

仕様設計エキスパートシステムのアーキテクチャ†

中 村 行 宏^{††} 雪 下 充 輝^{††}
打 橋 知 孝^{††} 小 栗 清^{††}

プロセッサ・ハードウェア設計における論理設計より上位の設計支援技術は十分には実用になっていない。そこで、上位の設計工程を支援すべく、設計記述のレベルを仕様記述にまで高め、仕様の記述/解析を支援する技術、さらには、動作論理を自動合成する技術の研究を進めた。まず、仕様記述にふさわしい高位の記述モデルを、「処理」、「データ」という対象（オブジェクト）とそれらの「関係」によって表現した。このモデルに基づいた日本語仕様記述を自然言語に制限を加えた形で実現した。実際のプロトタイプシステムでは、意味解析、正当性検証には可読性、保守性の高さから ES 構築支援ツール KBMS のルール（251 個）で記述した。また、動作論理合成ではオブジェクトの分類、情報の補充にルール（96 個）を用い他の部分を Lisp で記述した。この結果、日本語仕様記述 84 行を処理するのに、仕様情報解析に約 30 秒、SFL 合成に約 5 分程度を要するという実験結果を得ている。修正、再解析を繰り返すことが設計作業の特徴であるので、解析処理は数十秒のオーダーであるものの、実用上は一層の高速化が必要である。一方、合成処理は最後に一度実行するだけであるから、数分オーダーでも実用上問題がないと考える。この論文で述べた手法は、ユーザは使い慣れた日本語を用いてハードウェアの設計を行うことができ、筆者らが狙いとしている仕様設計の段階からの支援を進めて行っていく上で有効な方法であると考えられる。

1. はじめに

プロセッサ・ハードウェア設計の現状を概観すると、レイアウト設計などの低位レベルの設計支援技術は実用になっているが、論理設計より上位の設計支援技術は十分には実用になっていない。そのため、全設計工程の 70% 以上を論理設計の上位の工程、すなわち、方式設計、機能・動作設計、論理設計で占めており¹⁾、これら上位の設計工程を支援するツールが強く望まれている。この要望に応えるべく、我々は高位の CAD の研究・開発を進めている。

研究に当たっての一貫した考え方は、(1)従来の接続記述ではなく、上位設計にふさわしい高位の設計記述法を開発し、人間にとって認識しやすい概念記述をそのまま設計データとして扱い可能とする、さらに、(2)方式設計の特徴である試行錯誤への対処に適した処理環境を実現する、というものである。そのため、設計という知的活動の中のルーチンワーク的な部分を自動化して、本来の設計の核となる部分に設計者が専念できる CAD システムの開発を目標とした。

この目標に対し、接続よりも高位の概念である「動作」を直接記述できる動作記述言語 SFL、会話処理

による検証系、ゲート論理回路の自動合成系を研究・開発済みである^{2)~7)}。

SFL をベースとした動作設計システムは、マン・マシンインタフェースの高位化という点で、現在の実用ツールとして最先端に位置するが、まだ、論理設計の専門家でないとは十分には使いこなせない。そこで、さらに、設計記述のベースを高位化して、仕様記述にまで高めることを次の目標とした。

一般に、プロセッサ・ハードウェアの論理仕様は自然言語で書かれている。そこで、論理設計の非専門家でも抵抗なく記述可能でドキュメント性の高い自然言語を設計言語として用い、仕様の記述/解析を支援する技術、さらには、動作論理を自動合成する技術の実現を目指し研究を進めた。この自動合成系の実現には、設計ノウハウを知識ベース化するなど知識処理技術の導入が必要であり、エキスパート・システムとしてプロトタイピングを行った。ただし、今回のプロトタイプにおいては、まったく自由な自然言語を許したのでは、ハード設計用 CAD としての本質以外の所での困難が大きくなるので、まずはハード仕様設計に的を絞ることにより制限を設けた。

なお、プログラム言語をベースにした仕様記述言語から RT レベル記述を生成するシステムは MIMOLA⁸⁾、Cathedral-II⁹⁾が存在し、自然言語によるインタフェースを備えた CAD についての発表¹⁰⁾もあるが、自然言語による仕様から動作論理を自動合成する技術については、これまで発表されていない。

† An Architecture of Specification Design Expert System by YUKIHIRO NAKAMURA, MITSUTERU YUKISHITA, TOMOTAKA UCHIHASHI and KIYOSHI OGURI (Knowledge System Laboratory, NTT Communications and Information Processing Laboratories).

†† NTT 情報通信処理研究所知識処理研究部

2. 仕様記述言語

筆者らは RT (レジスタ・トランスファ) レベルの動作記述言語 SFL をベースとした設計支援技術を開発し、ハードウェアの動作だけを記述することにより論理回路まで自動で合成可能な設計環境を整えてきた。

設計記述のベースをさらに仕様記述にまで高めるため、

- (1) 仕様記述にふさわしい、高位の記述モデル
- (2) 仕様記述向きの高位のマン・マシンインタフェース、について以下の研究を進めた。

2.1 基本モデル

ハードウェアの仕様を記述するのに必要な基本モデルについて述べる。

ここで扱う仕様は、SFL の上位に位置づけすることとし、記述のレベルを RTL 動作レベルから少しでも高位化、抽象化することを考えた。そこで、筆者らは、ハードウェアの動作を「処理 (プロセス)」と「データ」という対象 (オブジェクト) の記述と、それらオブジェクト間の「関係」の記述により表現することとした。このような簡明なモデルを考えた理由は、限られたタイプのオブジェクトによりプロセッサを構成する要素を表現し、ハードウェア設計のための限られた関係記述によってオブジェクトを結び付けることにより、記述を容易にし、かつあいまいさを少なくしようとしたからである。

このような狙いのもとにプロセッサの仕様を分析し

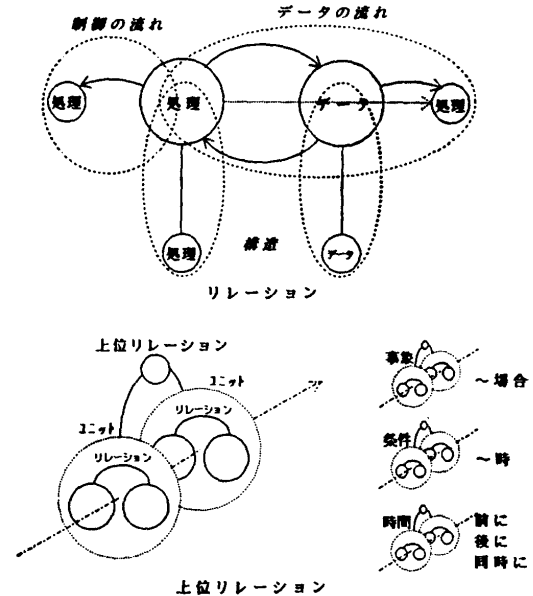


図 1 日本語仕様記述の基本モデル

Fig. 1 Basic model of specification description.

- (1) オブジェクトとしては、システムの動作の中で操作される情報を記述するための「データ・オブジェクト」、システムの動作の中の個々の処理を記述するための「プロセス・オブジェクト」を用意した。また、
- (2) これらオブジェクトを結び付ける関係としては、「構造関係」、「データの流れの関係」、「制御の流れの関係」を用意し、この関係をリレーションと呼ぶ。さらに、

表 1 リレーション

Table 1 Relation.

リレーション種別	関係	オブジェクト		機能
		ソース	シンク	
構造	構成 利 用 性	プロセス データ プロセス データ	プロセス データ プロセス データ	構成上の階層を示す データの集合を定義する 呼び出し構造を示す データの値を定める
データの流れ	参照 生成 保持 加工 転送	プロセス プロセス プロセス プロセス	データ データ データ データ	データを参照する データを作り出す データを保持する データの内容を変更する データの値が条件値であるから、条件値の変更と等価 データの転送を行う
制御の流れ	起動 終了	プロセス プロセス プロセス	プロセス プロセス プロセス	他のプロセスを起動し、自分は終了 他のプロセスを起動し、自分は動作 他のプロセスを終了させる
制御の流れ (上位リレーション)	前 後 条件 時間	— —	— —	リレーション間の二項関係を決める リレーション間の条件関係を定める リレーション間の時間関係を定める

- (3) リレーションをさらに上位で関連付ける「上位リレーション」を用意した。動作仕様を記述するには、処理の流れの中で、処理の順序関係／実行順序を制御する条件関係の記述が必要になる。具体的には、「事象（～の場合）」、「条件（～の時）」、「時間（～の前／後／同時に）」であるが、これらはオブジェクトどうしを結び付けているものではなく、リレーション相互を関連付けているものであり、これらを上位リレーションとした。

図1、表1にオブジェクト、リレーション、上位リレーションの関連を示す。

以下に、オブジェクト、リレーションの詳細を示す。
オブジェクト

- プロセス : 論理装置内で、基本的な個々の動作（例：レジスタ書き込み）を示すものから、複数の動作の集合（例：ALU）までの動作対象をプロセスとして記述する。具体的には、①レジスタ等のハードウェアの機能に1対1に対応する動作、②ALUのように複数の動作の集合の機能的枠組み、③ALU中の加算等の機能的枠組み中の1動作、すべてをプロセスで記述する。
- データ : 装置内で保持されたり、加工されたり、転送される情報を示す。

リレーション

- 構造による関係 : 複数のプロセスと、それらを構造的にまとめて構成される一階層上のプロセスとの相互関係を示す。また、あるプロセスとそのプロセスの機能を利用する他のプロセスとの相互関係を示す。
- データの流による関係 : データとそれを操作、転送するプロセスとの相互関係を示す。この関係を用いて、プロセス間のデータの流れを記述することが可能である。
- 制御の流れによる関係 : あるプロセスとそのプロセスを起す動作・終了・発生させるプロセスとの相互関係を示す。

上位リレーション

- 制御の流れによる関係 : 前記のデータ／制御の流れのリレーション相互間の実行順序／条件によ

り関係付けられたリレーション相互の関係を示す。

2.2 日本語仕様記述言語の文法

前述のオブジェクト、リレーションモデルを基本構造とし、記述形式として自然言語（日本語）を採用したものが仕様記述言語である。ただし、対象がハードウェアの仕様記述に限定されること、および、言語の構文解析を容易にすることを考慮し、以下のような記述上の制限を設けた。

- 制限事項：(1) 使用可能な助詞・助動詞の制限など、予約語の設定
(2) 1つの動詞に対する構文は1つに制限し、構文解析上の曖昧性を除去

2.2.1 予約語

ハードウェアの仕様表現に限定されることを考慮し、あらかじめ予約語を設定した。予約語は、表2に示す153個である。

現在の仕様でDIPS・CPUの論理仕様などいくつかの既存の例を対象に記述実験をしたところでは、必要とされる規定を表現できることを確認している。ただし、自然言語でのメリットを十分生かしていない所もある。

例えば、「ALUは演算結果をメモリに書き込む」を記述しようとする、「書き込む」という予約語が無いので「ALUは演算結果をメモリに送り、演算結果をメモリに保持させる」という記述になる。今後、評価実験を継続し、使用頻度が高く予約語とすることにより記述が容易になるものは、適宜予約語として整備して行く予定である。

2.2.2 構文

日本語仕様記述においては、人間からみて自然な表現を狙い、オブジェクト、リレーションを“主語”+“目的語”+“動詞”の形の単文で表現し、上位リレーションを“単文”+“接続語”+“単文”の形の複文で表現する。

(1) 単文

文に曖昧性を持たせないために主語は1つに制限する。以下に文型を示す。

①基本型 “主語”+“目的語”+“動詞”

例) 単純計算機は演算器から構成される。

レジスタは計算結果を保持する。

②拡張型1 “主語”+“目的語1”

+“と”+“目的語2”+“動詞”

表 2 予約語リスト
Table 2 List of reserved verb.

動	成る 定める 保持する 加工する デクリメントする 反転する AND する 桁 OR する なる 生成する する 発生させる	構成する 利用する 送る デコードする インクリメントする EOR する 桁 AND する 加える 更新する 参照する なった 終了させる	持つ 用いる 受け取る エンコードする シフトする OR する 桁 EOR する 引く 生じる 変換する 起動する
	詞		

助詞	は へ	が として	を で	から の	に	により	によって
接続	場合 かつ	場合に または	時	時に	前に	後に	同時に
名詞	属性 真	条件 偽	内容	データ	要素	結果	プロセス

注) 動詞は「構成し」等の連用形,「構成される」等の受動型を省略

複数の目的語を“と”で連結する。

動詞が同一で異なる目的語を持つ単文を一括に記述可能とする。

例) 演算器は加算器から構成される。

演算器は乗算器から構成される。

↓

演算器は加算器と乗算器から構成される。

③拡張型2 “主語”+“目的語1”+“動詞1”
+“目的語2”+“動詞2”

動詞1は連用形とする。~~~~~: 複数可

主語だけが同じ単文を一括に記述可能とする。

例) 演算器は加算器から構成される。

演算器は乗算器から構成される。

演算器は単純計算機を構成する。

↓

演算器は加算器と乗算器から構成され、単純計算機を構成する。

(2) 複文

①基本型 “主語”+“単文1”+“接続詞1”
+“目的語1”+“動詞1”

“単文1”を条件節として、単文中の主語と目的語の間に割り込ませた形とする。

これは、処理の主体である主語が行う処理を一括して記述することを主眼にしているためと、構文解析処理を簡単化するためである。

例) 命令フェッチ部は命令コードの内容が0である時、ADD を起動する。(制御の流れを表す複文)
条件節: 命令コードの内容が0である

実行節: 命令フェッチ部は ADD を起動する

②拡張型 “主語”+“単文1”+“接続詞1”
+“目的語1”+“動詞1”
+“単文2”+“接続詞2”
+“目的語2”+“動詞2”+…

動詞1は連用形。~~~~~: 複数可

条件ごとに異なった動作を行う処理を一括に記述可能とする。

例) 命令フェッチ部は命令コードの内容が0である時、ADD を起動する。

命令フェッチ部は命令コードの内容が1である時、SUB を起動する。

↓

命令フェッチ部は命令コードの内容が0である時、ADD を起動し、命令コードの内容が1である時、SUB を起動する。

3. 仕様解析技術

日本語仕様記述言語で記述された仕様は、パーサ部分である字句解析部、構文解析部、および、仕様記述からハードウェア構成を解析する意味解析部により解析される。また、解析結果はレポート出力部より得られる。

(1) 字句解析部では、入力された漢字かな混じり文の日本語仕様記述のテキストを予約語情報に基づいて単語に切り出す。

(2) 構文解析部では、名詞と予約語の列を、主語・目的語・動詞といった単文構造と、それらを組

み合わせて生成された複文構造を表す情報に変換する。

- (3) 意味解析部では、①単文に対し、主語・目的語として表されているオブジェクト相互間のハードウェア構成上での関係（包含関係等）を、動詞の意味により明らかにし、②複文に対し、単文相互間の関係（順序、条件等）を、接続詞の意味により明らかにする。上記の処理を意味付け処理と呼ぶ。③意味付けの正当性の検証をルールで行い、④①の結果からリレーションネットワークを、②の結果から上位リレーションネットワークを生成する。
- (4) レポート出力部では、エラーの表示に加えて、構造、データの流れ、制御の流れをユーザの必要時に随時木構造で表示する。

3.1 字句解析部

仕様記述入力が漢字かな混じり文であり、「かな」である助詞の切り出しが容易であることから、助詞に着目する。また、使用される名詞に固有名詞が多いことから、一般的な名詞辞書は持たない。そこで、設計仕様に対応させ登録した助詞・助動詞・接続詞・動詞・少数の名詞を予約語とする。字句解析部は、優先順位の関係から、予約語である助詞→助動詞→接続詞→動詞→名詞の順で検索を行い、最後にどれにも符合しないものを名詞として切り出し、最終的に仕様記述を名詞と予約語の列に変換する。

3.2 構文解析部

構文の解析は、①単文における主語・動詞・目的語の関係から導かれるリレーションレベルの意味解析と、②複文の構造から導かれる上位リレーションレベルの意味解析とに分けて行う。単文の解析は、動詞と目的語を指示する助詞による構文パターン情報を用いて行い、複文の構造の解析は、接続詞などの文を切り出す予約語の位置とそれによって切り出された単文の組合せの情報を用いて行う。

上記字句解析、構文解析の処理系は Lisp によって作成されており、図 2 に一例を示すような解析結果出力が得られる。

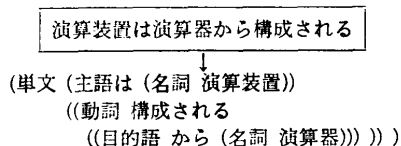


図 2 解析結果例

Fig. 2 Example of analysis result.

仕様記述で扱う単文では、唯一の主語に対して複数の動詞の組合せを許し、各々の動詞に対して複数の目的語の組合せを許す。したがって、解析結果はこの構造を容易に記述できるリスト構造としている。

3.3 意味解析部

意味付け処理は、前述のように、あらかじめ用意した動詞をキーワードとした構文パターンと、単文中の動詞、主語、および、目的語とのパターンマッチ処理により各オブジェクト相互間の関係を明らかにする。処理の基本がパターンマッチ処理であることから、エキスパートシステム構築支援ツール KBMS¹¹⁾を用いて、If-then 形式のプロダクションルールで処理を記述した。

また、正当性検証処理では、日本語仕様記述されたハードウェアの正当性を検証する際に使用する検証項目の可読性、保守性を高めるため、検証知識をルールで記述した。ここでは、構造における上下関係、データの流れに関するデータの供給側/受取側の完備性等の静的な関係の正当性を検証し、クロック等に関わる動的な関係は SFL レベルでシミュレーションを行い正当性をチェックする。

KBMS では事実に関する知識はフレーム^{11), 12)}、推論を行うための経験的な知識をルール、により表現している。

ここで、フレームは、フレーム名と、データを表すいくつかの変数からなり、変数は、データの性質を表す変数名（スロット名）と実際の値（スロット値）を持つ。また、フレームは、個々の事実を表す「インスタンス」と、複数のインスタンスに共通なデータの型等の宣言を行う「クラス」に分類される。

フレームに対してルールの照合を行うことにより推論を行う。

以下に、解析処理について具体例を用いて説明する。

3.3.1 解析処理

まず、解析処理の前処理であるフレーム生成について述べる。フレーム生成は、(1)オブジェクト、単文、複文フレームの生成、次にリレーションが階層を持つことから、(2)単文からリレーションの生成、(3)複文から上位リレーションの生成を行うという3段階からなっている。

- (1) オブジェクト、単文フレーム、複文フレームの生成
ここで、

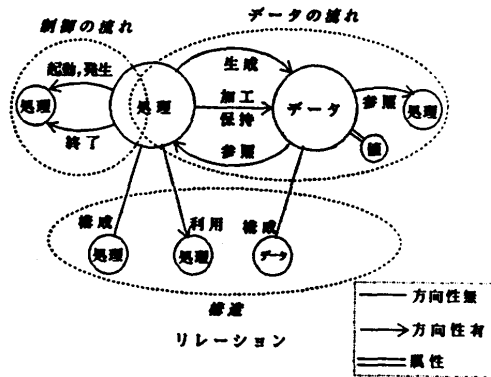


図4 オブジェクト間のリレーション
Fig. 4 Relationships between entities.

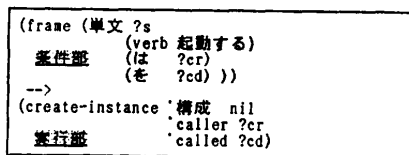


図5 リレーションフレーム生成ルール
Fig. 5 A rule for making relation instance frames.

代入されている目的語 ?cd を called (起動先) のオブジェクトとする「起動リレーション」クラスのインスタンスを生成する。生成されたリレーションフレームを図3(b)に示す。

動詞は能動態以外に受動態も扱えるよう、ルールを記述している。

上述のルールにより、単文レベル、および、それから生成されるリレーションレベルのフレームが生成される。

表4 解析可能接続詞
Table 4 Analyzable conjunctions.

関係	接続詞
事象時	場合に
条件時	時に
"/	前に
"/	後に
"/	同時に

(3) 複文から上位リレーションの生成 (図3(3))

単文フレームからリレーションのフレームを生成後、複文の解析を行う。単文レベルの解析では動詞に着目しているが、複文レベルでは接続詞に着目して処理を進める。処理手順は単文レベルと同様にルールで行う。対象とするフレームが複文クラスのフレームである点が異なるが、基本的な考え方は同様であるため、詳細は省略する。

現在解析可能な接続詞を表4に示す。

3.3.2 正当性検証

前節において、一括生成したリレーションネットワークの検証を行う。

検証項目として、構造に関するもの、データの流に関するもの、制御の流れに関するものがあり、それぞれのリレーションフレームに関するルールによって検証を行う。

これらのルールの検証項目一覧を表5に示す。

上記の項目をもとにして、リレーションごとに数個、合計で150余りのルールにより検証を行う。

具体的なルールとして、構造に関するチェックルールを用いて説明する。

表5 検証ルール項目
Table 5 Items of checking rule.

構造 再帰的構造 プロセスのグループ化	構造を示す関係がループを生じている プロセスが2つ以上のプロセスの構成要素となっている	25ルール
データの流れ 存在 保持と更新 プロセスとの関係 更新とデータの流れ プロセスとデータの入出力 名称の一貫性	未生成で存在する 保持されているが更新されていないデータがある、またはその逆 プロセスによって操作されていない 更新されたデータが利用されていない データを受け取るが、ほかにデータを渡さないプロセスがある または、ほかにデータを渡すがそのデータを受けていない プロセス間を移動するデータの名称が一貫でない	64ルール
制御の流れ 再帰 起動の欠如 プロセスの孤立	プロセスの起動関係でループを生じている 起動されていないものを終了する プロセスの起動/終了のつながりから孤立している	28ルール
上位の制御の流れ 非条件節の存在 再帰	構造の記述を複文の条件節に記述する 時間的なループを生じている	32ルール

```

(Frame (構成 ?p1 (上位 ?u) (下位 ?l) ))
(Frame (構成 ?p2 (上位 ?l) (下位 ?u) ))
-->
(call (format t "~%構成関係で矛盾があります
      A A" ?p1 ?p2) )

```

図 6 検証ルール例

Fig. 6 Example of check rule.

「AはBから構成される」という記述が存在し、上位構成物=A、下位構成物=Bなるリレーションフレームが存在する場合に、別のところで、

「BはAから構成される」という記述が存在し、上位構成物=B、下位構成物=Aなるリレーションフレームが生成されると、ハードウェアの構成上矛盾が生じる。このような矛盾を検出するルールを図6のように記述することにより、再帰的な構成を検出可能である。

このように、上位リレーションを除くリレーションの場合は、図7(a)のようにオブジェクトD1と因果関係があるP1、P2、P3は二項関係により直接関連付けられているので、D1を正当性検証時のパターンマッチ対象の中心としてルールを容易に記述可能である。

しかしながら、上位リレーションの正当性検証においては以下のような問題が生じる場合がある。

例えば、図7(b)のようにプロセスP4とP5に関する時間軸における因果関係の矛盾(P4がP5を起動した時間より後でP5がP4を起動しようとする時

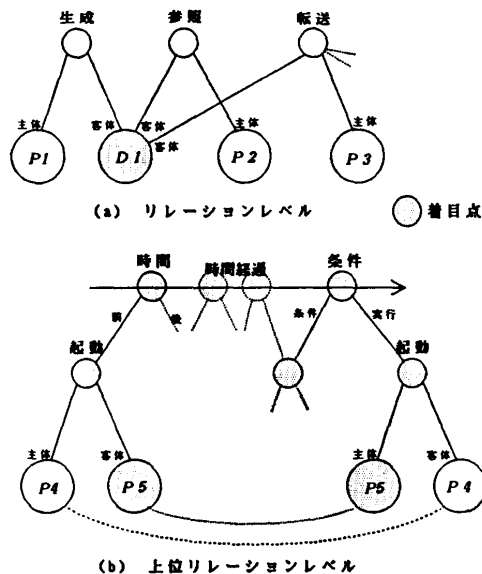


図 7 検証ルールの着目点

Fig. 7 Viewpoints of check rule.

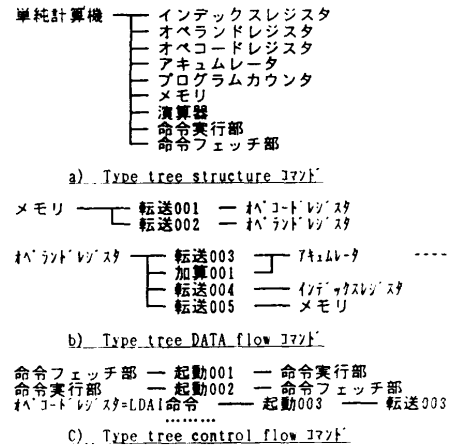


図 8 レポートの実行例

Fig. 8 Example of report executing.

間的ループ)を検出しようとする、上位リレーションの前後関係を組み合わせると、全体の時間の流れの中での矛盾を検出する必要性が生じる。

現在、4段の時間関係までチェックするルールを生成し、そのカバー率を実験中である。この検証を完全に行う手法については引き続き研究を進める。

3.4 レポート機能

仕様記述結果(仕様レベルの設計データ)に対し、前述の検証機能によりエラーをレポートするのは当然として、リレーションネットワークを解析することにより、設計者の意図どおりの仕様になっているかをシステムが報告してくれれば非常に有効である。ここでは、「構造」、「データの流れ」、「制御の流れ」を木構造で表示するコマンドを用意した。実験例で示す単純計算機についてそれぞれのコマンドによるレポート例を図8に示す。このように出力したレポートから仕様の全体像を把握することができる。その結果、必要であれば、記述を変更し、再度、解析処理をやり直す。

以上の操作により、各種誤りのチェックが設計の進捗に合わせて行える。

4. 動作論理合成技術

仕様設計が固まると、次に詳細な設計データを作成する必要がある。筆者らは、すでに、RTレベルの設計言語 SFL と SFL からゲートレベルの論理回路を自動合成するシンセサイザの開発に成功している。そこで、仕様レベルから論理回路レベルまでの体系的な設計支援技術の構築を狙い、仕様記述から RTL 動作記

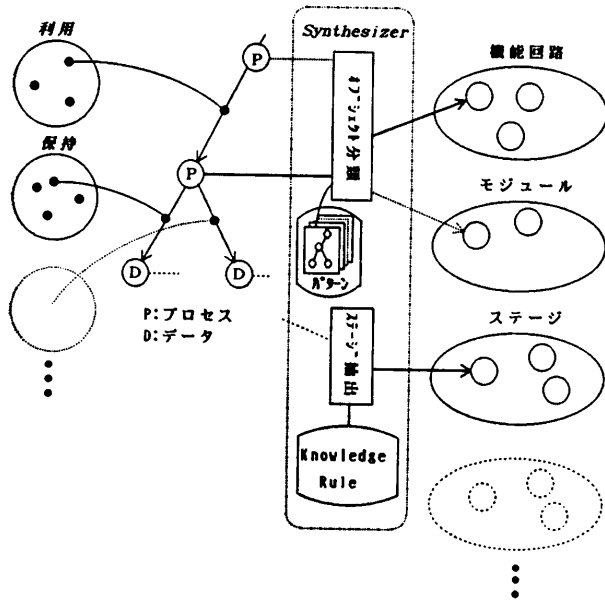


図 9 オブジェクト分類
Fig. 9 Classification of entities.

述への論理合成技術について考えた。

仕様記述では、前述したように処理、データ、リレーションにより表現している。そこで、これより、詳細な SFL 動作論理を合成するため、以下に示すような方法を検討した。

- (1) まず、仕様解析結果（オブジェクト×リレーション解析）により、オブジェクトを分類して、SFL の概念への対応付けを行う（処理にはモジュール/ステージ（SFL 用語：後述）/レジスタ/組合せ回路、データには端子/記憶素子、リレーションには信号線/デコーダ出力、が対応する）（図 9）。
- (2) 次に、モジュール、端子などに応じて、あらかじめ用意した SFL の記述形式を持つクラス・フレームに対し、仕様記述に応じてインスタンス・フレームを生成する。ここで、クラス・フレームは SFL の各記述項目をスロットとして持っており、仕様解析により得られたデータをもとにシステムがスロット値を与えていく。
- (3) 仕様記述で陽に記述されていないスロット値を知識により補充する。
- (4) 仕様記述段階では明記されていないが、動作記述で詳細化されるべき項目として、①クロック分割、②状態割付、③組合せ回路の機能などがある。これらを自動合成するには、最も高度な

設計ノウハウを知識ベース化する必要があるが、今回は、その第一歩としてシステムから設計者に問い合わせを行う会話処理形態として試作した。

以下にこれらの処理について具体的に述べるが、まず最初に、論理の合成先である SFL 動作記述について、概要を述べる。

4.1 動作記述言語 SFL

SFL 記述のモデルは、機能表現するのではなく、内部がどのような仕掛になっているかを表現する作りの記述のモデルである。図 10 に示すように

- ① ハードウェアの枠を表すモジュール（サブモジュール含む）、
- ② 動作主体を含み、動作を独立に制御できるステージ、
- ③ データを表すデータ端子、
- ④ 制御の流れを表す指示端子、
- ⑤ 動作主体を含まず、ただ使われるだけの存在で、ブラックボックスの扱いが可能な機能回路、メモリ、レジスタ、

等のハードウェア要素から成る。このモデルで動作主体であるステージが客体である機能回路、端子、指示端子を使う手順、および、他のステージへのタスクの遷移等を表現する。このように、並列に独立動作可能なステージの動作を記述することにより並列処理ハードウェアを記述することが可能である。

4.2 オブジェクト分類

リレーションネットワークにおけるオブジェクトがハードウェア要素のどれに対応するかを分類する。仕様記述のオブジェクトと SFL のハードウェア要素との対応は表 6 のようになっており、対応付けルールは、構造/データの流れ/制御の流れに関するものを

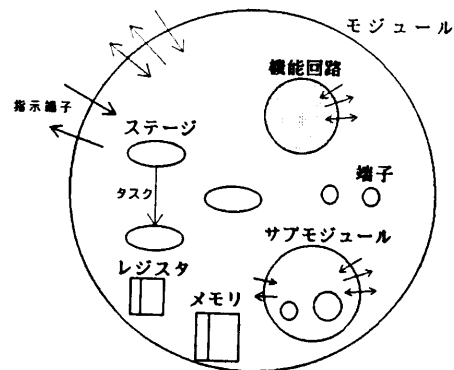


図 10 SFL のモデル
Fig. 10 Model of SFL.

表 6 仕様記述と SFL 記述との対応
Table 6 Relations between entities and elements.

仕様記述のオブジェクト	SFL の記述要素
プロセス	モジュール, レジスタ, 機能回路, 機能名など
データ	端子, 内部バスなど
制御	デコーダ, 指示端子など

表 7 ルールの分類
Table 7 Classification of rules.

プロセス	プロセスが機能回路に対応するか プロセスがレジスタに対応するか プロセスがモジュールに対応するか プロセスがステージに対応するか
構造	オブジェクトが下位のオブジェクトを持つか オブジェクトが構造の最下位であるか
データ	データがプロセスから外へ流れるか データがプロセスの中へ流れるか
制御	制御がデータの操作に影響するか 制御がプロセスから影響を受けるか

基本とし、これらの組合せによってプロセスの分類を行っている。表 7 にルールの分類を示す。

このルールは、各ハードウェア要素が持つ特徴（他プロセスへの制御の関係、構造のレベルおよび下位に持つプロセスの種類等）に注目したものである。

機能回路を例に分類の考え方を以下に示す。

機能回路の特徴は、以下の 4 つである。

- ①データを持つ
- ②データを加工する
- ③プロセスにより利用される
- ④プロセスを制御しない

したがってリレーションネットワーク内で、機能回路であるプロセスは、

- ①保持リレーションの保持の主体、
- ②加工リレーションの加工の主体、
- ③利用リレーションの利用の客体、

というリレーションを備えており、

- ④利用リレーションの利用の主体、

というリレーションを備えていないというパターンが定まる。

つまり、あるプロセスに関して①②③のリレーションがあり、④のリレーションがないならば、それを機能回路と同定する。

そこでこのようにリレーションのパターンが定まっていることに着目して、プロセスとハードウェア要素

との対応付けを行う。（図 9）

他のハードウェア要素の対応において着目する点を以下に示す。

- モジュールは構成要素としてモジュールまたは機能回路を持つプロセスである。したがって下位に“機能回路またはモジュールに対応するプロセス”を持つプロセスはモジュールとして同定する。
- ステージは自分の下位に構成要素を持たず、他のプロセスを制御する処理プロセスに 1 対 1 に対応させる。
- レジスタ、および、メモリは判断が難しいことから、“～はレジスタ（メモリ）である”の記述があることで判断する。
- 端子は、個々のデータの流れ、制御の流れを 1 対 1 に対応させる。

ここで、図 11 の仕様記述の例で演算器に着目すると、演算器は命令セットから利用され (③)、メモリデータを送られ (①)、演算結果を更新し (②)、かつ ④に示した記述がないので、機能回路に該当している。

4.3 フレーム生成

プロセスの分類によって認識されたハードウェア要素（モジュール・機能回路・ステージ）に対し、SFL 記述を合成するために必要な情報を取り出す必要がある。

SFL 記述は、モジュール、ステージから機能回路等のハードウェア要素に至るまで、その記述形式が用意されている。

そこで、その記述形式を枠組みとしてあらかじめ用意して SFL 動作記述を引き出すこととした。つまり、記述形式に対応するフレームを用意しておき、リレーションネットワークから得られる情報をフレームの持つスロットに対応させ、最終的に SFL 記述を生成する。

具体的には、KBMS のフレーム機能を用いて、

演算器は、ADD と AND から成り、
命令セットによって利用される。
演算器は、演算結果を更新する。
ANDX は、演算器とインデックスレジスタとレジスタと
レジスタのアドレスを利用し、
インデックスレジスタの内容をレジスタへ送り、
レジスタの内容とレジスタを演算器に送り、
演算器の結果をレジスタへ送る。
レジスタは、レジスタとインデックスレジスタと演算器に送られ、
bit 長が 8 である。
演算結果は bit 長が 8 である。

図 11 仕様記述の例

Fig. 11 Example of a specification.

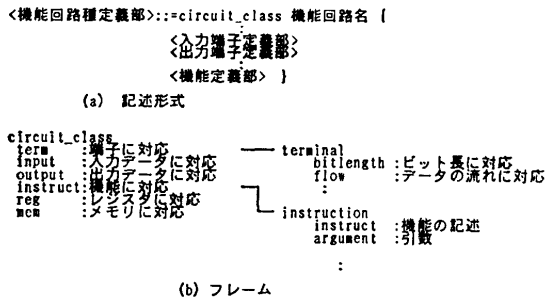


図 12 SFL 記述形式とフレーム (機能回路の例)
Fig. 12 SFL description style and frame.

SFL の構文・記述形式に対応したクラスフレームを用意する。このクラスフレームは図 12 に示すように SFL の記述を生成するのにどのような情報が必要であるかをスロットとして持っており、各スロットはその値に対する制約、値を得るための関数、デフォルト値等を情報として持っている。

図 11 の例の場合には、演算器は機能回路であったので、図 12(a) に示す機能回路の SFL 記述形式に対応するフレーム (クラス: 図 12(b) の circuit_class) に対応付けたインスタンスが生成される。

4.3.1 スロット補充

分類されたプロセスに対応するフレームのスロットを埋めるに当たり、仕様記述では陽に記述されていないが SFL では明示されなければならないものがある (例: 端子)。

‘端子’の値を得るためには、データ・制御の流れがどのプロセス間の流れであるか、流れの向きはどちらであるか、データの送元での値が記述されている場合上位リレーションが存在して条件となり得るか、等に着目した「知識」を用いて、データや条件に分類して種々の端子に対応付ける。

例えばプロセスの境界を越えて送出されるデータの流れは出力端子として判断し、ステージ間で境界を越えて相手の操作の制御を行う流れは指示端子と判断する。このようにスロットの値を自動的に設定する。

図 11 の例において、演算器に関係するデータの流れは以下のとおりである。

- ① “ANDX はアキュムレータの内容とメモリデータを演算器に送る”，
- ② “ANDX は演算器の結果をアキュムレータへ送る”，
- ③ “メモリデータは、演算器に送られ、bit 長が 8 である”，
- ④ “演算結果は、bit 長が 8 である”。

ここで①は演算器に対するデータの入力、②はデータの出力、③、④は各 bit 長を示す。これにより図 12 に示したフレーム [terminal] のスロット [bit-length, flow] が埋まり、演算器に対応するフレーム [circuit_class] の端子を示すスロット [term] と方向性を示す [input] が埋まる。

4.3.2 動作手順・機能 (演算) 定義入力

仕様記述では、動作の順序関係は部分的な二項関係だけが記述され、ステージ内の全体の順序は記述されていない。例えば、 $A \rightarrow B$, $C \rightarrow D$ (前→後) なる二項関係が記述されていた場合、全体の関係は $A \rightarrow B \rightarrow C \rightarrow D$, $C \rightarrow A \rightarrow B \rightarrow D$, 等、どれが正しいかは判断できない。したがって、知識で全体の順序を決定することは困難である。

また、ステージは複数の「状態」をとることができるが、仕様記述からは、どの動作をステージとして設計すべきか、または、あるステージの 1 つの「状態」として設計すべきかは、陽には判別できない。ここをいかに設計するかは、並列制御に関する高度な設計知識が必要である。何を知識ベース化するかを明らかにするため、今回の実験システムでは、独立に動作可能な処理の集合 (プロセス) は 1 ステージ (タスクは 1 つに制限) に対応させ、ステージ内の状態にどのような個別の処理を割り付けるかを設計者に問い合わせを行い会話的に情報収集を行うこととした。具体的には、ステージ S に処理 A, B, C がシステムにより割り当てられた場合、例えば、処理 A, B は状態 1 に、処理 C は状態 2 に対応させ、状態 1 から状態 2 に遷移するという指定を会話的に行う。

次に、機能回路の機能については、あらかじめライブラリに登録しておくか、登録されていなければエディタを通じて設計者に問い合わせすることとする。

演算器の機能 (AND, ADD) の記述はここで入力され、フレーム [instruction] のスロット [instruct] が埋まる。

以上により、SFL 記述生成に必要な詳細情報をすべて獲得する。そして生成されたフレーム (インスタンス) を展開することにより、SFL を生成する。

5. 仕様設計 ES の構成

以上の研究に基づき、仕様設計エキスパートシステム (ES) のプロトタイプを試作した。本 ES は図 13 に示すように、仕様解析技術を実現する仕様情報抽出部と動作論理合成技術を実現する設計情報合成部から

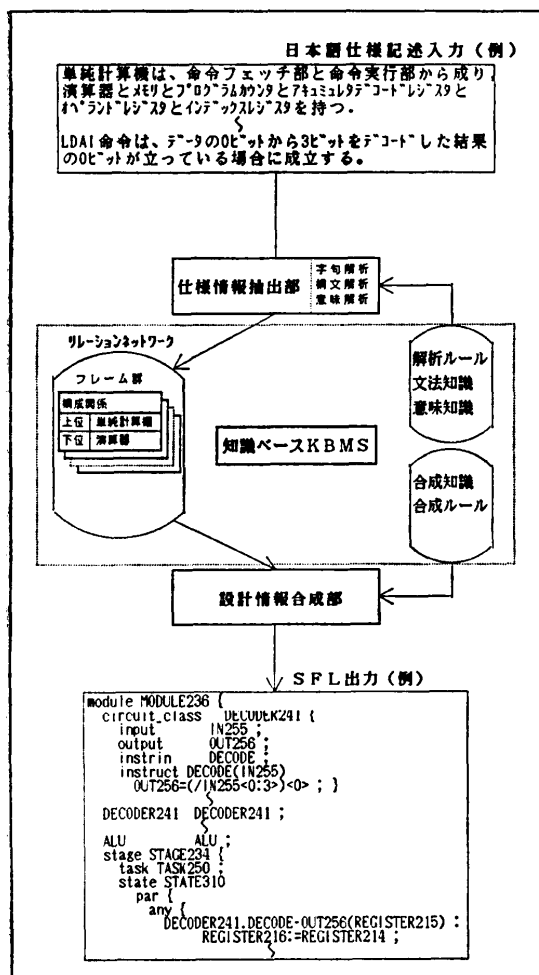


図 13 仕様設計 ES の概要

Fig. 13 Outline of specification design ES.

構成される。

3, 4章をまとめると、仕様情報抽出部では入力された日本語を論理装置の仕様として、その構造・データの流れ・制御の流れを解析・抽出する。次に、仕様記述の意味的矛盾検出を行うため、モデルに沿ってハードウェアを表現するリレーションネットワークを生成する。このリレーションネットワークにおいて、矛盾の無いものを最終的に仕様情報のリレーションネットワークとして出力し、設計情報合成部に渡す。

次の設計情報合成部では、リレーションネットワークから設計情報を抽出し、RTL (レジスタ・トランスファレベル) 動作記述言語 SFL による動作論理を合成する。

6. 実験例

単純計算機を対象に、上記プロトタイプを用いて、

仕様解析、動作論理合成の実験を実施した。結果を図 14 に示す。

SFL 記述は、①モジュールの宣言、②ハードウェア要素 (機能回路、サブジュール等) の定義、③実部品名の宣言、④ステージの動作記述、から構成される。

この例では、独立動作する命令フェッチ部と命令実行部がステージに展開されている。命令フェッチ部は、状態を2つ持ち、オペコードの7ビット目をみて2バイト命令かどうか判断し、命令実行部を起動する記述に展開されている。

日本語仕様記述 84 行を処理するのに、仕様情報解析に約 30 秒、SFL 合成に約 5 分程度を要するという実験結果を得ている。情報解析処理でのルールは、対象とする仕様記述が単文であれば動詞、複文であれば接続詞をキーワードとして処理を進めていることに着目して、キーワードごとにルールのグループ化を行っている。そこでは、グループ化のためのキーワードそのものを各ルールの先頭に置くなどの工夫をしている。これは、高速なパターンマッチングアルゴリズム Rete¹³⁾ の特性を考慮したものである。

また、修正、再解析を繰り返すことが設計作業の特徴であるが、解析処理は数十秒のオーダーであり実用上は一層の高速化が必要である。一方、合成処理は最後に一度実行するだけであるから、数分オーダーでも実用上問題がないと考える。

7. おわりに

本論文では、機能・論理設計より高位の仕様設計を支援するためのエキスパートシステムについて、1つの構成方法を示した。具体的には、ハードウェア要素に関する知識および SFL 動作記述に関する知識を知識ベース化することにより、①日本語による仕様記述の解析・評価、②機能・動作論理を合成する技術のプロトタイプについて述べた。

仕様情報抽出での意味解析、正当性検証には可読性、保守性の高さから ES 構築支援ツール KBMS のルール (251 個) を用いた。

また、SFL 合成では、仕様記述の基本要素であるオブジェクトの分類、SFL のみで必要な情報の補完にルール (96 個) を用いた。

この論文で述べた手法は、ユーザに使い慣れた日本語を用いてハードウェアの設計を行うことができ、筆者らが狙いとしている仕様設計の段階からの支援を進めていく上で有効な方法であると考えられる。

日本語仕様記述
単純計算機は、命令レジスタと命令実行部から成り、演算器とメモリとプログラムカウンタを持つ。

命令フェッチ部は、レジスタの内容が2バイト命令である場合にオペランドをメモリのプログラムカウンタで示すアドレスから読み出し、プログラムカウンタへ書き込み、命令実行部を起動し、そうでない場合には命令実行部を起動する。

```

SFL出力
-----
module MODULE236 {
  circuit_class {
    input IN251;
    output OUT252;
    instrin DECODE;

    instruct DECODE(IN251)
      OUT252=IN251<7>;

    reg REGISTER217, REGISTER216, REGISTER215,
      REGISTER214, REGISTER213;

    DECODE245 DECODE245;
    INCREMENTER INCREMENTER273, INCREMENTER263;
    MEMORY MEMORY;
    ALU ALU;

    stage STAGE235 {      命令フェッチ部
      task TASK249;
      state STATE284(状態284)
      par {
        REGISTER217:=INCREMENTER273.INCREMENT(REGISTER217);
        REGISTER215:=MEMORY.READ(REGISTER217);
        goto STATE285;
      }
      state STATE285(状態285)
      par {
        any {      成立するものはどれも
          DECODE245.DECODE=OUT252(REGISTER215); --'コード'の7ビット
          par {      が立っている時
            generate STAGE234.TASK250;
            REGISTER214:=MEMORY.READ(REGISTER217);
            REGISTER217:=INCREMENTER273.
              INCREMENT(REGISTER217);
          }
          else : generate STAGE234.TASK250;
        }
        goto STATE284;
      }
    }
    stage STAGE234 {      命令実行部
  }
}

```

SFL記述内	仕様記述内	対応表
STAGE235	命令フェッチ部	
TASK249	命令実行部	
REGISTER217	プログラムカウンタ	
REGISTER214	レジスタ	
REGISTER215	レジスタ	
REGISTER216	レジスタ	
REGISTER213	レジスタ	
INCREMENTER273	演算器	
INCREMENTER263	演算器	
MEMORY	メモリ	
ALU	演算器	

図 14 単純計算機の実行例
Fig. 14 Synthesizer output.

今後の課題として、より良いマン・マシンインタフェースの実現を目指して、

- ① 制限を有する日本語を使用するため、ユーザの思った記述ができない可能性があるため、動詞のキーワードから使用できる文例を表示、入力を援助するアシスト機能の実現、
- ② 仕様抽出処理での矛盾検出時の的確なユーザへの助言（どこがどのような理由で矛盾しているかの指示等）、
- ③ 従来の設計を効率よく利用するためのライブラリの構成法、
- ④ 動作論理合成の自動化率拡大のための設計知識の抽出法、

について、引続き検討を進めて行く。

謝辞 日頃のご指導、ご支援に対し、NTT 情報通信処理研究所 知識処理研究部 村上国男部長に感謝致

します。また、本論文のご討論に対し、同 仲西秀基主任研究員に感謝致します。

参 考 文 献

- 1) Breuer, M., Friedman, A. and Iosupovicz, A.: A Survey of the State of the Art of Design Automation, *Computer*, pp. 58-75 (Oct. 1981).
- 2) Nakamura, Y., Oguri, K., Nakanishi, H. and Nomura, R.: An RTL Behavioral Description Based Logic Design CAD System with Synthesis Capability, *Proc. 7th International Conference on Computer Hardware Description Languages and their Applications (IFIP CHDL 85)*, pp. 64-78 (1985).
- 3) Oguri, K., Nakamura, Y. and Nomura, R.: Evaluation of Behavior Description Based CAD System Used in Prolog Machine Logic Design, *Proc. of the International Conference on*

- Computer Aided Design (ICCAD-86)*, pp. 116-119 (Nov. 1986).
- 4) Nakamura, Y.: An Integrated Logic Design Environment Based on Behavioral Description, *IEEE Trans. CAD*, Vol. CAD-6, No. 3, pp. 322-336 (1987).
 - 5) Nakamura, Y. and Oguri, K.: An RTL Logic Design Aid for Parallel Control VLSI Processors, *Proc. of the 4th International Conference on VLSI (IFIP VLSI 87)*, pp. 13-28 (1987).
 - 6) 中村, 小栗, 野村: ハードウェア動作記述言語 SFL, 第 29 回情報処理学会全国大会論文集, pp. 1737-1738 (1984).
 - 7) 仲西, 打橋, 小栗, 中村: ハードウェア向き仕様記述システム, 第 32 回情報処理学会全国大会論文集, pp. 1909-1910 (1986).
 - 8) Marwedel, P.: The MIMOLA Design System: Tools for the Design of Digital Processors, *Proc. Design Automation Conf.*, pp. 587-593 (1984).
 - 9) De Mann, H., Pabaey, J., Six, P. and Cleasen, L.: Cathedral-II: A Silicon Compiler for Digital Signal Processing, *IEEE Design & Test of Computers*, pp. 13-25 (Dec. 1986).
 - 10) Samad, T. and Director, W.: Towards a Natural Language Interface for CAD, *IEEE 22nd DAC*, pp. 2-8 (1985).
 - 11) 服部, 清水, 土屋, 桑原, 和佐野: 知識ベース管理システム (KBMS), 情報処理学会, 知識工学会と人工知能, 41-6 (1985).
 - 12) Goldstein, I. P. and Robert, B.: Using Frame in Scheduling, *Artificial Intelligence: an MIT Perspective Vol. 1* (1979).
 - 13) Forgy, C.: Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, *Artif. Intell.*, Vol. 19, pp. 17-37 (1982).

(昭和 63 年 8 月 17 日受付)
(平成 元年 3 月 7 日採録)



中村 行宏 (正会員)

昭和 42 年京都大学工学部数理工学科卒業。昭和 44 年同大学大学院修士課程修了。同年日本電信電話公社入社。現在 NTT 情報通信処理研究所知識処理研究部研究グループリーダー。DIPS 論理装置の研究開発を経て、ハードウェア設計支援技術の研究に従事。電子情報通信学会, IEEE 各会員。



雷下 充輝 (正会員)

昭和 34 年生。昭和 56 年東北大学工学部電気学科卒業。同年日本電信電話公社入社。以来、同社電気通信研究所において、大規模辞書設計法, LSI 論理設計, プロセッサ方式設計の研究に従事。現在 NTT 情報通信処理研究所知識処理研究部研究主任。



打橋 知孝 (正会員)

昭和 38 年生。昭和 60 年早稲田大学理工学部電子通信学科卒業。同年日本電信電話株式会社入社。情報通信処理研究所にてプロセッサ方式設計, 知識獲得の研究に従事。人工知能学会会員。



小栗 清 (正会員)

昭和 26 年生。昭和 49 年九州大学理学部物理学卒業。昭和 51 年同大学院修士課程修了。同年日本電信電話公社入社。同社電気通信研究所において、DIPS アーキテクチャ, 論理装置などの研究, 設計開発を経て、ハードウェア設計支援技術の研究に従事。現在、NTT 情報通信処理研究所知識処理研究部主幹研究員。1987 年第 2 回元岡賞受賞。電子情報通信学会会員。