LSI の動作記述からの知識獲得について†

西田豊明†† 川村 正†† 堂下修司††

専門分野のドキュメントからの知識獲得に関する研究は，自然言語理解における重要な応用分野であるが，これまで本格的な研究はあまりなされていない．そこで，本論文ではそのような研究の第一歩として，LSI やバスの動作を記述した自然言語と図式からなるテキストを解析して，形式的な仕様記述を生成するシステムについて述べる．本システムは，個々の動作の間の因果関係・時間関係によるネットワーク構造を中間表現として用い，自然言語解析，動作推論の二段階で処理を行う．自然言語解析においては，解析を見通し良く体系的に行うために，我々が従来行ってきた機械翻訳システムで用いた方法を改良し，作用的形式を用いた中間言語とそのオブジェクト指向の解釈システムを採用した．また，動作推論部では，単に情報抽出を行って仕様記述を生成するだけでなく，新情報が与えられたときにそれまで部分的に構成されていた仕様記述の一般化と個別化を自動的に行う知識獲得の機能を組み込んだ．このようなシステムのプロトタイプを汎用計算機上の LISP システム (UTILISP) により作成した．このプロトタイプを用いて実験を行い，本システムが典型的な LSI の動作記述から使用記述を生成できることを確認した．

1. はじめに

本研究の動機と目標は次の二点である．

a）専門分野のドキュメントからの知識獲得は，自然言語理解の重要な応用分野である．しかし，これまで主として行われてきた研究は，関係情報など静的な情報を抽出することに焦点があてられており，ここで取り上げるような，ハードウェアの動作記述など，動的な内容のテキストからの情報抽出に関する研究は，二，三の例を除いてほとんどなかった．また，単に情報抽出を行うだけではなく，帰納的推論によって，得られた知識を一般化することが必要である．例えば，ハードウェア・マニュアルでは，タイムチャートなどによって典型例を示すだけのことが多いので，システムが一般化する能力を持たないと，字づら上に書かれていることしか情報が得られないことになり，人間に比較して著しく能力が劣ることになる．

b）言語表現と対象世界の対応づけを体系的かつ柔軟に行うことは重要である．我々が以前行っていたモンテギュー文法に基づく機械翻訳システムでは，体系的な対応づけは可能であったが柔軟性に欠ける面があり，これを解決することが残された課題であった．

以上のような問題に対する第一歩として，我々は，LSI やバスの動作記述の解析を行うシステムを作成した．このシステムでは，オブジェクト指向の手法を用いて柔軟かつ体系的な自然言語解析を行っている．また，獲得した知識を一般化する機能を組み込むことにより，典型例からより一般的なモデルが得られるようになった．

2. システムの概要

2.1 入　力

本システムの入力は，自然言語テキスト (英文) と，タイムチャートである*．タイムチャートは，現在は付録に示すような機械的なコーディング規則により，人間が記号化してシステムに与えている．

2.2 出　力

本システムの出力は，記述されている LSI のプロトコルの形式的な記述である (仕様記述とよぶ)．ここで，対象としている LSI やバスは，論理的には並行動作する動作主 (agent) とみなすことができるが，本システムでは，これを有限状態オートマトン記述で表した．

2.3 処理過程

本システムは，自然言語解析部と動作推論部から構成される．システムの構成を**図1**に示す．

自然言語解析部は，入力として与えられた自然言語テキストを解析し，中間表現として依存格構造を生成する．ここで，処理を見通し良く行うため，作用的形式を用いた中間言語とそのオブジェクト指向の解釈システムを用いる．次に，依存格構造より LSI の動作に関する情報を抽出して，LSI の個々の動作の因果関

† Knowledge Acquisition from Specifications of LSI Actions by TOYOAKI NISHIDA, TADASHI KAWAMURA and SHUJI DOSHITA (Department of Information Science Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学教室

* 本システムの入力は主に文献 4)，5) に現れる記述や図に基づいている．

係グラフを生成する．この因果関係グラフは，動作推論部に送られる．

動作推論部は，因果関係グラフをトラバースすることにより仕様記述を生成する．新情報が与えられたときは，それまでに構成されていた仕様記述の一般化と個別化を自動的に行う．

入力がタイムチャートで与えられた場合は，人手によって記号化して因果関係グラフを得，これを動作推論部へ送る．

3. 自然言語解析部

本論文の目的の一つは，自然言語解析を見通し良く，体系的に行うことができるような処理の枠組をつくることである．これに関しては，我々が従来行っていたモンテギュー文法を用いた機械翻訳システム[3]の手法を改良し，これを用いた．ただし，あいまい性解消の問題は本論文の範囲外であると考えたので，浅いレベルの意味情報を用いて解決する．

3.1 作用的形式を用いた中間言語 (PAL)

以前我々が開発した機械翻訳システムでは，中間表現としてモンテギュー文法の内包論理式に基づく論理形式を用いていた．本システムでは，これをさらに柔軟なものに改良するため，作用的形式を用いた中間言語 (PAL: Purely Applicative Language) を用いている．PAL は，モンテギュー文法の高階内包論理からラムダ式を除いて単純化したものであり，すべての表現が $x(y)$ のように作用的形式で書かれた言語である．ここで，x の位置にくるものを作用子 (functor)，y の位置にくるものを引数 (argument) とよぶ．

自然言語解析における PAL の構成法を次に示す．

- 原則として，解析木における修飾句 (modifier) を作用子とし，被修飾句を引数とする．
- 動詞句では，動詞またはその動詞句中に含まれている動詞句を引数とする．
- 前置詞句では，前置詞を作用子とする．
- 必要に応じ，格を表す ＊subject，＊object や，受動態を表す ＊en などの機能語 (function word) を作用子として用いる．

図2に PAL の表現例を示す．

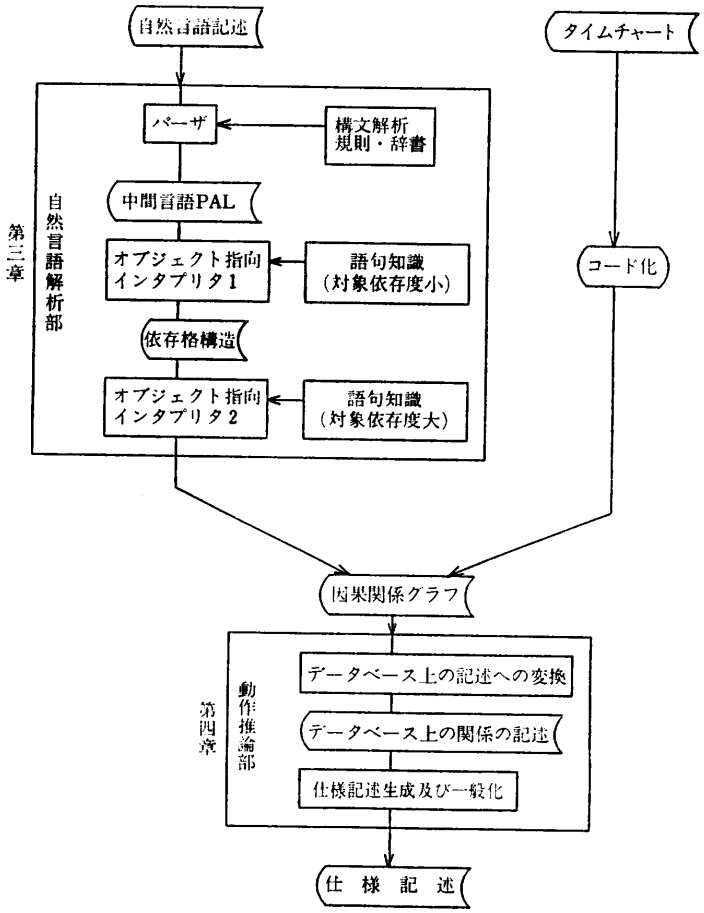


図1 システム構成
Fig. 1 System configuration.

3.2 オブジェクト指向 (object-oriented) な意味解釈

PAL の意味解釈はオブジェクト指向なインタプリタによって行う．すなわち，PAL に含まれる各語彙項目にオブジェクトとよぶ計算要素を代入し，これを評価することによって新たな中間表現として依存格構造を生成する．このオブジェクト指向の考え方は，Smalltalk[2]や LOOPS[1]によってプログラミング言語に導入されたものであるが，我々は，これは次のような理由で自然言語解析にも有効な概念であると考え，導入した．

a) 階層度の高いプログラムの実行に必要な，データと手続きの対称性を持っている．すなわち，各オブジェクトは作用子の位置にくれば手続き，引数の位置にくればデータとみなされて，対称的に解釈される．

b) 言語内の共通の知識を共有するのに，階層構造

```
(1) "The CPU samples the data." (sentence)
      (*subject(the(CPU)))((*object(the(data)))(sample))
(2) "the rising edge of the clock" (noun phrase)
      (of(the(clock)))((the(rising(edge))))
(3) "(a system) which he develops" (relative clause)
      which((*subject(he))(*object(empty-np))(develop)))
(4) "If the wait line is active" (if clause)
      if((*subject(the((*classifier(wait))(line))))((*comp(active))(be)))
(5) "The READY signal will not go inactive." (modal and negative)
      (*subject(the((*classifier(READY))(signal))))
            will(not((*comp(inactive))(go))))
(6) "the RD and the MREQ" (and conjunctive)
      (and(the(RD))(the(MREQ)))
```

図 2 中間言語 PAL (Purely Applicative Language) の表現例
Fig. 2 Examples of intermediate language PAL (Purely Applicative Language).

における性質の継承のメカニズムが有効である.

c) 語彙駆動型のシステムを作成しやすい.

モンテギュー文法で用いられていたラムダ変換は, より柔軟なメッセージ交換の機構によって代行される.

3.2.1 オブジェクト間のメッセージ交換

各単語に対応するオブジェクトは, それぞれ一つのクラスをなす. また, PAL に現れる個々の語はそのインスタンスに相当する.

オブジェクトは, 図3に示すようなスロット・フィラー構造である (: は予約されたスロットを表す). 計算要素は, 図4に示すように, オブジェクトの :receive スロットに書かれる. :receive スロットの値 (=の右辺) は, ラベルと手続きのペアである. あるオブジェクトが PAL の作用子の位置に現れると, このオブジェクトに :args というラベルと引数となるオブジェクト (のリスト) のペアがメッセージとして送られる. メッセージを受け取ったオブジェクトは, その :receive スロットの値から, 受け取ったラベルと同じラベルを持つ手続きを取り出し, これを実行する. このとき, *というラベルは, どんなラベルにもマッチするものとする. これらの手続きによって, オブジェクトのスロット値へのア

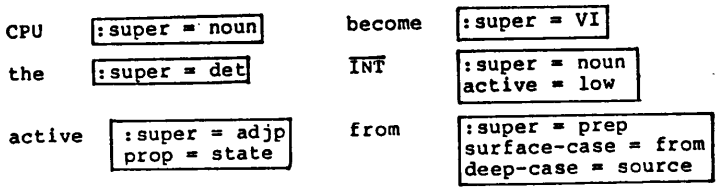


図 3 オブジェクトの構造
Fig. 3 Structure of objects.

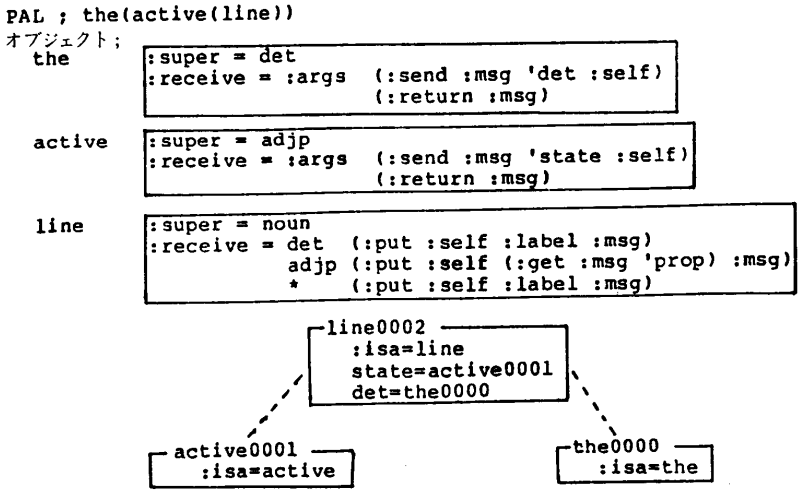


図 4 PAL: the (active (line)) に対する依存格構造の生成
Fig. 4 Generation of dependency case structure for PAL the (active (line)).

クセスやその更新を行う．このように，オブジェクト間のメッセージ交換によって，意味解釈に必要な格情報などを補いながら依存格構造を生成する．

依存格構造は，図4の処理結果に示したような，ある語に対し，その修飾語をラベル付きの枝でつないだ木構造である．ここで，ラベルは枝でつながれた語の間の関係を表す．文に対する依存格構造では，動詞を構造の最上位に置く．

図4の処理例では，line のインスタンス line 0002 の下位に active, the のインスタンス active 0001, the 0000 が結合した構造が得られる．

同じ品詞に分類される語の多くはパラメータを除いて同じ手続きを持つと考えられる．例えば，冠詞の多くは図4の the と，形容詞の多くは同図の active とパラメータ以外は同じ手続きを持つと考えられる．したがって，同じ品詞に属する語の間で手続きを共有できるのが望ましい．そこで，オブジェクトに階層構造を取り入れ，求めるラベルが見つからない場合には，上位オブジェクトを探索することにする（性質の継承）．

3.2.2 オブジェクト指向インタプリタによるPALの評価

インタプリタは，入力として与えられた PAL を次のように評価する．

（a） 入力がアトムであれば，それに対するオブジェクトを例示 (instantiate) する．対応するオブジェクトが定義されていないときは，その旨を表示する．

（b） 入力が作用子—引数のリストであれば，以下に示した関数の適用 (application) を行う．

（b-1） 作用子がアトムであれば，引数を評価し，その結果得られた値と作用子との間で 3.1 節で述べたメッセージ交換を行う．この結果返ってくる値は，上記（a）と同様オブジェクトのアトムである．

（b-2） 作用子が再び作用子—引数のリストであれば，これを評価し，その評価値と引数との間で関数の適用を行う．

3.2.3 応 用 例

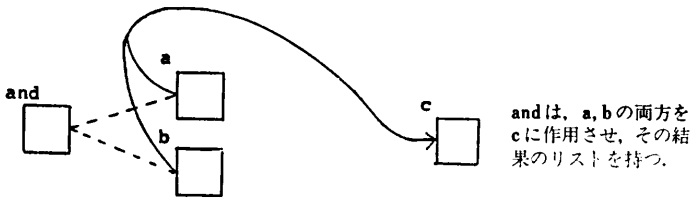
（1） 並列句

a and b という並列句に対する PAL の表現は and (a, b) である．これを評価すると，and は (a, b) を自身の ：dest スロットの値とし，評価値として and 自身を返す．この句が他の句と干渉する場合の処理方法を以下に述べる．これらの作用を図示したもの，および処理結果の一例を**図5**に示す．

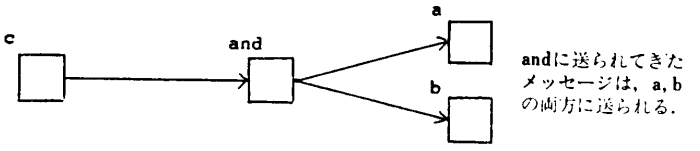
（a） この返された値が作用子になったとき，すなわち (and (a, b)) (c) のような場合，and は ：dest スロットの値のオブジェクトを順に作用子として引数に作用させ，その結果のリストを持つ．すなわち，(and (a, b)) (c) は，and (a(c), b(c)) の形で処理される（図5（a）の場合）．

（b） c (and (a, b)) のように and が引数になった場合，and に送られてきたメッセージをスロットに書かれたオブジェクトすべてに順に送る（図5（b）の場合）．

図5（c）の処理例では，(＊ subject (and (the ($\overline{\text{MREQ}}$)) (the ($\overline{\text{RD}}$)))) の部分が同図（b）の形で処理された後，この評価値と ((＊comp (active)) (become)) の部分が同図（a）の形で処理され，図のような依存格構造が得られる．



（a） (and (a, b)) (c) ⇒ and (a (c), b (c))

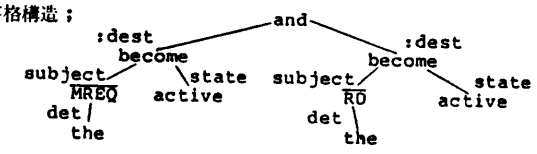


（b） c (and (a, b))

入力文; The $\overline{\text{MREQ}}$ and the $\overline{\text{RD}}$ becomes active.

PAL ; (*subject(and(the($\overline{\text{MREQ}}$))(the($\overline{\text{RD}}$))))((*comp(active))(become))

依存格構造；



（c） 処理例

図5 並列句 and の処理

Fig. 5 Processing a conjunctive phrase.

(2) 受動態

受動態では，意味上の主語および目的語がそれぞれ前置詞 by の目的語，主語となっている．受動態文の解析においては，各語句の意味上の格を依存格構造に反映させなければならない．そこで pipe とよぶオブジェクトを用いる．

pipe は，自身に送られてきたメッセージのラベルをあらかじめ定義されてある変換表に従って書き換え，書き換えたメッセージを :dest スロットのオブジェクトに送る．

受動態は，PAL では *en ("他動詞") により表現される．*en には *UKEMI というオブジェクトのインスタンスを代入する．*UKEMI は pipe の下位クラスとして定義する．*UKEMI の持つ変換表に，subject→object, by→subject と書き換える旨を記しておくと，pipe の機能により動詞のオブジェクトには書き換えられたメッセージが送られる．この処理を**図6**に示す．

3.3 因果関係グラフへの変換

3.2節で述べたようにして得られた依存格構造よりLSIの動作に関する情報を抽出する．抽出された情報は，LSIの個々の動作の間の因果関係・時間関係を表すネットワーク構造（因果関係グラフ）で表した．

3.3.1 因果関係グラフの形式

イベント間の関係を表すのには，**表1**に示した4種類の形式を用いる．また，これらの形式をイベントの生起する順にならべることによって，イベント間の時間関係を表す．このような表現を因果関係グラフとよぶ．

表1 因果関係グラフに用いられる形式と意味
Table 1 Forms and semantics of causal graph.

形　式	意　味
cause (e_1, e_2)	e_1 が原因となり，e_2 が起こる．
at (e_1, the$i$ (e_2, after (i, e_3)))	e_3 の生起後のある時刻 i で e_2 が起こり，その e_2 が起きた時刻 i に e_1 が起こる．
assert (symbol, e_1)	symbol は e_1 を表すものと定義する．(e_1 は信号をサンプルするイベント)
result (symbol, state, "因果関係グラフ")	symbol によって表されるイベントの結果が state と一致すれば，"因果関係グラフ" で表される一連の動作が起きる．

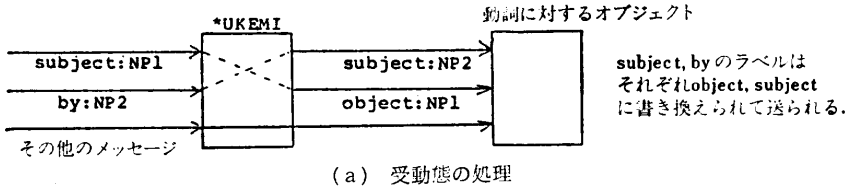
3.3.2 因果関係グラフへの変換

依存格構造からの情報抽出は，そこで用いられている単語や表現法に大きく依存する．そこで，ここでも前述のオブジェクト指向の手法を利用し，単語ごとに独自の手続きを持たせることにする．

また，情報抽出においては，LSIに関する基本的な知識はあらかじめオブジェクトとしてシステムに与えておき，これを利用する．ここで，基本的な知識とは，信号名，各信号が正論理か負論理か，などである．一例として，動詞 become の処理について述べる．become の用いられる典型的な記述例とその依存格構造，およびそれから得られる変換結果を**図7**に示す．

become に対する手続きは次のようになる．

(1) become の cause スロットを調べ，そこに書かれたオブジェクトにメッセージを送る．何も書

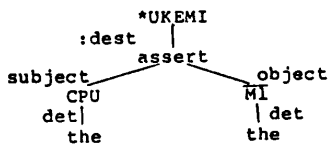


(a) 受動態の処理

入力文； The M1 is asserted by the CPU.

PAL ; (*subject(the(M1)))((by(the(CPU)))(*en(assert)))

依存格構造；



(b) 処理例

図6 受動態の処理
Fig. 6 Processing a passive phrase.

かれていなければ，time, condition, time-condition の各スロットについて同様にメッセージを送る．(time スロットには at に続く副詞節や when 節，condition スロットには if 節，time-condition スロットには until 節，after 節などがくる．) いずれのスロットにも何も書かれていなければ，処理を中止する．

(2) (1)でメッセージを受け取ったオブジェクトは，それの持つ手続きに従って処理を行い，become にその結果の値を返す．(図7の例では，edge は，name スロットの値として "asserted clock" を持つオブジェクトを返す．)

(3) become の subject スロット，state スロットのオブジェクトにメッセージを送り，両者から返ってきた値より，イベント名を生成する．(図7の例では，"asserted $\overline{\text{M1}}$" が生成される．) このイベント名を name スロットの値として持つオブジェクトを生成する．

(4) (2)で得たオブジェクトを e_1，(3)で得たオブジェクトを e_2 として，形式 cause (e_1, e_2) を生成する．

LSI の動作の記述に用いられる他の動詞 (raise, sample など) についても，become の場合と同様に，主語・目的語・前置詞句などを因果関係で結び，形式を生成する手続きを持たせる．また，上記の手続きと同様，ほとんどの名詞・形容詞には，その語が含意する因果関係を表す標準的な述語 (例えば形容詞 active では "asserted") を値として返す手続きを持たせれば良い．しかし，edge のような語の場合には，その語自身よりも修飾語句の持つ情報が重要であると考えられるので，さらに下位のオブジェクトにメッセージを送って，返されてきた値を処理する必要がある．

4. 動作推論部

3章で述べた因果関係グラフは，個々のイベントの間の因果関係・時間関係を表すものである．本章では，これらの関係から LSI の仕様記述を導く過程について述べる．ここでは主として同期式の動作を扱うものとする．

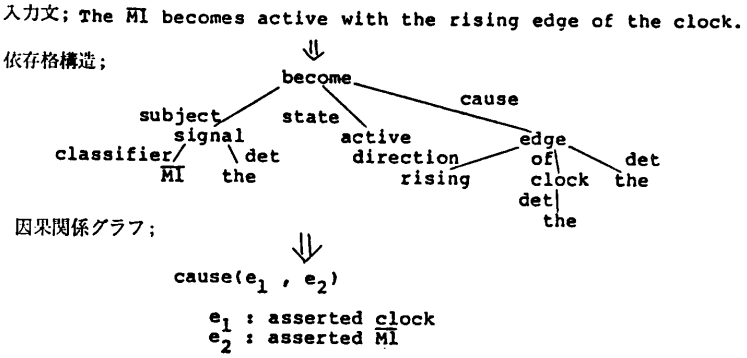


図 7 動詞 become の処理例
Fig. 7 Processing example of a sentence including a verb 'become'.

4.1 LSI 動作間の関係の記述

先に述べたように，因果関係グラフは個々のイベントの間の因果関係・時間関係を表すものである．仕様記述を求めるため，これらの関係をシステムのデータベース上に記述する．

4.1.1 関係記述のためのデータベース

イベント e_1 e_2 … e_n の間に関係 R_1 が成り立っているとする．この関係の記述は，関係データベース R_1 にリスト (e_1 e_2 … e_n) を登録することによって行われる．

データベースの入力，検索などは，データベースにメッセージを送ることによって行われる．これらのメッセージを表2に示す．また，検索は LIFO 的に行われる．

4.1.2 データベース上への関係の記述

因果関係グラフにおける各形式を変換し，データベース上に記述（登録）する手続きについて述べる．因果関係グラフで表されるイベント間の因果関係・時間関係は，データベース上では，原因 e_1 と結果 e_2 の因果関係を表す ＊cause (e_1, e_2)，e_1 の後に e_2 が起こることを示す ＊after (e_1, e_2) によって表される．ここで＊はデータベース上に記述される関係を表す．現在我々が対象としている範囲では，LSI の論理レベルでのふるまいだけしか考慮していないので，この二種類の関係で LSI の動作間の関係を記述し得る．

因果関係グラフは，形式をイベントの生起順に並べているが，システムはこれらの形式を次に述べる方法で順に処理し，データベースに登録する．

(1) cause (e_1, e_2) の登録

イベント e_1 (原因) を含む因果関係を関係 ＊cause のデータベース中から探索する．これがなかったときは，直前に登録された因果関係の原因となっているイ

表 2 データベースのためのメッセージ
Table 2 Messages for communicating with the database.

メッセージ	
PUT "パターン"	"パターン" をデータベースに登録する.
REMOVE "パターン"	"パターン" にマッチするすべてのパターンをデータベースから削除する. ここで, "パターン" 中に用いられる ? は, 任意のアトム・リストにマッチする.
FIND "パターン"	"パターン" にマッチするものをデータベース中より探索する. このメッセージを受け取ると, データベースはジェネレータとよぶオブジェクトを返す. ジェネレータに NEXT というメッセージを送るごとに, "パターン" にマッチするものが新しく登録されたものから順に一つずつ返される. マッチするものがなくなると, NIL が返される. "パターン" 中に (? "シンボル") の形式を用いると, ジェネレータは, シンボルと, これにマッチしたデータベース中のパターンとの a-list を返す. また ? が用いられると, パターンの有無のみを判定する.

ベント E'_1 を探索し, (E'_1, e_1) を関係 *after に, (e_1, e_2) を関係 *cause に登録する. 探索の結果 e_1 があった場合には, 単に (e_1, e_2) を関係 *cause に登録する.

(2) at $(e_1, the_i\ (e_2, after\ (i, e_3)))$ の登録

e_3 は, (1)の e_1 と同様に, e_3 を含む因果関係を探索し, その結果により関係 *after に登録する. 次に, e_3 と e_2 は単なる時間関係, e_2 と e_1 には因果関係があるとみなし得るので, (e_3, e_2) を関係 *after に, (e_2, e_1) を関係 *cause に登録する.

(3) assert (symbol, e_1) の登録

関係 *assert に (symbol, e_1) を登録する.

(4) result (symbol, state, "因果関係グラフ") の登録

関係 *assert より (symbol, e_2) を満たすイベント E を探索する. この結果得られた E_2 に対し, (e_1, E_2) を関係 *cause より探索する. この結果得られた E_1 と E_2 で, 関係*cause より (E_1, E_2) を削除し, $(E_1, (E_2.\ state))$ を登録する. これは, E_1 が原因となって起きた, 信号をサンプルするイベント E_2 の結果が state で表されていることを示す. 最後に, "因果関係グラフ"の処理を行う.

変換例を図8に示す.

4.2 仕様記述の生成

4.1節で述べたようにしてデータベース上に記述された因果関係グラフより, 仕様記述を生成する. 生成処理には, 新情報が与えられたときにそれまでに構成されていた仕様記述の一般化・個別化を行う機能を組み込んだ.

4.2.1 仕様記述の形式

仕様記述は, LSIの各モジュールごとに記述される. 各モジュールは, 次の形式で記述される.

(1) 各モジュールは状態を持つ.

(2) 各状態はいくつかの production-rule を持つ.

(3) production-rule は, 出力条件, 出力, 遷移条件, 次状態をそれぞれ持つ.

(4) 出力条件, 遷移条件は, それぞれ production-rule の出力, 状態遷移が起動される条件となるイベントもしくは条件である.

(5) 出力は, 出力条件が満たされたとき, 結果として起こるイベントである.

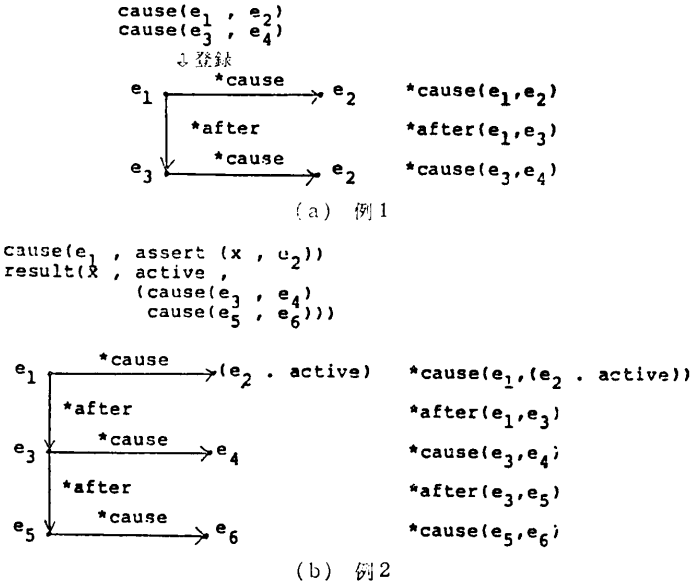


図 8 データベース上への関係の記述
Fig. 8 Description of event relation.

表 3 データベース上に記述される仕様記述の形式
Table 3 Forms of specification description described on database.

関係	登録されるリスト	意味
*HOLD	(m, s)	状態 s はモジュール m に属する.
*PRO	(s, p)	状態 s は production-rule P を持つ.
*COND-A	(p, c)	c は production-rule P の入出力条件である.
*ACTION	(p, e)	e は production-rule P の出力である.
*COND-T	(p, c)	c は production-rule P の遷移条件である.
*NS	(p, s)	s は production-rule P における次状態である.

(6) 次状態は，遷移条件が満たされたときの遷移先の状態である.

各モジュールは，最初初期状態にあり，出力条件，遷移条件が満たされるのを待つ．これが満たされると，出力イベントを起こし，次状態へ遷移する．遷移先の状態では再び出力条件，遷移条件が満たされるのを待つ．このように，状態遷移—イベントの出力を繰り返すことによって，LSI の動作をシミュレートすることができる.

仕様記述は，表 3 に示した 6 種類の形式によって 4.1.1 項で述べたデータベース上に記述される.

4.2.2 仕様記述の生成

仕様記述の生成は，データベース上に変換・記述された因果関係グラフをトラバースしながら各イベントに立ち寄って処理することで行う．このとき，イベントの生起する順にトラバースする必要があるが，因果関係において結果として起こるイベントは，時間的には原因となるイベントの直後（同期式では同一時刻）に起こると考えられるので，時間関係より先に因果関係をトラバースする必要がある．そこで，トラバースの手順は次のようになる.

(1) 根から関係 *cause をなぞれなくなるまでなぞる.

(2) 根から関係 *after を一つだけなぞる.

(3) (2)でなぞったイベントを根と見なし，(1)～(3)を再帰的に繰り返す.

次に，各関係から仕様記述への変換処理について述べる.

(1) *cause (e_1, e_2)

e_2 が引き起こされるモジュールの状態の production-rule において，e_1 を出力条件・遷移条件，e_2 を出力とする.

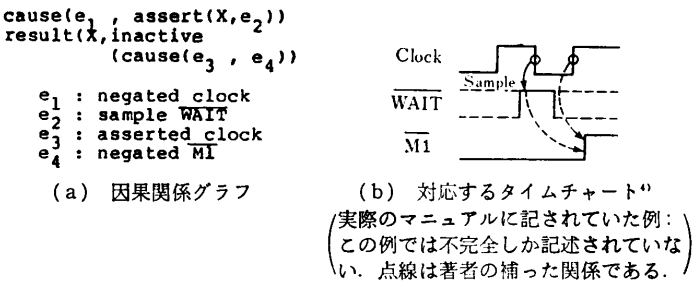
(2) *cause (e_1, (e_2. state))

e_2 が引き起こされるモジュールの状意の production-rule において，e_1 を出力条件，e_2 を出力，state と e_2 のイベント名に表れる信号名のリストを遷移条件とする.

(3) *after (e_1, e_2)

e_1 が出力条件であるような状態の production-rule を探索し，その次状態を e_2 に関して (1), (2) で述べたようにして定められた状態とする.

因果関係グラフからの仕様記述の生成例を図 9 に示す.



(a) 因果関係グラフ

(b) 対応するタイムチャート[4]
(実際のマニュアルに記されていた例：この例では不完全しか記述されていない．点線は著者の補った関係である.)

(c) 仕様記述

(d) 仕様記述を図示したもの
(大文字が遷移条件，小文字が「出力条件/出力」)

図 9 仕様記述の生成
Fig. 9 Generation of specification description.

4.3 仕様記述の一般化・個別化

マニュアルなどにおけるLSIの動作記述では，タイムチャートなどによって典型例を示すだけのことが多い．このような場合，典型例から得られた情報を一般化する能力が必要になる．例えば，**図10**の二つのタイムチャートから仕様記述を構成すると**図11**のようになるが，これはLSIの動作を正しく表していない．そこで，次に述べる仕様記述の一般化・個別化を行う．

4.3.1 状態の従属関係

状態 S_i, S_j が次の条件を満たすとき，S_i は S_j に従属しているということにする．

「S_i の出力条件・遷移条件はすべての S_j のそれに含まれ，かつ，それぞれの遷移条件に対する遷移先の状態を S_i', S_j' とすると，S_i' が S_j' に従属しているとき，あるいは，S_i の遷移先が定義されていないとき，S_i は S_j に従属している．」

上のように状態の従属関係を定義すると，図11の仕様記述では，S_4 が S_1 に，S_2 と S_5 は互いに，S_3 と S_6 は他のすべての状態に従属していることになる．ここで，S_i が S_j に従属しているとき，S_i を S_j と同一の状態であるとみなしても良いと考える．そこで，図11で S_1 と S_4, S_2 と S_5, S_3 と S_6 が同一の状態であるとみなすと*，**図12**のようなLSIの動作を正しく記述した仕様記述が得られる．

この手法は，オートマトンの状態数最小化の手法をもとに，新情報による仕様記述の修正を考慮して，柔軟な形にしたものである．

4.3.2 一般化・個別化の戦略（strategy）

前項で述べた状態の従属関係によって仕様記述の一般化・個別化を行う．このとき用いる戦略は次のものである．

「ある状態は，他のいずれかの状態と同じものである（従属している）と考える（一般化）．他のどの状態にも従属しなくなったとき，新しい状態と考える（個別化）」．

例として，**図13**のタイムチャートが入力として与えられた場合を考える．これは ϕ の三回目の立ち上がりで初めてBが立ち上がる，一種のカウンタである．この入力に対して仕様記述を生成する過程を**図14**に示した．まず，Aの一回目の立ち下がりまで入力すると，(a)のような仕様記述が得られる．次にAの二回目の立ち下がりまで入力すると，(b)のようになる．

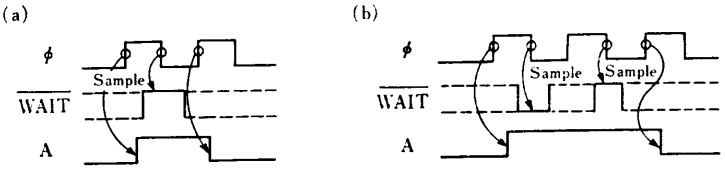


図10 タイムチャート入力例
説明：クロックの立ち下がりで $\overline{\text{WAIT}}$ をサンプルし，アクティブでなければ次のクロックの立ち上がりでAが立ち下がる．
Fig. 10 An example of time-chart inputs.
Explanation: $\overline{\text{WAIT}}$ is sampled with the falling edge of the clock. If $\overline{\text{WAIT}}$ is not active, A becomes active with the next rising edge of the clock.

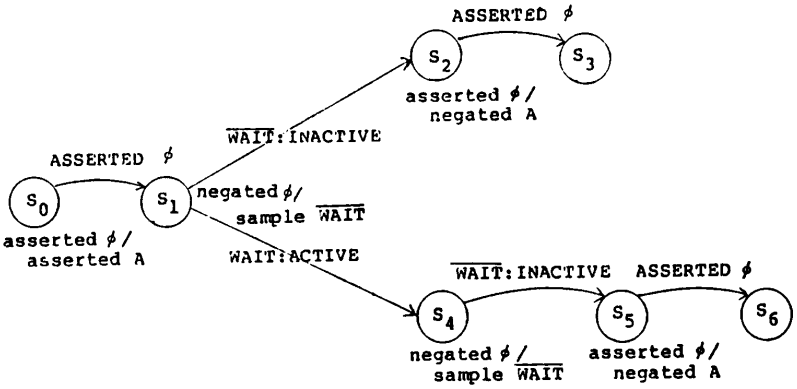


図11 図10に対する仕様記述（一般化を行わない場合）
Fig. 11 Specification obtained from the time-chart input in Fig. 10 (not generalized).

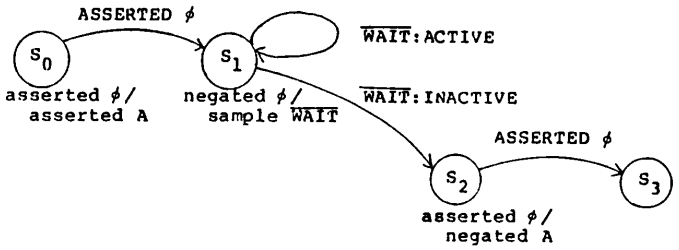


図12 図10に対する仕様記述（一般化を行った場合）
Fig. 12 Specification obtained from the time-chart input in Fig. 10 (generalized).

* 図11の S_3, S_6 のように複数の状態に従属している場合，どの状態と同一であるとみなすかは，さらに他の情報を必要とする．

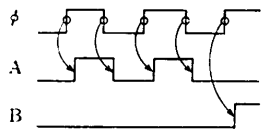


図 13 タイムチャート入力例
説明：1，2 回目のクロックの立ち上がりではAが，3回目にはBが立ち上がる．
Fig. 13 An example of time-chart inputs.
Explanation: A becomes active with the first and second rising edge of the clock, and B becomes active with the third rising edge of the clock.

このとき，S_2 は S_0 に，S_3 は S_1 にそれぞれ従属であり，これらは同じ状態であるとみなされている．すなわち，この時点での仕様記述は (b') のものであると考えられる．次に，図 13 を最後まで入力すると，(c) のようになる．このとき，S_4 は，出力条件が他のいずれとも異なるため，どの状態にも従属しない．すると，S_3 と S_1 の間で，それぞれの遷移先の状態 S_4 と S_2 の従属関係がなくなったために，従属関係がくずれる．そこで，ここではじめて S_3 は独立した一つの状態とみなされる．同様に，S_2 と S_0 の従属関係もくずれ，S_2 が独立した状態とみなされる．このような手順で一般化・個別化が行われる．

5. 実験と検討

以上に述べたシステムのプロトタイプを汎用計算機上の UTILISP によって作成した．システムのカーネル部は約 1500 行であった．このうち，自然言語解析部におけるパーザが約 650 行，オブジェクト指向インタプリタが約 350 行，動作推論部は約 500 行であった．現在のところ，構文解析用の文法規則は約 40 種，単語辞書は約 150 語，オブジェクト指向インタプリタ用のオブジェクトは約 120 種定義している．LSI に関する知識としては，モジュール名，信号名および信号の属するモジュールなどをオブジェクトの形で与えた．また，現在は，PAL の評価用の知識と因果関係グラフ生成用の知識は区別せず，同じオブジェクトとして扱っている．

実験は，このプロトタイプを用いて，LSI の動作記述からの仕様記述生成，タイムチャートを入力とした，動作推論部における仕様記述の一般化の二つに分けて行った．前者の実験では，対象とする LSI (Z 80) のマニュアル中に現れる，基本的なマシンサイクルに関する典型的な動作記述 27 例から仕様記述が生成可能であることを確認した．**図 15** に処理の一例を示す．また，この実験により，自然言語解析におけるオブジェクト指向の手法の有効性が確認できた．

また，後者の実験では，マニュアル中に現れるタイムチャート 4 例について，典型例から得られた知識を一般化して構成された仕様記述が生成できることを確認した．これと同時に，動作推論部では，同期式の動作は扱えることが確認できた．

現在の問題点を次にあげる．

(1) 複雑な文脈関係や量化表現を含む記述が処理

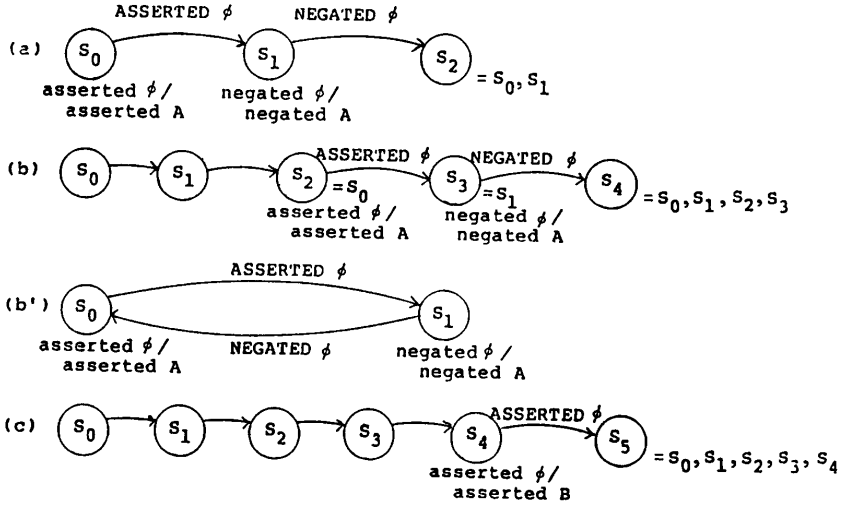


図 14 図 13 のタイムチャートに対する仕様記述の一般化・個別化 (=S_i は従属している状態を示す)
Fig. 14 Generalization and specialization of specification obtained from the time-chart input in Fig. 13. (=S_i represents that the state depends on S_i).

入力文1

```
THE *M1 BECOMES ACTIVE WITH THE RISING EDGE OF THE CLOCK #.
```

PAL

```
((*SUBJECT (THE *M1))
 ((WITH ((OF (THE CLOCK)) (THE (RISING EDGE)))) ((*COMP ACTIVE) BECOME)))
```

依存構造

```
BECOME0012:              ACTIVE0011:                  *COMP0010:
  :ISA=BECOME              :ISA=ACTIVE                  :ISA=*COMP
  STATE=ACTIVE0011         *CASE_MARKER=*COMP0010
  CAUSE=EDGE0009
  *SUBJECT=*M10002

EDGE0009:                  RISING0008:      THE0007:      CLOCK0006:
  :ISA=EDGE                  :ISA=RISING      :ISA=THE      :ISA=CLOCK
  DIRECTION=RISING0008                                      *DET=THE0005
  *DET=THE0007                                              *CASE_MARKER=OF000
  OF=CLOCK0006
  *CASE_MARKER=WITH0003

THE0005:      OF0004:      WITH0003:      *M10002:
  :ISA=THE      :ISA=OF      :ISA=WITH      :ISA=*M1
                                            *DET=THE0001
                                            *CASE_MARKER=*SUBJECT0000

THE0001:      *SUBJECT0000:
  :ISA=THE      :ISA=*SUBJECT
```

入力文2

```
ONE HALF CLOCK TIME LATER THE *RD GOES ACTIVE #
```

PAL

```
((LATER (*A (ONE_ADJP (HALF ((*CLASSIFIER CLOCK) TIME)))))
 ((*SUBJECT (THE *RD)) ((*COMP ACTIVE) GO)))
```

依存構造

```
GO0025:                    ACTIVE0024:                  *COMP0023:
  :ISA=GO                    :ISA=ACTIVE                  :ISA=*COMP
  STATE=ACTIVE0024           *CASE_MARKER=*COMP0023
  *SUBJECT=*RD0022
  TIME_CONDITION=TIME0019

*RD0022:                     THE0021:      *SUBJECT0020:
  :ISA=*RD                     :ISA=THE      :ISA=*SUBJECT
  *DET=THE0021
  *CASE_MARKER=*SUBJECT0020

TIME0019:                  CLOCK0018:
  :ISA=TIME                  :ISA=CLOCK
  *CLASSIFIER=CLOCK0018      *CASE_MARKER=*CLASSIFIER0017
  QUANT=HALF0016
  NUMBER=ONE_ADJP0015
  *DET=*A0014
  *CASE_MARKER=LATER0013

*CLASSIFIER0017:     HALF0016:      ONE_ADJP0015:      *A0014:
  :ISA=*CLASSIFIER     :ISA=HALF      :ISA=ONE_ADJP      :ISA=*A

LATER0013:
  :ISA=LATER
```

因果関係グラフ

```
((CAUSE E_0027 E_0029) (CAUSE E_0031 E_0033))

     (E_0027 (ASSERTED CLOCK))
     (E_0029 (ASSERTED *M1))
     (E_0031 (NEGATED CLOCK))
     (E_0033 (ASSERTED *RD))
```

仕様記述

```
CPU
  S_0043
    DEPENDENCY -----> NIL
    PRODUCTION_RULE :
      P_0044
        CONDITION FOR ACTION      -----> (ASSERTED CLOCK)
        RESULT                    -----> (ASSERTED *M1)
        CONDITION FOR TRANSITION  -----> (ASSERTED CLOCK)
        NEXT STATE                -----> S_0068

  S_0068
    DEPENDENCY -----> NIL
    PRODUCTION_RULE :
      P_0069
        CONDITION FOR ACTION      -----> (NEGATED CLOCK)
        RESULT                    -----> (ASSERTED *RD)
        CONDITION FOR TRANSITION  -----> (NEGATED CLOCK)
        NEXT STATE                -----> S_0104

  S_0104
    DEPENDENCY -----> (S_0043 S_0068)
```

図 15 自然言語記述からの仕様記述生成の例

Fig. 15 An example of generation of specification obtained from natural language description.

This signal*1 is sampled at the same time as the interrupt line*2, but this line*3 has priority over ···

(a) 複雑な文脈関係を含む記述

The interrupt signal ($\overline{INT}$) is sampled by the CPU with the rising edge of the last clock at the end of any instruction.

(b) 量化表現を含む記述

図 16 現在処理できない例
Fig. 16 Examples which the current system cannot handle.

できない．**図 16** に処理できない例をあげる．図16(a)では下線部 *3 の "this line" が下線部 *1，*2 のいずれを指示しているのか判定できない．また，同図(b)では下線部 "any" があるため，この記述の内容を単純な時間関係では表せない．

(2) 抽象度の高い表現 (cycle, state など，語それ自身に抽象的な情報の含まれる場合) を扱っていない．

また，本論文では扱わなかったが，言語のあいまい性を解決するには，動作推論部で得られた対象に関する情報を言語解析のためにフィードバックするような制御が必要になってくると考えられる．

6. おわりに

本論文では，LSI やバスの動作記述を解析し，形式的な仕様記述を生成するシステムについて報告した．自然言語解析では，作用的形式を用いた中間言語とそのオブジェクト指向の解釈システムを用いることで，見通しよく，体系的な処理が行えることがわかった．また，抽出した情報を一般化する機能を組み込むことにより，知識獲得の能力が向上した．

今後の課題としては，より多くの記述からの情報抽出，抽象度の高い表現の処理，複雑な非同期式の動作を扱うことなどがあげられる．

謝辞 末筆ながら，討論していただいた研究室の諸氏に感謝致します．また，UTILISP を提供していただいた近山隆氏，移植に際しお世話になった東京大学工学部の白井英俊氏に感謝致します．

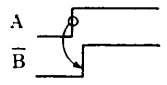
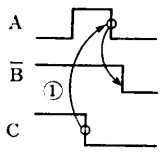
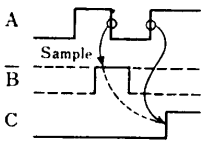
本研究の一部（主として前半）は文部省科学研究費一般研究（B）59460206 に，一部（主として後半）は文部省科学研究費特定研究(1)—59118003—によるものである．

参考文献

1) Bobrow, D. G. and Stefik, M.: The LOOPS Manual (Preliminary Version), working paper, Memo KB-VLSI-81-13, Xerox Corporation (1983).
2) Goldberg, A. and Robson, D.: *The Smalltalk-80: The Language and Its Implementation*, ISBN 0-201-11371-6, Addison Wesley, Reading, MA (1983).
3) 西田豊明，堂下修司：モンテギュー文法に基づく機械翻訳への新しいアプローチ，*Bit*, Vol. 15, No. 3, pp. 231-247 (1983).
4) Zilog, Z 80-CPU/Z 80 A-CPU Technical Manual (1977).
5) Zilog, Z 80-PIO Technical Manual (1977).

付録 タイムチャートの記号化

タイムチャートによる入力は，現在，次表に示すようなコーディング規則に従って人間が記号化し，システムに与えている．

タイムチャート	記号化
	cause (e_1, e_2) e_1: asserted A e_2: negated B̄
	at (e_1: the*i* (e_2, after (i, e_3))) e_1: asserted B̄ e_2: negated A e_3: negated C (①が明らかに因果関係でない場合)
	cause (e_1, assert (x, e_2)) result (x, inactive (cause (e_3, e_4)) e_1: negated A e_2: sample B̄ e_3: asserted A e_4: asserted C

(昭和 59 年 10 月 29 日受付)
(昭和 60 年 5 月 9 日採録)