解　説

SDL 言語の特質と処理系の現状と動向†

若　原　　恭††

1. まえがき

SDL (Specification and Description Language) は，通信システムの機能に関する仕様や動作を曖昧なく簡潔に記述するため，国際電信電話諮問委員会 CCITT で開発し勧告として標準化した[1)-4)] 形式記述技法 FDT (Formal Description Technique) である．SDL の開発は 1972 年に開始し，まず 1976 年に SDL の原形ともいえる最初の勧告が作成された．その後，4 年ごとに言語機能の拡張が進められ，1988 年には形式記述能力が高い最新版勧告が制定された．この最新版 SDL は長い年月をかけて多数の関係者の意見をもとにして作成されたため，機能が豊富過ぎる，複雑であるなどの指摘を受けることもあるが，完成度は高いものと考えられており，今後機能が大幅に追加・変更される可能性は小さいといわれている．

SDL は，CCITT における各種通信方式やプロトコルの記述に実際に使用されているだけでなく，通信運用会社や通信機器製造会社における通信システムの開発・運用・保守にも広く使用されている．さらに最近では国際海事衛星機構 INMARSAT や国際電気通信衛星機構 INTELSAT などにおいても採用されはじめており，最近の調査によると，世界で 5,000 人以上のユーザがいる[3)]．一方，SDL の言語仕様の研究と併行して，SDL による正確な仕様や動作の記述の作成，SDL により記述された仕様からプログラムやテストデータの生成を効率よく支援する技術などの処理技術についての研究も近年積極的に進められており，その成果をもとに多くの支援ツールが実際に開発され使用されている．本解説では，このような SDL とその処理系の概要を紹介するが，詳細な言語機能やその形式的定義については CCITT 勧告とその付録文書[1)]を参照されたい．また，SDL を実際に使う場合に有用な入門書やガイドライン文書が出版され始めており[5)]，CCITT および国際標準化機構 ISO からも発行される予定である[6)]．

2. SDL の特徴と基本概念

SDL は，交換機や通信端末など通信システムに要求される機能仕様の記述，およびそのようなシステムの実際の動作の記述の両者に適用できるものとして開発され，以下に示す特徴を備えている．

(1)　システムのモデル

SDL では通信処理を実行する単位をプロセスと呼び，一般に記述対象のシステムを複数のプロセスから構成する．各プロセスは，拡張有限状態機械としてモデル化され，他プロセスとの間で信号を送受することが可能で，信号を入力することにより，その入力情報ともとの状態に応じて必要な処理を実行し次の状態に遷移する．この信号送受において，各プロセスは非同期的に動作する．つまり，各プロセスは任意の時点で他プロセスに信号を送出することが可能でその後必要な処理を継続して実行することができる．このため，各プロセスは容量が十分大きな FIFO (First In First Out) 型の信号受信バッファを 1 個もっており，他プロセスから受ける信号はすべてこのバッファを経由して入力する．

このようなモデル化の結果，SDL は通信システムの動作や仕様，特にその通信プロトコル部分を明示的に表現できる．また，一般に SDL で記述された仕様はそれを実現するシステムの構造との整合性が高い．

(2)　記述の構成と方法

システムの記述は，①システムを構成する機能単位であるブロックやプロセスの間の静的な信号送受関係によりシステムの枠組みと構造を表現する静的構造記述と，②各プロセスの動的な動きを表す動作記述とから構成する．

システム記述のうち，静的構造部分は，ブロック，プロセスやそれらの間の関係を宣言的に記述する．ま

† Features of SDL and Recent Trends of Support Techniques for SDL by Yasushi WAKAHARA (KDD Kamifukuoka R&D Laboratories).

†† KDD 上福岡研究所

た，プロセス動作部分は，型を用いてその動作のひな型を定義するが，各型においては状態ごとに各信号入力に対する動作処理を手続き的に記述する．各型によって定義された動作をとるプロセスの実体はインスタンスとして必要な個数だけ生成することができる．このような型とそのインスタンスを総合的に組み合わせることによりシステム全体の動作を記述する．たとえば，交換機では複数の通信の交換処理を同時に実現する必要があるが，これら各処理の機能仕様は一般に同一であるため，その仕様は一つのプロセス型定義で与えることができる．

これらの記述に使用するデータとその操作の記述には，自然言語を用いた非形式的な記述と，実現手段に依存せずその性質のみによって規定することが可能な抽象データ型を利用した形式的記述の両者があり，それらを混在させて使用することもできる．

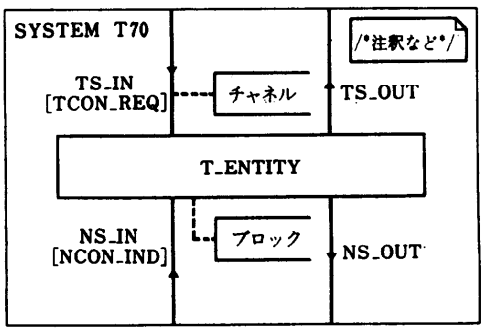
SDL のシンタクスには，図形式の SDL/GR (Graphical Representation) とテキスト形式の SDL/PR (Phrase Representation) とがある．両者は基本的には等価であるが，一般に GR による記述は理解しやすいといわれており[1)-4)]，実際に使用される SDL の多くは GR で，SDL 支援システムの多くが GR をサポートしている．

3. SDL の言語機能

3.1 階層化による静的構造の記述

複雑で大規模なシステムを簡明に記述するため，SDL では機能を段階的に詳しく記述していく階層的記述法を採る．具体的には，システムを構成する機能単位をブロック (BLOCK) と呼び，一般に，システムは一つまたは複数個のブロックから構成する．各ブロックはさらにいくつかの(サブ)ブロックから構成することもできるが，最下位レベルのブロックは必ず1個以上のプロセスをもつ．ブロック同士の間で信号を送受する通信路をチャネル (CHANNEL) と呼び，プロセス間の通信路をシグナルルート(SIGNALROUTE)と呼ぶ．チャネルはさらに詳細化して記述することが可能であり，そこにはブロックや（サブ）チャネルを含むことができる．チャネルやシグナルルートは，一方向性でもよいし双方向性でもよい．記述対象としない部分はまとめて環境 (ENV) と呼び，環境との間にチャネルやシグナルルートを設定することもできる．プロセスは最初から存在するものとして定義することもできるし，システム動作中任意の時点で動的に生成することもできる．また，プロセスはすべての処理を終了すると消滅する．

静的構造の記述で使用される主な記述要素を表-1 (a)と(b)に示す．図-1 は OSI 基本参照モデルのトランスポート層プロトコル（クラス0）を処理するシステム T70 の静的構造を表す GR 表現[6)]の一部とそれに対応する PR 表現である．このシステムは1個のブロック T_ENTITY と4本のチャネルとから構成される．この図で，チャネル名の後の[ ]内の文字列はそのチャネルで授受する信号の名前を表す．図-2 はブロック T_ENTITY の内部構成を表す図[6)]で，このブロックは2個のプロセス T_MANAGER と TPM と9本のシグナルルートとから構成され，プロセス TPM はプロセス T_MANAGER により生成される．この図で，プロセス名の後の( )内の二つの数字は，それぞれ，システム動作開始時に存在するプロセス・インスタンス数とシステム動作中に存在するプロセス・インスタンス数の上限値（無指定のときは∞）を表す．たとえばプロセス T_MANAGER はシステム動作中常に1個存在する．また，シグナルルート名の後の[ ]内の文字列はそのルートで授受する信号の名



(a) SDL/GR 表現

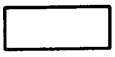
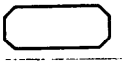
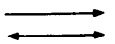
```
SYSTEM T70;
  /*注釈など*/
  CHANNEL TS_IN FROM ENV TO T_ENTITY
    WITH TCON_REQ;
  ENDCHANNEL TS_IN;
  CHANNEL TS_OUT FROM T_ENTITY TO ENV;
  ENDCHANNEL TS_OUT;
  CHANNEL NS_IN FROM ENV TO T_ENTITY
    WITH NCON_IND;
  ENDCHANNEL NS_IN;
  CHANNEL NS_OUT FROM T_ENTITY TO ENV;
  ENDCHANNEL NS_OUT;
  BLOCK T_ENTITY REFERENCED;
ENDSYSTEM T70;
```

(b) SDL/PR 表現

図-1 システム静的構造記述の例

表-1 SDL の記述要素

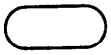
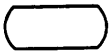
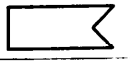
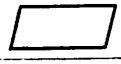
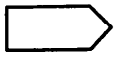
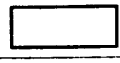
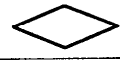
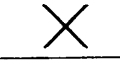
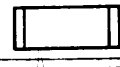
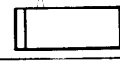
(a) 静的構造記述に使われる要素

記述要素	SDL/GR表現	SDL/PR表現
システム		SYSTEM ENDSYSTEM
ブロック		BLOCK ENDBLOCK
プロセス		PROCESS ENDPROCESS
チャネル		CHANNEL ENDCHANNEL
シグナルルート		SIGNALROUTE
プロセス生成		なし

(b) 静的構造記述とプロセス動作記述の両者に使われる要素

記述要素	SDL/GR表現	SDL/PR表現
テキスト枠		なし
テキスト拡張		なし
コメント		COMMENT
ノート	/* ～ */	/* ～ */

(c) プロセス動作記述に使われる要素

記述要素	SDL/GR表現	SDL/PR表現
開始		START
状態/次状態		STATE/ NEXTSTATE
信号入力		INPUT
信号入力 ガード条件		PROVIDED
セーブ		SAVE
信号出力		OUTPUT
タスク処理		TASK
判断分岐		DECISION ENDDECISION
停止		STOP
コネクタ		JOIN a; b:
プロセス生成		CREATE
手続き呼出し		CALL
マクロ		MACRO ENDMACRO
手続き戻り		RETURN

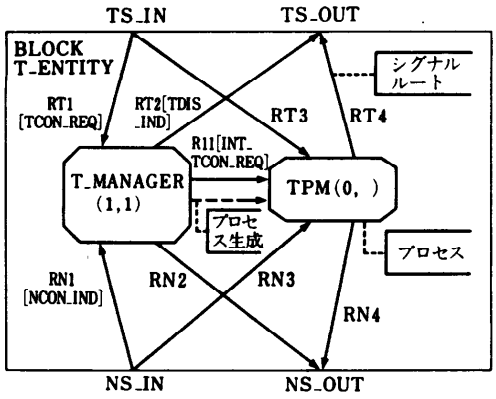
前を表す．図-1(a)，図-2(a)のような GR 表現をブロック相互関係図(BID : Block Interaction Diagram)と呼ぶ．

3.2 プロセス動作の記述

プロセスは，データを記憶したり処理することが可能なように拡張された有限状態機械である．この機械の処理の流れは，処理が中断したいくつかの安定状態に分けることができ，処理を実行することによりある状態から別の状態へ遷移する．遷移のきっかけは信号の受信入力であるが，信号入力にガード条件を付けたり，信号の受信があったときそれをすぐには入力せずバッファにセーブしておき別の信号の到着を待つようにすることもできる．また，受信入力を陽に記述していない信号を受信したときはその信号を無視して棄却しもとの状態に戻る．

状態の遷移にともなって種々の処理を実行することができるが，この処理には，信号の出力，条件判断，タスク処理，同一ブロック内の他のプロセスの生成などが含まれる．ここで，タスク処理とは変数に対する各種の操作や非形式的に書かれたテキストで表されたオペレーションを表す．SDL/PR ではこれら各処理はセミコロン；によって区切られた一つのステートメントにより記述され，原則として記述された順序に従って実行される．SDL/GR では，これらの記述に使用する各シンボルはそれらを連結する矢印または直線（フローライン）に従って実行される．

プロセス動作の記述に使用される記述要素を，



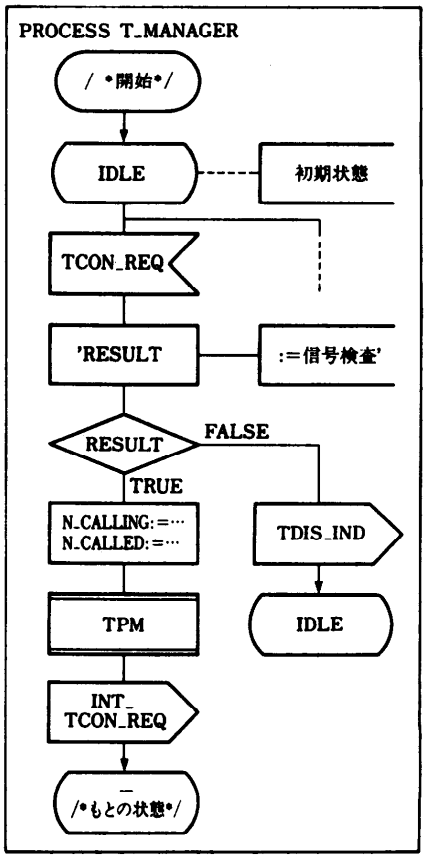
(a) SDL/GR 表現

```
BLOCK T_ENTITY;
  CONNECT TS_IN AND RT1, RT3;
  CONNECT TS_OUT AND RT2, RT4;
  CONNECT NS_IN AND RN1, RN3;
  CONNECT NS_OUT AND RN2, RN4;
  SIGNALROUTE RT1 FROM ENV TO
    T_MANAGER WITH TCON_REQ;
  SIGNALROUTE RT2 FROM T_MANAGER TO
    ENV WITH TDIS_IND;
  SIGNALROUTE RT3 FROM ENV TO TPM;
  SIGNALROUTE RT4 FROM TPM TO ENV;
  SIGNALROUTE R11 FROM T_MANAGER TO
    TPM WITH INT_TCON_REQ;
  SIGNALROUTE RN1 FROM ENV TO
    T_MANAGER WITH NCON_IND;
  SIGNALROUTE RN2 FROM T_MANAGER TO ENV;
  SIGNALROUTE RN3 FROM ENV TO TPM;
  SIGNALROUTE RN4 FROM TPM TO ENV;
  PROCESS T_MANAGER(1,1) REFERENCED;
  PROCESS TPM(0, ) REFERENCED;
ENDBLOCK T_ENTITY;
```

(b) SDL/PR 表現

図-2 ブロック静的構造記述の例

表-1(b)と(c)に示す．図-3 は図-2 のプロセス T_MANAGER の動作記述[6]の一部で，その動作は次のとおりである．このプロセスは，最初に状態 IDLE に移り，信号 TCON_REQ を受信すると，TCON_REQ の正当性を検査しその結果誤りがあれば (RESULT=FALSE) 信号 TDIS_IND を送出してもとの状態に戻る．検査の結果正しければ (RESULT=TRUE) 変数 N_CALLING と N_CALLED に適当な値を設定した後プロセス TPM を生成し信号 INT_TCON_REQ を送出してもとの状態に戻る．ここで，状態名—は直前の状態名を表す．また，信号の入出力における相手プロセス名は信号出力に対してのみ記述できるが，信号入力，また相手プロセス名が記述されない信号出力の相手プロセス名は静的構造図のチャネルやシグナル



(a) SDL/GR 表現

```
PROCESS T_MANAGER;
  START/*開始*/;
    NEXTSTATE IDLE;
  STATE IDLE COMMENT 初期状態;
    INPUT TCON_REQ;
      TASK 'RESULT:=信号検査';
      DECISION RESULT;
      TRUE:
        TASK N_CALLING:=…;
        TASK N_CALLED:=…;
        CREATE TPM;
        OUTPUT
          INT_TCON_REQ;
        NEXTSTATE -/*もとの状態*/;
      FALSE:
        OUTPUT
          TDIS_IND;
        NEXTSTATE IDLE;
      ENDDECISION;
ENDPROCESS T_MANAGER;
```

(b) SDL/PR 表現

図-3 プロセス動作仕様の例

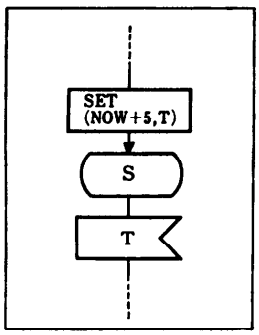
ルートに付けられた信号名を参照して決められる．SDL/GR によりプロセス動作記述を表す図-3（a）のような図をプロセス・グラフと呼ぶ．

一般に，プロセス動作記述においては，‘と’で囲まれた部分により処理内容を非形式的に記述することができ，この記法はタスクや判断分岐などに適用できる．さらに，SDL/GR では，各シンボルに対するテキストはそのシンボルの中に記述するが，書ききれない場合は表-1（b）に示すテキスト拡張シンボルを用いて記述する．図-3（a）にこれらの例を示す．

SDL には，タイミング処理などの記述のためタイマ機能がある．タイマは SET 処理により起動され，SET 時に指定された時間をその後経過するとタイマ信号を発生してそのプロセスに出力し，動作を停止する．以降このタイマ信号は通常の信号と同様に扱われる．ただし，タイマ信号を発生する前に RESET 処理が実行されるとタイマはその動作を中止する．図-4 にタイマの使用例を示す．図-4 では，処理 SET (NOW +5, T) の実行によりタイマが起動し，その時点から5単位時間後にタイマ信号Tを発生する．つまりこのプロセスの状態Sは5単位時間だけ継続する．

プロセスの記述の複雑化を回避するための手段として手続き (PROCEDURE) とマクロ (MACRO) がある．これらは，通常のプログラミング言語がもつ手続きやマクロと同様の機能をもつ．手続きで使える言語要素はプロセスとまったく同一である．また，マクロはプロセスだけでなく，ブロックや手続きを含めることもできる．

規模が大きいシステムの場合その記述は通常の紙面一枚に入りきらなくなる．このような場合，その記述を紙面の大きさに合わせて分割する必要があるが，SDL/GR では分割された各部分を連結させるためコネクタを使用する．コネクタには，その識別のための名前と連結先を含むページの番号を付ける．



(a) SDL/GR 表現

```
SET(NOW+5,T);
NEXTSTATE S;

STATE S;
INPUT T;
```

(b) SDL/PR 表現

図-4 タイマの使用例

3.3 データの記述

SDL のデータ型は，ISO で標準化された形式仕様記述言語 LOTOS に採用された ACTONE と同等の抽象データ型であり，その内部構造や実現方法を規定せずに性質のみによって定義する．具体的には，この定義は，①データのとりうる値 (LITERAL 宣言)，②これらの値に対し適用できる演算子名とその入出力パラメータのデータ型 (OPERATOR 宣言)，および③データと演算子から構成される等式によって演算子の意味を定義する公理 (AXIOM 宣言) とから構成される．例としてブール型の定義の一部を図-5 に示す．この図で，ブール型変数の値は True または False のいずれかである．ここではブール型変数に対する演算子はNOT と XOR のみを定義しており，前者は入力出力ともに1個のブール型変数であり，後者は入力が2個，出力が1個のブール型変数である．NOT と XOR の定義は AXIOMS 以下の等式で与えられる．

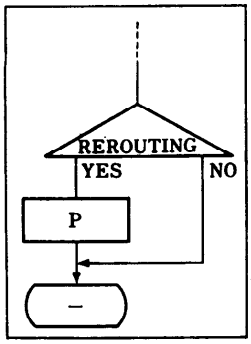
Integer, Character, Array など実際の通信システムの記述で多用する基本的なデータ型は，CCITT 勧告書にあらかじめ predefined データ型として与えられている．なお，データ型の記述は常にテキスト形式であり，GR と PR に共通である．

3.4 その他の記述機能

通信システムにおいては互いに類似の仕様や動作をもつものが少なくない．このような場合共通部分を一つにまとめて記述し，共通でない部分については適用するシステムの構成や運用条件に応じて使い分けるよう記述することができる．このような機能はオプションと呼ばれ，静的構造記述，プロセス動作記述の両者に対して適用できる．図-6 にプロセス動作記述におけるオプションの使用例を示す．図-6 は，このプロ

```
NEWTYPE Boolean;
  LITERALS True, False;
  OPERATORS
    "NOT" : Boolean          -> Boolean;
    "XOR" : Boolean, Boolean -> Boolean;
  AXIOMS
    "NOT"(True)          == False;
    "NOT"(False)         == True;
    "XOR"(True, True)    == False;
    "XOR"(True, False)   == True;
    "XOR"(False, True)   == True;
    "XOR"(False, False)  == False;
ENDNEWTYPE Boolean;
```

図-5 抽象データ型の例




```
ALTERNATIVE
REROUTING;
  YES:
    TASK P;
    NEXTSTATE −;
  NO:
    NEXTSTATE −;
ENDALTERNATIVE;
```

(a) SDL/GR表現　　(b) SDL/PR表現

図-6 オプションの使用例

セスがリルーティングを行う交換機に適用される場合はタスクPを実行し，そうでなければタスクPを実行しないことを表す．

一般に仕様や動作の記述では，プログラムなどと同様に種々の注釈を与えることが必要となる．SDL では，注釈の記述には2通りの方法がある．第一の方法は，コメントと呼ばれるもので他の記述要素に対し付加するものである．第二の方法はノートと呼ばれるもので，記述要素の間を含め任意の場所に記述することができる．これらはやはり静的構造記述とプロセス動作記述の両者に使用できる．コメントとノートの記述法は表-1(b) のとおりで，図-1 と図-3 にそれらの使用例を示す．なお，これらの図から分かるように，SDL/GRでは，図形シンボルに属さない各種宣言文やそれらのノートなどのテキストはすべて表-1(b)に示すテキスト枠の中に記述する．

4. SDL の処理系

4.1 概　要

一般に，通信システムの仕様は，それに基づいてプログラムを作成するなど通信システムを開発・保守するためのベースとなる．また，通信システムの動作記述は，完成したシステムを理解し機能の追加や変更を加えるうえでの基礎資料となる．したがって，SDL によるこれらの記述は，長い間の使用に耐えるものであることが必要であり，特に，記述内容が正しいことと，記述内容が容易に理解できることが要求される．

SDL の言語設計に際しては，このような要求条件を満たすように考慮が払われた．しかし，言語機能だけでこれらの要求条件を満たす記述を作成することは容易でない．近年，正しいプログラムを効率よく作成するためにプログラミング環境が重要であると認識されているが，これと同様上記要求条件を満たすためには SDL による記述を支援する環境が重要となる．このような支援環境には，計算機入力・編集，検証，完全化など種々の処理機能が含まれる．さらに，仕様をもとにしてプログラムを開発する作業の効率向上を図るため，仕様からプログラムへの変換やテストデータ生成機能も含められることが多い[7]．

これらの支援機能は原理的には必ずしも記述言語に依存するとは限らないが，実際の支援ツールは言語に依存したものとなる．また，このような支援機能に関する研究はプログラミング支援機能に比較して歴史が浅く，まだ研究途中の段階にあるが，SDL に関する支援ツールについては，SDL が広く実用されていることからも分かるように，比較的多くの研究開発が進められている．これらのツールは，関連したものをまとめた支援システムとして開発されることが普通である[7)~18)]が，ここでは，次節以降機能別にそれらの例を紹介する．

4.2 計算機入力と編集

SDL による仕様や動作の記述を計算機処理するためにまず最初に必要となる処理機能は，計算機への入力と編集である．この実現には，SDL/PR に対してはワードプロセッサなどの通常のエディタを利用することも可能であるため PR 専用のエディタの研究開発は比較的少なく，SDL エディタといえば SDL/GR を対象としたグラフィックエディタを意味することが多い．

一般に，SDL/GR エディタは通常のテキストエディタに比較すると，PR で使用するセミコロンという特別な記号を入力することなくシンボルごとの区切りが自然に規定できる，フローラインにより他と連結されておらない到達不可能なシンボルをただちに発見できるなど，文法誤りが少なくより正しい記述を容易に作成できるという特徴がある．特に構造化エディタの場合は，エディタが文法知識をもっており，入力されたシンボルが文法誤りとなる場合はただちに誤りであることを出力できる．最近ではこのような SDL エディタはマルチウィンドウ表示機構やマウス機構を備えたワークステーションで実現されることが多い．

SDL グラフィックエディタの対象とする図形式にはブロック相互関係図，プロセス・グラフに加え，ブロックの階層構成を示す木図や，プロセス相互で授受する信号を時系列的に表した信号シーケンス図など必ずしも CCITT の正式な勧告に含まれてはいないが

関連ある図形式を同時に扱えるものもある[9),11),16)]. 一般に，これらのグラフィックエディタでは，各図形式で使用するシンボルをアイコンとしてパレット状に並べ，そのアイコンと入力場所をマウスで指示することにより所要のシンボルを入力していく方式が多い．また，各種の図形式は必ずしも独立ではなく，たとえば木図に含まれたブロックやプロセスの一つをマウスで指定することにより，そのブロックの相互関係図やプロセス・グラフをただちに画面に表示させるなど他の図形式のアクセス手段として利用することが多い[11),16)]．またブロック相互関係図や信号シーケンス図においてはシンボルの大きさを固定することができないため，シンボルの拡大縮小機能が具備されている[16)]．

SDL の特徴を利用し，入力・編集操作を単純にするための工夫がいくつか試みられている．たとえば，プロセス・グラフにおいて次に入力すべきシンボルの位置が直前に入力したシンボルの直下の位置であればそのマニュアル指定を省略できるエディタがある[12)]．このときフローラインを自動的に付与することも一部可能となる．また，一般に SDL エディタでは図形シンボルと文字入力のための二つのモードがあるが，図形の入力できる領域とテキストの入力できる領域を明確に分け，これにより入力すべき場所に応じて入力モードを自動的に切替え，マニュアルによるモード切替えを不要にした方式もある[12)]．

エディタに付属する機能としては，いったん作成した SDL 記述の検索機能，ドキュメンテーションのための印刷出力などがある．SDL 記述の検索には，通常のファイル検索と同様ファイル名をもとにして行う方式が多いが，使用されているシンボルの状態名，信号名，タスク名などをキーとする方式もある．また，最近では，図形シンボル間の相互関係をキーとして指定する方式も検討されている[19)]．一方，SDL 記述の印刷出力においては一般に大きな SDL/GR 形式の図面をページ単位に分割する必要がある．図面の分割印刷を自動化したシステムでは，3. で紹介したページ相互間の関係を示すコネクタを自動的に付加する．

なお，このようなエディタにおいてはシンボルの形状や大きさがユーザの好みに依存することがあるため，これらの変更を可能とするカスタマイズ機能をもつエディタもある[11)]．これにより，ユーザの属す組織に応じた図形形状を定義することができる．

4.3 仕様の検証

一般に人手で作成した仕様は誤りを含むことが多いため，SDL で書かれた仕様に含まれる誤りを検出する検証技術の研究開発が活発である．

検証技術を論じるためには，まず検出対象となる誤りを明確にする必要があるが，ここでは，SDL 文法誤りを除外して議論する．このような範囲で，実際に研究開発されている検証技術が対象とする誤りを表-2 に示す．表-2 では，誤りを判定するための基準があらかじめ明確に定義できるか否かに応じて，誤りを論理誤りと意味誤りとに分けた．

論理誤りはその判定基準が明確に定義されるため原理的には機械的な検出が可能であり，実用的な論理誤り検出ツールも多く開発されている．たとえば，主な不完全誤りは，着目するデータや信号に対する宣言や操作を，実行順序に従ってすべて検査していくことによって検出できる[20)]．また，信号入出力の対応不一致

表-2 仕様誤り

仕様誤り	誤りの定義	具体例
論理誤り	誤り判定基準があらかじめ明確に定義でき多くのシステムに共通なもの	不完全誤り ●宣言されたが使用されないデータ，信号，ブロック，プロセス． ●宣言されずに使用されたデータ，信号，ブロック，プロセス． ●値を設定されないで参照されたデータ，信号． ●行先の不明な信号出力． ●行先のないコネクタ 内部矛盾 ●信号の入力・出力の対応不一致． ●手続き呼び出しにおけるパラメータ不一致． ●チャネル・シグナルルートにおける信号の不一致． ●信号，データの二重宣言． ●状態の次にある，信号入力・セーブ以外の要素． 実行不可能誤り ●デッドロック． ●アクセス不可能な部分． ●実行条件のもれ． ●分岐条件のもれ．
意味誤り	誤り判定基準が個々のシステムに固有であらかじめ明確には定義できないもの	もとの要求に対する不一致．

やデッドロックなどのプロトコルエラーは，各プロセス相互での信号送受に基づくシステムのダイナミックな動作を模擬して検査するプロトコル検証手法により検出できる[21]．このプロトコル検証では，特にプロセスが多い場合，検査すべき状態数などが急激に増大していくためこれを回避するための方法が重要となり，実際そのような手法がいくつか提案されている[22]．また，実行不可能誤りのうち実行条件や分岐条件のもれについては，条件の記述を論理的に検査することが必要であるが，条件の記述が単純な論理式であれば容易である．しかし，条件の記述に複雑な関数などを使用している場合には機械的な検査は困難となる．

一方，意味誤りの検証についてはまだ技術課題が多い．これまでに検討されてきた意味検証法は，①もとの要求をなんらかの形式で具体的に表明として表現し，その表明が成立するか否かを検査する方法[9),23]，②仕様をインタプリティブにまたは仕様をプログラムに変換して実行しその実行結果を検査する方法[24)~26]，③なんらかの着目点に従って仕様を編集加工して部分抽出しレビューを容易にする方法[11),16]に分類できよう．しかしながら，もとの要求が必ずしも明確でない，仕様を実行させるために多くのデータ類を準備する必要があり手間がかかる，一般に仕様の規模が大きく複雑であるため部分抽出のための着目点が必ずしも明らかでないなどの理由から，意味検証は一般に困難であり，意味検証技術は必ずしも実用性が十分達成できた段階にあるとはいえない．今後の研究の発展が期待されるテーマといえよう．

4.4 仕様からプログラムの生成

SDL は，基本的には，3. で述べたとおり，手続き型言語であり，状態の記述や抽象データ型部分を除くと，比較的通常のプログラミング言語に近いため，SDL の記述からそれを実現するプログラムを生成することは比較的容易である．SDL からプログラムを自動的に生成するツールは，トランスレータ，コードシンセサイザ，プログラムシミュレータなど種々の呼び名があるが，SDL に対する処理系としてはきわめて研究開発例が多い[8)~14),16),27)~31]．得られるプログラムの実行速度や保守性のよさ，後述のマニュアル作業の容易さなどが重要な評価尺度となろう．変換して得られるプログラムの記述言語はCが多いが，その他 Pascal, Ada や CHILL などがある．これらのツールにおけるプログラム生成の基本的な実現法は次のとおりである．

SDL による記述は，静的構造記述とプロセス動作記述とから構成されるが，ダイナミックな動作はすべてプロセス動作記述によって規定されるため，SDL からプログラムの生成はプロセス動作記述が主体となる．静的構造記述の参照が必要となるのは，主としてプロセス間で送受する相手プロセスを識別するためである．一方，プロセス動作記述をプログラムへ変換する場合，状態機械の実現法が一つの鍵となる．この実現には，①機械自体を一つのソフトウェア・モジュールとして作成し状態遷移に関する情報をデータとして扱う方法と，②プログラミング言語の switch 文や case 文を使って状態と信号入力を場合分けすることにより状態遷移情報を直接プログラムとして記述する方法とがある．一般に前者に比較して後者の方法は実行効率はよいが言語への依存性が大きく移植性に難がある[28]．

SDL からプログラムへの変換に際しては，もともと SDL で記述されていない部分，抽象データ型，および，SDL で形式的に記述されない以下の部分については通常マニュアルで変換する．

①SDL の非形式的手法により記述されている部分
②他のソフトウェア部分とのインタフェース部分
③プログラムを実行するためのオペレーティングシステムとのインタフェース部分
④ハードウェアなどリソースに依存する部分

一方，このようなプログラム生成ツールのなかには，SDL で記述されない処理内容を部品としてあらかじめ知識ベースに蓄えておき，SDL 記述ではその処理内容を参照することによって効率よくプログラムを生成する方式がある[13),16),31]．たとえば，SDL における状態，入力，出力，タスク，分岐判断などの各要素に対応するプログラム部品を，下位レベルのルーチンとして用意しておき，変換に際してはこれら部品に置換していく．このような部品化の方法により，いったん作成したプログラムの再利用を可能とし，その結果，信頼性の高いプログラムを効率よく生産していくことが容易となる．

4.5 その他の支援技術と処理系

（1） PR/GR 変換

3. において，SDL の図形式表現 GR とテキスト表現 PR とは原理的には等価であると述べた．実際，両者間の相互変換を実行するツールが広く作成され使用されている．しかしながら，実際には以下に示すような問題がある．

① マクロ

SDL/GR と PR は意味的には等価であるが，構文上では必ずしも1対1には対応しない．たとえば，判断分岐は，表-1（c）に示すとおり，GR では一つのシンボルで表現されるが，PR では DECISION と ENDDECISION の2行に分かれる．したがって，図-7 に示すように，DECISION を含むが ENDDECISION を含まない PR ステートメント部分を一つのマクロにすると，これを正しく GR に変換することはできない．このように，正しい PR/GR 相互変換を実現するためには，GR と PR とが1対 N 対応の関係にある部分をマクロ化する場合，それらをまとめてマクロ化する必要がある．

② プロセス生成シンボル

GR のブロック相互関係図では，あるプロセスとそれから生成されるプロセスとは点線の矢印で両者を結合することによりその生成関係が表される．しかしこの図に対応する PR ではそのような生成関係の情報をもたない．したがって，ブロック相互関係図のみを PR に変換するとプロセスの生成関係の情報が失われる．しかし，PR 表現において，たとえばノート機能を利用してプロセス生成関係の情報を蓄えておくことにより正しい PR/GR 相互変換を実現することができる．

一般に，SDL の各図形シンボルやステートメントのレイアウトの自由度が高いため，たとえば GR をいったん PR に変換し，さらにその PR を GR に逆変換しても最初の GR とは一致しない．しかし，図-8 に示すように，もう一度 PR に変換してさらに GR に逆変換すると最初に変換して得た GR と一致するのが普通である．

```
PROCESS T_MANAGER;
  START;
    NEXTSTATE IDLE;
  STATE IDLE;
    INPUT TCON_REQ;
      TASK 'RESULT:=信号検査';
      DECISION RESULT;
      TRUE
        TASK N_CALLING:=…;
        TASK N_CALLED:=…;
        CREATE TPM;
        OUTPUT
          INT_TCON_REQ;
        NEXTSTATE −/*もとの状態*/;
      FALSE
        OUTPUT
          INT_TDIS_IND;
        NEXTSTATE IDLE;
      ENDDECISION;
ENDPROCESS T_MANAGER;
```

マクロ化

図-7 PR から GR へ変換できないマクロの例

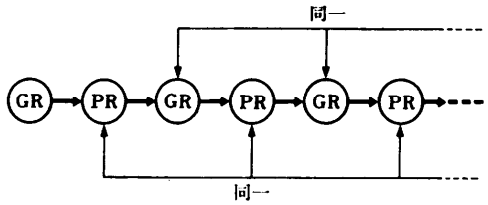


図-8 PR/GR 変換における変換結果の一致

（2） 完全化処理

完全化処理は，SDL による記述に含まれる不足部分を（半）自動的に補うための処理[13),32)~34)]であり，エラー訂正法ともいえる．このような処理は一般に容易ではなく，ここでは二つの完全化手法を以下に紹介する．

① 信号入力（INPUT）の抜け

これは **4.3** で述べたプロトコルエラーの一つであり，プロトコル検証法により信号入力の記述が抜けている状態を検出できるので，信号入力をそこに補うことは容易にできる．しかし，その信号入力を実行した後どの状態に遷移させるかについては一意に定まらない．そこで，ユーザとのインタラクションによってこれを決定する方法[32)]，あらかじめプロトコルエラーが発生しないような訂正パタンをいくつか用意しておき，そのパタンに従って自動的に訂正する方法[33)]などが検討されている．後者の場合には，必ずしももとの要求どおり訂正できるとは限らないため，ユーザが確認する必要がある．

② 例外処理

一般に新しい通信サービス要求を具体化していく場合，その中心は正常処理シーケンス部であり，残りの異常処理部は作成漏れとなることが少なくない．このような異常処理の例としては，タイムアウト処理，サービス利用者の途中放棄などの例外操作などがある．このような異常処理とそれを必要とする状態に関する情報をルールとして知識ベースにあらかじめ蓄えておき，マニュアルで作成した仕様にこの知識ベースを参照することにより例外処理を自動的に付加する手法がある[13)]．

（3） テストデータ生成

通信システムを仕様に基づいて作成した場合，そのプロダクトが所要の条件を満たしているか否か検査する必要がある．このような検査法の一つは，検査対象に外部からテストデータを与え，そのテストデータに対して期待される動作と実際の動作結果とを比較検査することである．特にテストデータがプロトコルデー

タでかつ被テスト対象の内部状態を観測しない場合，このような検査法はコンフォーマンス試験と呼ばれている．このような試験では，それに用いるテストデータを効率よく仕様から生成する方法が重要となる．よいテストデータの目安は，①できるかぎり多くの誤りを検出できること，②テストデータを構成する信号シーケンスが少なく，かつ長さが短いことなどである．SDL が有限状態機械をベースとしているので，原理的にはこのようなテストデータは状態機械の同定法を適用することにより生成可能となる．しかしながら，一般に①を満たすテストデータの量はきわめて多くなるため，それらすべてに対して検査することは不可能に近く，なんらかの指標により実際に使用するテストデータを選択抽出する必要がある．この選択抽出の方法についての検討は少なくない[35)-38)]が，まだ決定的に優れた方法は分かっていない．

（4） 簡明化処理

通常 SDL で書かれる仕様や動作の記述はマニュアルで作成されるが，規模が大きい場合は多人数で作成することとなる．一般に同一の意味をもつ仕様の SDL 表現法は複数通りあるため，このような大規模な仕様では表現が不統一になり，分かりにくくなることがある．このような場合には，もっとも分かりやすい簡明な表現に統一しておくことが望ましい．与えられた SDL 記述をこのような簡明な表現に変換する簡明化処理の検討はまだ多くないが，以下のような方法が検討されている[39),40)]．

①レイアウトの標準化
②状態などの等価な記述要素の一括化
③判断分岐要素の順序入替えによる表現単純化
④go to 要素（JOIN）の削除
⑤実行不可能な要素の削除

5. あ と が き

本稿で紹介した最新版 SDL に対しては，プロセス自体の階層的記述機能，プロセス間通信の隠蔽化機能，非決定的遷移機能などが欠けているとの意見もある[41),42)]が，大勢としては言語そのものについての研究は一段落したといえよう．実際，その標準を定めた CCITT では，今研究期（1989〜1992）に入ってからは，オブジェクト指向などの新しい技術の採用についても審議しているが，むしろ SDL の普及を図るための活動や，勧告書の完全化に向けた作業を中心に行っている．このように SDL は一応の完成をみた言語とみなすことができ，現在はその実用期に入ったといえ[43),44)]，実際，冒頭で述べた通り，SDL は広く使用されはじめている．ただし，実際の SDL 記述では，たとえば抽象データ型などの高度な機能は使用せず，通常 SDL のもつ機能の一部のみを使用している．一方，SDL に関する各種ツールの研究開発も活発であり，商品となって販売されているものも少なくない．このような商品化は，従来ヨーロッパが中心であったが，最近はアメリカや日本においてもいくつかの例が見られるようになってきた．

しかしながら，SDL のみで通信システムのすべてをうまく記述できるとは限らず，他の手法を採用したほうがよい場合もある．たとえば，パケットデータの構造，プロセス相互間で送受する信号の流れ，処理能力や動的パフォーマンスなどは，SDL ではうまく表現できないので，表やシーケンスチャートなど他の手法が実際には必要となる．

本文では触れなかったが，SDL には第2の図形式表現 PE (Pictorial Element) がある．これは，GR における各状態シンボルの中に，その状態における各種リソースの使用状況をアイコンのようなシンボルで表現する記法である．この記法では，二つの連続する状態間でリソース使用状況に差があれば，そのようなリソース使用状況の変更を及ぼす処理をその状態間の遷移で実行することを表す．このように，PE は SDL において宣言的な処理記述法を一部採用したものとみなすことができる．ただし，リソースの使用状況は通信システムのアーキテクチャや対象とする通信サービスに依存するため必ずしもリソース使用状況表示法の標準化は容易ではない．実際 CCITT 勧告で規定されているリソース使用状況表示用シンボルは電話用交換機に対応する PE のみである．

今後は，通信システムの動作や仕様の抽象度をより高度化するため，それが提供するサービスを中心として記述していく方法も考えられる[45)]．この場合サービス記述のための技法が新たに必要となるだけでなく，サービス仕様から SDL で記述されたシステム仕様への変換技術が重要となる．また，システム・プロダクトの管理からは仕様とそれを実現するプログラムは常に対応したものとする必要がある．このためには，たとえばプログラムの修正が仕様にただちに反映できるようにすることが必要であり，SDL で記述された仕様をプログラムから抽出し作成する技術の研究が重要になっていくものと考えられる．

参考文献

1) CCITT 勧告 Z. 100, Melbourne (Nov. 1988).
2) 若原, 角田, 伊藤, 長谷川：通信システム仕様記述言語 SDL, 国際通信の研究, No. 137, pp. 88-98 (1988, 7).
3) Belina, F. and Hogrefe, D.: The CCITT-Specification and Description Language SDL, Computer Networks and ISDN Systems, Vol. 16, pp. 311-341 (1988/89).
4) 丸山：ソフトウェア関連の標準化動向, 電子情報通信学会誌, Vol. 72, No. 5, pp. 564-567 (1989).
5) Saracco, R., Smith, J. R. W. and Reed, R.: Telecommunications Systems Engineering using SDL, North Holland, Amsterdam (1989).
6) CCITT Contribution COM-X-R 29 (1988) also ISO/IEC/JTC 1 Proposed Draft Technical Report (Nov. 1988): Guidelines for the Application of Estelle, LOTOS and SDL.
7) 伊藤, 市川：通信分野における自動プログラミング, 情報処理, Vol. 28, No. 10, pp. 1405-1411 (1987).
8) Vefsnmo, E.: DASOM-An SDL Tool, in Saracco, R. and Tilanus, P. A. J.: SDL '87: State of the Art and Future Trends, pp. 35-42, North Holland, Amsterdam (1987).
9) Kossman, H.: An Integrated Set of Tools for Software Design Based on SDL, ibid., pp. 147-155, North Holland, Amsterdam (1987).
10) Jackson, L. N., Cheng, K. E., Choong, T. S., Pascoe, R. S. V. and Chain, G. J.: MELBA at the Age of Eight: An Automatic Code Generation System, ibid., pp. 371-381, North Holland, Amsterdam (1987).
11) 小山田：SDL に基づく交換ソフトの設計支援, 電子情報通信学会・交換・通信ソフトウェア仕様記述言語の標準と技術展望専門講習会資料, pp. 67-76 (1988, 6).
12) 増井：SDL 支援システムの適用, 同上, pp. 77-86 (1988, 6).
13) 山崎：知識ベースを用いた交換ソフト仕様処理, 同上, pp. 97-105 (1988, 6).
14) Atlevi, M.: SDT-The SDL Design Tool, in Turner, K. J.: Formal Description Techniques, pp. 55-60, North Holland, Amsterdam (1988).
15) 宗森, 水野：SDL グラフィックエディタの設計と製作, 情報処理学会論文誌, Vol. 29, No. 7, pp. 676-685 (1988).
16) Wakahara, Y., Kakuda, Y., Ito, A. and Utsunomiya, E.: ESCORT: An Environment for Specifying Communication Requirements, IEEE Software, Vol. 6, No. 2, pp. 38-43 (1989).
17) Encontre, V.: GEODE: An Industrial Environment for Designing Real Time Distributed Systems in SDL, in Færgemand, O. and Marques, M. M.: SDL '89: The Language at Work, pp. 105-115, North Holland, Amsterdam (1989).
18) Kikuchi, N., Shigeta, Y., Miyake, K., Tanaka, W. and Nabeta, M.: An Integrated System Development Method and Support System Based on SDL and C++, ibid., pp. 135-144, North Holland, Amsterdam (1989).
19) 伊藤, 若原：通信ソフトウェア仕様用図式検索手法, 電子情報通信学会春季全国大会, B-328 (1989).
20) Wakahara, Y. and Kakuda, Y.: Verification of Requirements Specifications for Telecommunications Software by the Investigation of Specification Execution Sequences, Proc. IEEE Globecom '87, pp. 661-666 (1987).
21) West, C. H.: General Technique for Communication Protocol Validation, IBM J. Res. Dev., Vol. 22, No. 4, pp. 393-404 (1978).
22) Holtzmann, G. J. and Patti, J.: Validating SDL Specifications: an Experiment, Proc. of 9th International Workshop on Protocol Specification, Testing and Verification (1989).
23) 角田, 若原：時相述語論理を用いたプロトコル自動意味検証, 電子情報通信学会技報, FTS 89-14, pp. 25-32 (1989).
24) Wakahara, Y. and Ito, A.: A Prototyping System for Telecommunications Software Based on Abstract Execution of Requirements Specifications, Computer Networks and ISDN Systems, Vol. 13, No. 2, pp. 119-128 (1987) also in Saracco, R. and Tilanus, P. A. J.: SDL '87: State of the Art and Future Trends, pp. 315-326, North Holland, Amsterdam (1987).
25) Unruh, E.: SCAN, an Expert System for the Analysis of SDL Specifications, in Saracco, R. and Tilanus, P. A. J.: SDL '87: State of the Art and Future Trends, pp. 137-146, North Holland, Amsterdam (1987).
26) 宗森, 田中, 佐藤, 勝山, 水野：SDL ビジュアルウォークスルーシミュレータ, 情報処理学会ソフトウェア工学研究会資料, 59-3 (1988).
27) Jackson, L. N., Fidge, C. J., Pascoe, R. S. V. and Gerrand, P. H.: Computer-Aided Program Generation from System Specifications, Proc. of ISS 84, pp. 33 A 4. 1-7 (1984).
28) Johansen, U. and Vefsnmo, E.: Automatic Program Generation of SDL Specification: Principles and Solutions, in Saracco, R. and Tilanus, P. A. J.: SDL '87: State of the Art and Future Trends, pp. 349-359, North Holland, Amsterdam (1987).
29) Karlsson, J. and Mansson, L.: Using SDL as Specification and Design Language and Ada as Implementation Language, in Saracco, R. and

Tilanus, P. A. J.: SDL '87: State of the Art and Future Trends, pp. 339-348, North Holland, Amsterdam (1987).
30) 長谷川, 野村: SDL とCを組み合わせた通信プログラム仕様の記述法およびその処理系, 電子情報通信学会春季全国大会, B 329 (1989).
31) 田中, 佐藤, 水野: プログラム部品利用による通信ソフトウェア開発支援, 情報処理学会マルチメディア通信と分散処理研究会資料, 42-3 (1989).
32) Zafiropuro, P., West, C. H. et al.: Towards Analyzing and Synthesizing Protocols, IEEE Trans. Commun, Vol. COM-28 (1980).
33) Kakuda, Y. and Wakahara, Y.: Component Based Synthesis of Protocols for an Unlimited Number of Processes, Proc. Compsac '87, pp. 721-730 (1987).
34) 田倉, 市川: 要求仕様に基づくプロトコル仕様の完全化について, 情報処理学会ソフトウェア工学研究会資料, 61-1 (1988, 7).
35) Chow, T. S.: Testing Software Design Modeled by Finite State Machine, IEEE Trans, Soft. Eng., Vol. SE 4, No. 3, pp. 178-187 (1978).
36) 加藤, 鈴木: オートマトンモデルに基づく OSI トランスポート・プロトコルの製品検証, 情報処理学会分散処理システム研究会 (1984, 11).
37) 佐藤, 宗森, 井手口, 水野: 有限オートマトンに基づくシステムの試験系列自動生成手法の提案, 電子情報通信学会論文誌, Vol. J 72-B-I, No. 3, pp. 183-192 (1989).
38) Hogrefe, D. and Brömstrup: TESDL: Experience with Generating Test Cases from SDL Specifications, in Færgemand, O. and Marques, M. M.: SDL '89: The Language at Work, pp. 267-279, North Holland, Amsterdam (1989).
39) 若原: ソフトウェア仕様における決定木表現の一最適化, 電子情報通信学会論文誌, Vol. J 71-D, No. 9, pp. 1887-1890 (1988).
40) Wakahara, Y., Ito, A., Utsunomiya, E. and Nitta, F.: Simplification of Requirements Specifications in SDL, in Færgemand, O. and Marques, M. M.: SDL '89: The Language at Work, pp. 189-198, North Holland, Amsterdam (1989).
41) Orava, F.: Formal Semantics of SDL Specifications, Proc. of 8th International Workshop on Protocol Specification, Testing and Verification (1988).
42) Hogrefe, D.: Nondeterminism and SDL, in Turner, K. J.: Formal Description Techniques, pp. 157-167, North Holland, Amsterdam (1988).
43) Belina, F., Hogrefe, D. and Trigila, S.: Modelling OSI in SDL, ibid., pp. 135-142, North Holland, Amsterdam (1988).
44) 宗森, 水野: 国際標準仕様記述言語 SDL を用いたオフィス情報システム記述支援環境の実現, オフィスオートメーション, Vol. 10, No. 1, pp. 66-73 (1989).
45) Ichikawa, H., Itoh, M. and Shibasaki, M.: Protocol-oriented Service Specifications and their Transformation into CCITT Specification and Description Language, Trans. of IECE of Japan, Vol. E 69, No. 4, pp. 524-535 (1986).

(平成元年10月2日受付)