状態になる．この［idl e(B)］が疑似イベントである．

5．2．4　内部イベントの記述

内部イベントの記述は，予め宣言されている内部イベントが使用できる．記法はユーザイベントと同様である．

［記法］

内部イベント：　internal-event-name((Var, Var)

［例］

path (A,B)　sig-onhook (B,A) : busy (A) .

ある端末Bがオンフックしたことにより，端末Bと関係のあるすべての端末Aに対してsig-onhookが送信され，上記規則が適用される．

5．3　規則の記述

5．3．1　規則の記述

規則は，現状態，イベント，次状態を記述する．現状態とイベント，イベントと次状態の間には，少なくとも1文字以上の空白が必要である．イベントの終わりにはコロンを，次状態記述の終には，ピリオドを記述する必要がある．また，サービス内の規則の記述順序性はなく，任意の順序で記述が可能である．

［記法］

規則　：　p-string　event-string :　next-string.
または　p-string　internal-event :　next-string.

注）p-string：＝現状態
　　next-string：＝次状態

［例］

idle (A) offhook (A) : dial-tone (A).
dial-tone (A), idle (B)　dial (A,B) :　ringback (A,B), ringing (B,A).

これはＰＯＴＳでの規則であるが，最初の規則は，端末Aが空状態 idle (A) で発呼するoffhook (A) と，ダイヤル受信待機状態 dial-tone (A) に遷移することを規定している．次の規則は，ダイヤル受信待機状態から，空状態の端末Bidle (B)にダイヤルdial (A,B)すると，端末Bは呼び出し音受信状態となり，端末Aは呼び返し音受信状態 ringback (A,B)に遷移することを規定している．

5．3．2　ＳＴＲを記述する上での制約

ＳＴＲの規則を記述する上で，いくつかの制約がある．

［制約1］

ＳＴＲの次状態記述で用いる識別子は，その同じ規則の現状態またはイベントの記述で用いる識別子でなければならない．

［制約2］

ユーザイベントの記述では，イベントが発生した対象物についての状態記述が現状態記述に含まれていなければならない．

［制約3］

ひとつの規則上では，現状態（＝状態記述要素の集合）とイベントとに記述する識別子の関係は，イベントの識別子を元に，1）その識別子を第1引数に持つ状態記述要素を辿り，2）辿れた状態記述要素に第2引数が有る場合は，その識別子を第1引数に持つ状態記述要素を辿り，現状態に記述した全ての識別子が辿れなければならない．

例1：　制約3を満足する場合

現状態：　状態1 (A, D)，状態2 (D)，状態3 (B, C)，状態4 (C)
イベント(A, B)

この例では，図５－１に示すように，イベントの識別子A，Bから現状態の全ての識別子が辿れている．

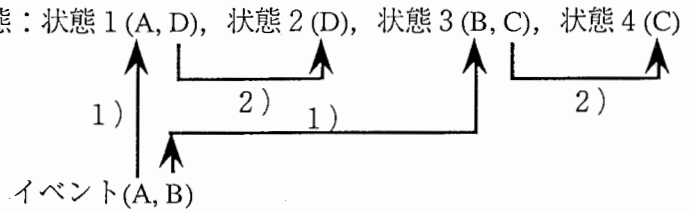


図５－１　制約３の満足例

例２：　制約３を満足しない場合

現状態：　状態１(A, D)，状態２(D)，状態３(B)，状態４(C)

イベント(A, B)

この例では，図５－２に示すようにイベントの識別子A，Bから現状態の「状態１(A, D)，状態２(D)，状態３(B)」は辿れるが，状態４(C)は辿れないので，制約３を満足しない．

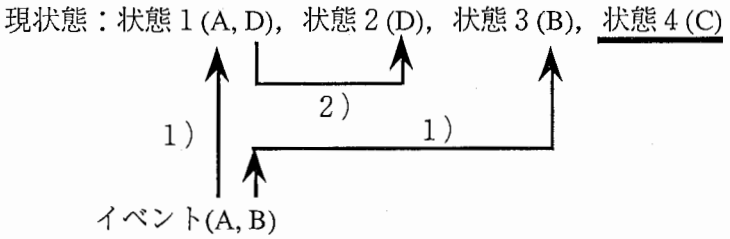


図５－２　制約３により構文エラー例

また，現状態記述にnot記述がある場合には，not記述を除いたものでこの制約３を満たし，かつnot記述のnotを外した状態記述要素に置き換えた現状態記述で制約３を満たさなければならない．

［制約４］

内部イベントによる動作記述では，記述した状態記述要素の全ての第１引数は同一でなければならない．

［制約５］

ある対象物の現状態記述と次状態記述には以下の制約がある．

(1)　ある対象物の状態記述要素がｎｏｔ記述のみとして規則の現状態記述にあり，次状態記述にない時は，規則適用による実状態の変化としてその対象の状態は現状態と同じになる．

(2)　ある対象物の状態記述要素がｎｏｔ記述以外として規則の現状態記述にあり，次状

態記述にない時は，ある対象物の該当の状態記述記述は無くなった状態となる．

(3) ある対象物の状態記述要素が規則の次状態記述にあり，現状態記述にない時は，構文誤りとなり，その対象物の現状態を記述しなければならない．

［制約６］

複数の対象物（状態記述要素の第１引数の識別子が違う場合）の記述に対する制限：

ある状態記述要素を基点として，他の状態記述要素との関係が以下の手順で辿れなければならない．

１）基点の状態記述要素の第１引数と同じ識別子を第１引数にもつ持つ状態記述要素を辿れる

２）基点の状態記述要素の第２引数と同じ識別子を第１引数に持つ状態記述要素を辿れる

３）辿れた状態記述要素の第２引数と同じ識別子を第１引数に持つ状態記述要素を辿れる

この操作により，全ての状態記述要素が辿れること．

図５－３に，この制限を満足する例を示し，図５－４にこの制限により構文エラーになる例を示す．

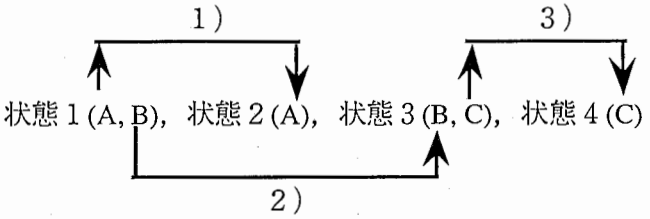


図５－３　制限満足例

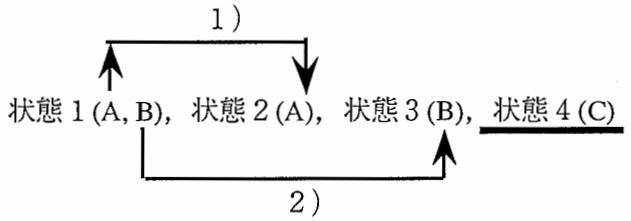


図５－４　構文エラー例

付録１：STR の BNF 記述

```
################################################
  BNF Description for STR-V2 (dated April 1994)
################################################

STR
   : STR_descriptions
   | Undefined_String STR_descriptions
   ;

Undefined_String
   : Strings
   ;

Strings
   : STRING Strings
   | STRING
   ;

STR_descriptions
   : STR_description
   | STR_description STR_descriptions
   ;

STR_description
   : Decls Rules
   | Rules
   ;

Decls
   : Decl Decls
   | Decl
   ;

Decl
   : Term_Var_declaration
   | User_Var_declaration
   | Primitive_declaration
   | Term_Initial_Primitive_declaration
   | User_Initial_Primitive_declaration
   | Event_declaration
   | Time_declaration
   | Count_declaration
   | Event_delivery
   | Macros
   | Inhibits_sets
   | Delivery_range
```

```
  | Internal_events_declaration
  ;

Term_Var_declaration
  : TERMINALS ':' type_list
  ;

User_Var_declaration
  : USERS ':' type_list
  ;

Primitive_declaration
  : PRIMITIVES ':' primitive_descriptions
  ;

primitive_descriptions
  : primitive_description
  | primitive_description ',' primitive_descriptions
  ;

primitive_description
  : name_string '(' type ')'
  | name_string '(' type ',' type ')'
  ;


primitive_strings
  : primitive_string
  | primitive_string ',' primitive_strings
  ;

primitive_string
  : name_string '(' Var ')'
  | name_string '(' Var ',' Var ')'
  ;

p_primitive_strings
  : p_primitive_string
  | p_primitive_string ',' p_primitive_strings
  ;

p_primitive_string
  : primitive_string
  | NOT '[' primitive_strings ']'
  ;

Term_Initial_Primitive_declaration
  : TERMINAL_INITIAL_PRIMITIVE ':' term_init_prim_string
```

```
  ;

term_init_prim_string
  : primitive_string
  ;

User_Initial_Primitive_declaration
  : USER_INITIAL_PRIMITIVE ':' user_init_prim_string
  ;

user_init_prim_string
  : primitive_string
  ;

Event_declaration
  : EVENTS ':' event_descriptions
  ;

event_descriptions
  : event_description
  | event_description ',' event_descriptions
  ;

event_description
  : name_string '(' type ')'
  | name_string '(' type ',' type ')'
  | '[' state_diffs ']'
  | TIMEOVER '(' primitive_strings ')'
  ;


event_string
  : name_string '(' Var ')'
  | name_string '(' Var ',' Var ')'
  | '[' state_diffs ']'
  | TIMEOVER '(' primitive_strings ')'
  ;

state_diffs
  : state_diff
  | state_diff ',' state_diffs
  ;

state_diff
  : primitive_string
  ;

Internal_events_declaration
```

```
  : INTERNAL_EVENTS ':' internal_event_descriptions
  ;

internal_event_descriptions
  : internal_event_description
  | internal_event_description ',' internal_event_descriptions
  ;

internal_event_description
  : internal_event_name '(' type ',' type ')'
  ;

internal_event_string
  : internal_event_name '(' Var ',' Var ')'
  ;

Time_declaration
  : LIMITED_TIME_PRIMITIVES ':' time_strings
  ;

time_strings
  : time_string
  | time_string time_strings
  ;

time_string
  : limited_time_state limited_time_sec
  ;

limited_time_state
  : primitive_strings
  ;

limited_time_sec
  : NUMBER SEC
  | Limited_time
  ;

Count_declaration
  : COUNT_PRIMITIVES ':' count_descriptions
  ;

count_descriptions
  : count_description
  | count_description count_descriptions
  ;

count_description
```

```
  : name_string '(' type ')' NUMBER
  | name_string '(' type ',' type ')' NUMBER
  ;

count_string
  : name_string '(' Var ')'
  | name_string '(' Var ',' Var ')'
  ;

Event_delivery
  : INTERNAL_EVENT_DELIVERATION ':' event_delivery_descriptions
  ;

event_delivery_descriptions
  : event_delivery_description
  | event_delivery_description event_delivery_descriptions
  ;

event_delivery_description
  : event_string EVENT_DELIV_AROW internal_event_string
  | event_string EVENT_DELIV_AROW internal_event_string '|' internal_event_string
  ;

Macros
  : MACRO_PRIMITIVES ':' macro_strings
  ;

macro_strings
  : macro_string
  | macro_string macro_strings
  ;

macro_string
  : Macro_primitive '=' '{' p_primitive_strings '}'
  | Macro_primitive '='
      '{' '(' Macro_or_primitives ')' '}'
  ;

Macro_primitive
  : Capital_string_name '(' Var ')'
  | Capital_string_name '(' Var ',' Var ')'
  ;

Macro_or_primitives
  : Macro_or_primitive
  | Macro_or_primitive '|' Macro_or_primitives
  | '(' Macro_or_primitives ')'
  ;
```

```
Macro_or_primitive
   : p_primitive_string
   ;

Inhibits_sets
   : INHIBITED_PRIMITIVE_SETS ':' inhibit_strings
   ;

inhibit_strings
   : inhibit_string
   | inhibit_string inhibit_strings
   ;

inhibit_string
   : '{' primitive_strings '}'
   | '{' primitive_strings '}' '(' inhibit_primitive_strings ')'
   ;

inhibit_primitive_strings
   : inhibit_primitive_string
   | inhibit_primitive_string ',' inhibit_primitive_strings
   ;

inhibit_primitive_string
   : name_string '(' '$' ')'
   | name_string '(' '$' ',' Var ')'
   ;

Delivery_range
   : DELIVERY_RANGE ':' delivery_strings
   ;

delivery_strings
   : delivery_string
   | delivery_string delivery_strings
   ;

delivery_string
   : RANGE '(' internal_event_string ':' primitive_strings ')' '='
         '{' '}'
   ;

type_list
   : type
   ;

Rules
```

```
  : RULES ':' rule_strings
  ;

rule_strings
  : rule_string
  | rule_string rule_strings
  ;

rule_string
  : p_strings rule_event_string ':' next_strings '.'
  ;

p_strings
  : p_string
  | p_string ',' p_strings
  ;

p_string
  : p_primitive_string
  | COND ':' cond_string
  | COND ':' '(' cond_strings ')'
  | '(' or_strings ')'
  | Macro_primitive
  | EQUAL '[' count_string ']'
  | NOT_EQUAL '[' count_string ']'
  ;

/* or_strings がひとつの or_string だけからなることは容認してもよい */
or_strings
  : or_string
  | or_string '|' or_strings
  | '(' or_strings ')'
  ;

or_string
  : p_primitive_string
  | COND ':' cond_string
  | COND ':' '(' cond_strings ')'
  | Macro_primitive
  | EQUAL '[' count_string ']'
  | NOT_EQUAL '[' count_string ']'
  ;

cond_strings
  : cond_string ',' cond_strings
  | cond_string
  ;
```

```
cond_string
   : p_primitive_string
   | Macro_primitive
   | EQUAL '[' count_string ']'
   | NOT_EQUAL '[' count_string ']'
   ;

rule_event_string
   : event_string
   ;

next_strings
   : next_string
   | next_string ',' next_strings
   | EMPTY
   ;

next_string
   : primitive_string
   | Macro_primitive        /* NOT を含むマクロプリミティブの場合はエラー */
   | '>' internal_event_string
   ;

type : Capital_char
   ;

Var
   : Capital_head_string
   | Capital_char
   ;

Capital_string_name
   : Capital_head_string
   ;

name_string
   : STRING
   ;

internal_event_name
   : STRING
   ;
```

付録２：通信サービスの STR 記述例

```
##############################################
##   POTS Service                           ##
##    $Id: pots.str,v 1.5 1994/08/23 02:39:02 kawa Exp $
##############################################

Primitives:
     idle(A),dial-tone(A),busy(A),busy-dial(A,B),
     path(A,B),hangup(A),ringing(B,A),ringback(A,B)

Terminal-Initial-Primitive:
     idle(A)

Events:
     offhook(A),onhook(A),dial(A,B),wrong-dial(A),
     timeover(dial-tone(A)),timeover(busy(A)),timeover(busy-dial(A,B))


Limited-Time-Primitives:
     dial-tone(A)          10sec
     busy(A)              20sec
     busy-dial(A,B)        20sec

Internal-Events:
     sig-onhook(A,B)

Internal-Event-Deliveration:
     onhook(A) --> sig-onhook(A,B)

Macro-Primitives:
     Calling(A,B) = {ringback(A,B),ringing(B,A)}
     Talk(A,B)    = {path(A,B),path(B,A)}
     Busy(A,B)    = {(busy(A)|busy-dial(A,B))}

Delivery-Range:
     range(sig-onhook(A,B):busy-dial(A,B)) ={}

Rules:
idle(A) offhook(A): dial-tone(A).
Calling(A,B) offhook(B): Talk(A,B).
path(A,B) onhook(A): idle(A).
Busy(A,B) onhook(A): idle(A).
dial-tone(A) onhook(A): idle(A).
ringback(A,B) onhook(A): idle(A).
hangup(A) onhook(A): idle(A).
path(A,B) sig-onhook(B,A): busy(A).
ringing(A,B) sig-onhook(B,A): idle(A).
```

dial-tone(A),idle(B) dial(A,B): Calling(A,B).
dial-tone(A) dial(A,A): busy(A).
dial-tone(A),not[idle(B)] dial(A,B): busy-dial(A,B).
dial-tone(A) wrong-dial(A): busy(A).
dial-tone(A) timeover(dial-tone(A)): busy(A).
busy(A) timeover(busy(A)): hangup(A).
busy-dial(A,B) timeover(busy-dial(A,B)): hangup(A).

##
CW Service
$Id: cw.str,v 1.6 1994/08/23 02:38:52 kawa Exp $
##

Primitives:
idle(A),dial-tone(A),busy(A),busy-dial(A,B),
path(A,B),hangup(A),ringing(B,A),ringback(A,B),
cw(A),m-cw(A),cw-ringing(A,B),path-passive(A,B),waiting(A),cw-config(A,B)

Terminal-Initial-Primitive:
idle(A)

Events:
offhook(A),onhook(A),dial(A,B),wrong-dial(A),
timeover(dial-tone(A)),timeover(busy(A)),timeover(busy-dial(A,B)),
cw(A),flash(A),
timeover(Hold(A,B),waiting(A),cw(A))

Internal-Events:
sig-onhook(A,B)

Limited-Time-Primitives:
dial-tone(A) 10sec
busy(A) 20sec
busy-dial(A,B) 20sec
Hold(A,B),waiting(A),cw(A) 1sec

Internal-Event-Deliveration:
onhook(A) --> sig-onhook(A,B)

Macro-Primitives:
Calling(A,B) = {ringback(A,B),ringing(B,A)}
Talk(A,B) = {path(A,B),path(B,A)}
Busy(A,B) = {(busy(A)|busy-dial(A,B))}
CW-calling(A,B)={ringback(A,B),cw-ringing(B,A)}
Hold(A,B) = {path-passive(A,B),path-passive(B,A)}

Inhibited-Primitive-Sets:

{cw-ringing(A,B),cw(A)} (busy($))
{cw-ringing(A,B),cw-ringing(A,C)} (busy($))
{path-passive(A,B),busy-dial(A,B)}(busy($))

Delivery-Range:
range(sig-onhook(A,B):busy-dial(A,B)) ={}
range(sig-onhook(A,B):cw-ringing(A,B)) = {}

Rules:
idle(A) offhook(A): dial-tone(A).
dial-tone(A),idle(B) dial(A,B): Calling(A,B).
dial-tone(A) dial(A,A): busy(A).
dial-tone(A),not[idle(B)] dial(A,B): busy-dial(A,B).
Calling(A,B) offhook(B): Talk(A,B).
path(A,B) onhook(A): idle(A).
path(A,B) sig-onhook(B,A): busy(A).
Busy(A,B) onhook(A): idle(A).
dial-tone(A) onhook(A): idle(A).
ringing(A,B) sig-onhook(B,A): idle(A).
ringback(A,B) onhook(A): idle(A).
hangup(A) onhook(A): idle(A).
dial-tone(A) wrong-dial(A): busy(A).
dial-tone(A) timeover(dial-tone(A)): busy(A).
busy(A) timeover(busy(A)): hangup(A).
busy-dial(A,B) timeover(busy-dial(A,B)): hangup(A).

path(A,B),dial-tone(C),cond:m-cw(A) dial(C,A): CW-calling(C,A),path(A,B),cw-config(A,B),cw-config(A,C).
cond:path(A,B),dial-tone(C),cond:m-cw(A),cond:cw-config(B,A) dial(C,A): busy-dial(C,A).
CW-calling(C,A),Talk(A,B) flash(A): cw(A),Talk(A,C),Hold(A,B).
Talk(A,C),Hold(A,B),cond:cw(A) flash(A): Talk(A,B),Hold(A,C).
cond:cw(A),path(A,C),cond:path-passive(A,B),cw-config(A,B),cw-config(A,C) onhook(A): ringing(A,B).
cond:path-passive(A,B) sig-onhook(B,A): empty.
path-passive(A,B) onhook(A): idle(A).
path-passive(A,B),cw(A),cond:path(A,C),cw-config(A,B),cw-config(A,C) sig-onhook(B,A): empty.
path(A,C),cond:path-passive(A,B),cond:cw(A),cw-config(A,B),cw-config(A,C) sig-onhook(C,A): waiting(A).

cw-ringing(A,C),path(A,B),cw-config(A,B),cw-config(A,C) onhook(A): ringing(A,C).
cw-ringing(A,B),cw-config(A,B),cw-config(A,C) sig-onhook(B,A): empty.
cond:cw-ringing(A,B),path(A,C),cw-config(A,B),cw-config(A,C) sig-onhook(C,A): waiting(A).

ringing(A,B),Hold(A,B),cw(A) offhook(A): Talk(A,B).
cond:ringing(A,B),cond:path-passive(A,B) onhook(A): empty.
ringing(A,B),path-passive(A,B),cw(A) sig-onhook(B,A): idle(A).

Hold(A,B),waiting(A),cw(A) flash(A): Talk(A,B).
cond:path-passive(A,B),waiting(A),cond:cw(A) onhook(A): ringing(A,B).
path-passive(A,B),waiting(A),cw(A) sig-onhook(B,A): busy(A).

Hold(A,B),waiting(A),cw(A) timeover(Hold(A,B),waiting(A),cw(A)): Talk(A,B).

waiting(A),CW-calling(B,A) flash(A): Talk(A,B).
waiting(A),cw-ringing(A,B) onhook(A): ringing(A,B).
waiting(A),cw-ringing(A,B) sig-onhook(B,A): busy(A).

cond:idle(A) cw(A): m-cw(A).
cond:idle(A),m-cw(A) cw(A): empty.

##
3WC Service
Created by Waka,28,Jul,1994
$Id: 3wc.str,v 1.9 1994/09/07 07:16:44 waka Exp $
##

Primitives:
idle(A),dial-tone(A),busy(A),busy-dial(A,B),
path(A,B),hangup(A),ringing(B,A),ringback(A,B),
recall-dial-tone(A),waiting(A),
m-3wc(A),3wc(A),3wc1(A,B),3wc2(A,B),path-passive(A,B)

Terminal-Initial-Primitive:
idle(A)

Events:
offhook(A),onhook(A),dial(A,B),wrong-dial(A),
timeover(dial-tone(A)),timeover(busy(A)),timeover(busy-dial(A,B)),
flash(A),3wc(A),
timeover(Hold(A,B),waiting(A),3wc(A))

Internal-Events:
sig-onhook(A,B)

Limited-Time-Primitives:
dial-tone(A) 10sec
busy(A) 10sec
busy-dial(A,B) 10sec
Hold(A,B),waiting(A),3wc(A) 1sec

Internal-Event-Deliveration:
onhook(A) --> sig-onhook(A,B)

Inhibited-Primitive-Sets:
{3wc(A),3wc(A)}
{path-passive(A,B),busy-dial(A,B)}(busy($))

Macro-Primitives:

Calling(A,B) = {ringback(A,B),ringing(B,A)}
Talk(A,B) = {path(A,B),path(B,A)}
Busy(A,B) = {(busy(A)|busy-dial(A,B))}
Hold(A,B) = {path-passive(A,B),path-passive(B,A)}

Delivery-Range:
range(sig-onhook(A,B):busy-dial(A,B)) ={}

Rules:
Talk(A,B),cond:m-3wc(A) flash(A): Hold(A,B),recall-dial-tone(A),3wc(A).
cond:Talk(A,B),cond:m-3wc(A),cond:3wc(B),not[3wc(A)] flash(A): empty.
cond:Talk(A,B),cond:m-3wc(A),cond:3wc(B),cond:Talk(A,C),not[3wc(A)] flash(A): empty.

recall-dial-tone(A),idle(B) dial(A,B): Calling(A,B).
recall-dial-tone(A),not[idle(B)] dial(A,B): busy-dial(A,B).
recall-dial-tone(A) dial(A,A): busy(A).
recall-dial-tone(A) onhook(A): idle(A).
Hold(A,B),(recall-dial-tone(A)|hangup(A)),3wc(A) flash(A): Talk(A,B).
cond:path-passive(A,B),(path(A,C)|ringback(A,C)|Busy(A,C)|recall-dial-tone(A)|hangup(A)),
cond:3wc(A)
onhook(A): ringing(A,B).
cond:path-passive(A,B) sig-onhook(B,A): empty.
path-passive(A,B) onhook(A): idle(A).
path-passive(A,B),3wc(A) sig-onhook(B,A): empty.

cond:Hold(A,B),cond:Calling(A,C) flash(A): 3wc1(A,B),3wc2(A,C),ringback(B,C).

Hold(A,B),Calling(A,C),ringback(B,C) offhook(C): Talk(A,B),Talk(A,C),Talk(B,C).
3wc1(A,B),3wc2(A,C),3wc(A),Hold(A,B),Calling(A,C),ringback(B,C) flash(A): Talk(A,B),idle(C).
cond:path-passive(A,B),ringback(A,C),cond:3wc(A),3wc1(A,B),3wc2(A,C) onhook(A): ringing(A,B).
cond:path-passive(A,B),(ringback(A,C)|Busy(A,C)|hangup(A)),not[3wc(A)]
sig-onhook(B,A): empty.

path-passive(A,B),(ringback(A,C)|Busy(A,C)|hangup(A)) onhook(A): idle(A).
3wc1(A,B),3wc2(A,C),path-passive(A,B),3wc(A) sig-onhook(B,A): empty.

Hold(A,B),cond:Talk(A,C),cond:m-3wc(A),cond:3wc(A) flash(A): 3wc1(A,B),3wc2(A,C),Talk(A,B),Talk(B,C).
path(A,C),cond:path-passive(A,B),cond:3wc(A) sig-onhook(C,A): waiting(A).
Hold(A,B),waiting(A),3wc(A) flash(A): Talk(A,B).
Hold(A,B),waiting(A),3wc(A) timeover(Hold(A,B),waiting(A),3wc(A)): Talk(A,B).
cond:path-passive(A,B),waiting(A),cond:3wc(A) onhook(A): ringing(A,B).
path-passive(A,B),waiting(A),3wc(A) sig-onhook(B,A): busy(A).

3wc1(A,B),3wc2(A,C),cond:Talk(A,B),Talk(A,C),Talk(B,C),3wc(A),cond:m-3wc(A) flash(A): busy(C).
3wc1(A,B),3wc2(A,C),path(A,C),path(A,B),Talk(B,C),3wc(A) onhook(A): idle(A).
path(A,B),path(A,C) onhook(A): idle(A).
(3wc1(A,B),3wc2(A,C)|3wc1(A,C),3wc2(A,B)),path(A,B),3wc(A),cond:path(A,C)
sig-onhook(B,A): empty.

path(A,B),cond:path(A,C) sig-onhook(B,A): empty.

cond:Hold(A,B),cond:busy-dial(A,C),cond:3wc(A) flash(A): busy-dial(B,C).
cond:Hold(A,B),cond:busy(A),cond:3wc(A) flash(A): busy(B).
Hold(A,B),busy-dial(A,C),busy-dial(B,C),3wc(A) flash(A): Talk(A,B).
Hold(A,B),busy(A),busy(B),3wc(A) flash(A): Talk(A,B).
Hold(A,B),hangup(A),Busy(B,C),3wc(A) flash(A): Talk(A,B).
Hold(A,B),hangup(A),hangup(B),3wc(A) flash(A): Talk(A,B).
Hold(A,B),Busy(A,C),hangup(B),3wc(A) flash(A): Talk(A,B).

Hold(A,B),ringing(A,B),3wc(A) offhook(A): Talk(A,B).
cond:path-passive(A,B),cond:ringing(A,B) onhook(A): empty.
path-passive(A,B),ringing(A,B),3wc(A) sig-onhook(B,A): idle(A).

cond:idle(A) 3wc(A): m-3wc(A).
cond:idle(A),m-3wc(A) 3wc(A): empty.

idle(A) offhook(A): dial-tone(A).
Calling(A,B) offhook(B): Talk(A,B).
path(A,B) onhook(A): idle(A).
Busy(A,B) onhook(A): idle(A).
dial-tone(A) onhook(A): idle(A).
ringback(A,B) onhook(A): idle(A).
hangup(A) onhook(A): idle(A).
path(A,B) sig-onhook(B,A): busy(A).
ringing(A,B) sig-onhook(B,A): idle(A).
dial-tone(A),idle(B) dial(A,B): Calling(A,B).
dial-tone(A) dial(A,A): busy(A).
dial-tone(A),not[idle(B)] dial(A,B): busy-dial(A,B).
dial-tone(A) wrong-dial(A): busy(A).
dial-tone(A) timeover(dial-tone(A)): busy(A).
busy(A) timeover(busy(A)): hangup(A).
busy-dial(A,B) timeover(busy-dial(A,B)): hangup(A).