知識工学と人工知能　69－2
（１９９０．３．２２）

部品合成による日本語からのLisp関数自動生成

水野　良彦・重永　実
山梨大学工学部

本研究では，仕様文中に出現する述語が１つのプログラムとなることに着目して，部品を合成することにより自動的にプログラムに変換することを試みている．入力された日本語仕様文に対して，構文解析，意味解析を行い中間表現に変換し，プログラム化に必要な情報を補う．次に，部品データベースより類似部品を同定し，それらを合成することにより目的プログラムを得る．この時，不都合箇所があれば，プロダクションルールにより中間表現を変更し，処理を続けていく．最後に，目的プログラムを部品化し，新しい部品となれる場合には，ここで部品化したものを，部品データベースに登録し，次回からはこの部品を用いることができるようにする．

AUTOMATIC PROGRAMMING OF LISP FUNCTION BASED ON PARTS COMPOSITION THROUGH JAPANESE SPECIFICATION

Yoshihiko MIZUNO and Minoru SHIGENAGA

Faculty of Engineering, Yamanashi University
Takeda-4, Kofu 400, Japan

A method of automatic programming of Lisp function through Japanese specification sentences is described. At first an input specification sentence is analyzed syntactically and semantically, and transformed into an intermediate representation which is a semantic network consisting of some partial specifications of component programs. Then similar network to the whole intermediate representation or its component networks are searched in a data base which has component programs and their intermediate representations, and an object program is synthesized using component programs. If any appropriate network is not found, the intermediate representation is altered by production rules. The newly synthesized program is registered into the data base and used as a part of component program.

1．はじめに

近年，ハードウェアの進化などに伴って，ソフトウェアに対する要求も多様化・複雑化してきた．このことに対処するために，過去に作成したことのあるソフトウェアを多少の変更で，組み合わせることができれば生産性や信頼性の向上などが期待できる．

一方，我々が仕様を渡されて，それをプログラムに変更するときのことを考えてみる．この時に行なう過程としては，考え易いように一度中間目標を立てて仕様を簡単なプログラムに分割し，既に持っているプログラミング知識や，他の文献に記載されているプログラム断片との間で類似性を検証して，それを組み合わせることによりプログラム化していく．また，この時に作成したプログラムは新たな知識として記憶されていく．このことは，前出のソフトウェアの多様化・複雑化を解決する手段と類似している．

現在まで行なわれてきた自動プログラミングに関する内外の研究[1][2]は，新たにプログラムを作り出すことに重点がおかれている研究が多かった．本稿では，上記のような我々が行なう過程を計算機に代行させること，すなわち，既にあるものを合成してプログラムを作り上げることを目的としている．

なお，システムが解くことのできる問題の難易度はLisp入門書[3]に出てくる例題や練習問題程度を目指している．問題の与え方は，日本語の仕様文を与え，入力仕様の目的に見合ったLisp関数を出力する．

2．部品を利用したプログラム合成

プログラムを合成による部品の表現方法として，どのような処理をする部品であるかという情報のほかに，大別して次の２つが考えられる[4]．

Black box：入力仕様を満たすために，部品内部のアルゴリズムに対して手を加えず，先行する部品と後続する部品の入出力情報だけを利用して組み合わせる．

White box：入力仕様を満たすために，部品内部情報であるアルゴリズムを変更，補充して組み合わせる．

Black box部品では，部品をそのまま用いるので抽象度が低く，ある限られた範囲でしか適用できないが，自動化の時には入出力に関する情報だけを用いればよいので，自動プログラミングシステムを実現しやすくなる．よって，本システムでは，Black box部品を中心に用いている．

また，一般に，仕様文を日本語で表現したものは，プログラムにするときに必要な項目を満たしていることがほとんど無い．このため，入力仕様文をそのままプログラム化することは難しい．このことに対処するためには，入力仕様文が表現していることを洗い出し，不確定な情報や不必要な情報を取捨選択しなければならない．

2.1 仕様文と中間表現

仕様文中に現れる述語(動詞，形容詞)は，プログラムの振舞を１対１に表現している場合が多い．このことに着目して述語ごとに類似部品を同定し，これらの部品を合成すれば目的プログラムを得ることが期待できる．また，原則として，日本語文の接続関係は，そのままプログラムの接続関係に影響を与える．

一方，日本語による仕様文に見られるプログラムの入出力を表す単語には，表2-1 に示すような特徴がある．

これらの特徴を反映するために，一度入力仕様文の述語ごとに格構造表現を生成し，同音異義語の中からの意味の決定，プログラムにしたときの接続関係，入出力に関連する単語の決定などを行っている．

しかしながら，格構造では，同じ意味を持つ仕様文で同一の格構造を得ることができない場合がある．例えば，図2-1 に示すような仕様文では，同じ意味を持つが，いずれも格構造表現が異なる．

表2-1　日本語における入出力単語の特徴

出力	主述語の任意の格 (ほとんどの場合が対象格である)
入力	①主述語の格構造の一部で出力とは別の格
	②出力と同じ格で出力単語に対する修飾語

① リストから　先頭の　要素を　取り出す
② リストの　先頭から　要素を　取り出す
③ リストの　先頭の　要素を　取り出す

波線：起点格，　実線：対象格，　波線：述語

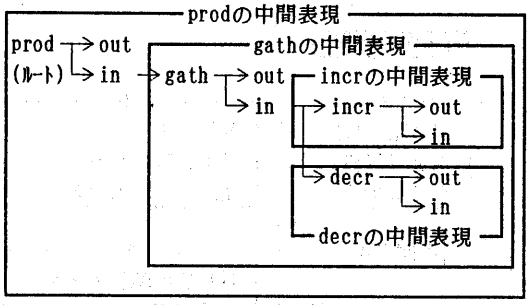
図2-1　日本語仕様文の持つ意味と格構造

①を基準に考えれば，②では起点格を構成している句の数が違うし，③では起点格がない．このため，格構造をそのまま用いれば，意味的には同じであるにも関わらず，異なる表現になってしまう．部品の処理概要を表すためには，入力仕様文の表現方法にはよらず，同じ意味を持つものに対しては，常に同じ表現で記述されていなければならない．

以上の点を満足するために，述語をプログラムに変換したときの振舞ごとにグループ化し，そのグループごとにフレーム構造状の述語フレームを用意した．この述語フレームにあるスロットやファシットを格構造や単語辞書などを利用して埋めることにより，表現が違うが意味的に見れば同じものは，同一の表現で表すことができるようにしている．本システムでは，この述語フレームのスロットをすべて埋めたものを中間表現として，部品の同定，部品の合成，目的プログラムの部品化を行っている．この述語フレームの種類を次の表2-2 に，中間表現の構成例を図2-2 に示す．

表2-2　述語フレームの種類[5]

	種類	フレームの持つ意味	主な述語
状態	conn (CONNect)	対象をある基準または別のものと比較して，どのような関係にあるかを表す	数える 判定する
	have (HAVE)	対象があるものの一部となっているか，または所有している状態にあることを表す	持つ 含む
	exis (EXISt)	対象がある場所に存在することを表す	ある
動作	incr (INCReace)	対象が何らかの作用を受けるか添加されることにより，出現または増加することを表す	付け足す 取り出す
	decr (DECReace)	対象が何らかの作用を受けるか削除されることにより，消滅または減少することを表す	取り除く 削除する
	sepa (SEPArate)	対象が何らかの作用を受け，複数の集団に分かれることを表す	分解する
	gath (GATHer)	対象が何らかの作用を受け，まとまった集団になることを表す	まとめる
	move (MOVE)	対象の一部または全体が，現在ある位置から別の位置に移ることを表す	複写する
	tran (TRANslate)	対象が何らかの作用を受け，現在ある状態から別の状態になることを表す	反転する 置き換える
	prod (PRODuct)	結果として，対象を作り出す働きを表す	作る，求める



	入力	出力
prod	gathの出力	プログラムの出力
gath	incrとdecrの出力	gathした結果(gathのout)
incr	プログラムの入力	incrした結果(incrのout)
decr	プログラムの入力	decrした結果(decrのout)

図2-2　中間表現の構成例

表2-3　係り受け関係アークの種類

記号	記述	意味	例
onew	A←onew←B	AとBで１つの単語となる	数値 アトム 連想 リスト
math	A←math←B	AがBを受け数学的な意味を表す	最小 公倍数 2次 方程式
unit	A←unit←B	AとBで数量を表す単位となる	1 個 2 つ
spec	A←spec←B	BがAを制限する	先頭の 要素
prop	A←prop←B	BがAの属性を表す	リストの 長さ 要素の 値
have	A←have←B	BがAを所有している	リストの 要素 数列の 項
arra	A←arra←B	AとBが同格である	3と 5
funv	A←funv←B	AとBで関数の値を表す	2の 階乗

中間表現の入力と出力の句には，仕様文の係り受け関係解析の結果，前ページの表2-3 に示すようなアークで結合することにより意味を補う．

５節の中間表現の入力や出力を指し示すアークはすべて表2-3 の意味を持っている．

なお，述語フレームには，一般的な情報しか記述してない．このため，ある述語固有の情報に対しては，その述語の単語辞書中に記述することにより補う．

２．２ 部品の同定と合成

部品の同定は，部品データベース内より同じ中間表現で表されている部品を検索してくることにより行っている．類似部品であるための条件としては，

① 述語フレームが同じものを用いている
② 中間表現の構成が同じである

の２つを最低限要求している．このときに，複数の部品が候補として検索された場合，どのようにして最適な部品を決定するかということが問題となる．ここでは，このような問題を回避するために，目的プログラムを部品化して部品データベースに登録するときに，同じ中間表現で記述できる部品が既に存在するときには，目的プログラム部品の登録を棄却することにより，複数の部品が候補として上がることがないようにしている．

以上のこととは逆に，同定する部品がない場合がある．このような場合には，基本的にプログラムを合成することはできないが，プログラムに影響を与える抽象名詞(各，ｎ番目など)が原因となっているときには，部品合成部に処理を任せる．

部品の合成は，ある部品の出力が後続する部品の入力として妥当なものであるかどうかを調べながら合成していく．最後まで，矛盾なく部品を合成することができれば，入力仕様に対する目的プログラムを得ることができる．

２．３ プログラムの部品化

プログラムを部品化して，部品データベースに登録しておくための表現方式として，目的プログラムの中間表現上で関数の定義部分に該当する中間表現を取り除くことによって行っている．また，部品の入力は中間表現上で一番末端に位置する中間表現の入力をあて，部品の出力は始点となる中間表現の出力をあてている．この２つの情報を目的プログラムの入出力情報としている．

関数定義部分を取り除いた中間表現で部品データベース内にある中間表現と照合を取り，すべての中間表現と照合が取れない場合には，目的プログラムが新たな部品となり得ると見なすことができ，部品データベースに登録する．このとき部品データベースに登録する内容は，関数定義部分を取り除いた中間表現，入出力情報，目的プログラムのソースリストの以上３つである．

なお，以上説明したことの具体的な処理内容については，５．プログラム合成部を参照されたい．

３．システムの構成

本システムは，すべてFranz Lisp(opus43.1)で記述してある．本システムは大きく分けて，システムと，システムに付随する情報源(辞書，ルールなど)により構成されている．この様子を図3-1 に示す．

この他にも，名詞や動詞などの意味的な階層関係を表したデータや，あるプログラムを１つの名詞で表したらどのような名詞単語が該当するかを示したデータなどを補助的な情報源として用いている．

部品データベースには，一般的に用いるLispシステムの持っている関数を選び出し，その仕様文に対する

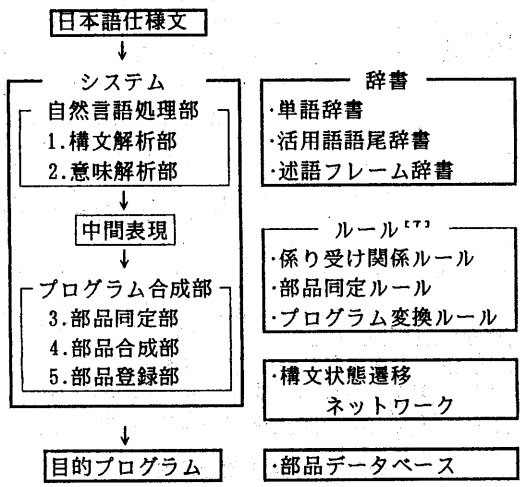


図3-1 システムの構成

中間表現を部品の中間表現として登録してある．部品として登録してある内容は，入出力の情報（部品に必要な引数の数と属性，出力の属性 ），関数の型，部品の中間表現，部品本体を１つのリストとしている．このリストは，入出力情報と関数の型を対リストにしたものをタグとした連想リスト構造を取っている．

また，部品データベース内では，このタグを用いて部品をグループ化している．

関数の型の種類と初期登録部品の数を次の表5-1 に示す．

４．自然言語処理部

４．１ 構文解析部[2]

構文解析部では，ローマ字によって分かち書きされた入力仕様文を句単位(文節とほぼ同じ)ごとに解析することにより句表現を生成する．句表現は，句の隣接関係を表した構文状態遷移ネットワークを文の始めから終わりまでたどった結果である．次に，この句表現の係り受け関係情報を用いて，入力仕様文を格と述語に分割する．最後に格と述語との間にどのような構文関係があるかを調べ構文木を出力し，意味解析部に処理を移す．

句表現や構文木は，複数個得られる可能性もあるが，意味解析やプログラム合成を行うことにより，最終的に目的プログラムが得られるまで，順次処理される．

このときに用いる情報源としては，単語辞書，活用語語尾辞書，構文状態遷移ネットワークがある．

４．２ 意味解析部

構文解析部の結果である句表現と構文木を用いて，まず格構造を生成する．本システムでは，格構造を生成することにより，入力仕様文の意味的な整合を計るとともに，プログラムにしたときの入力と出力に該当する単語を割り出している．次に，その格構造に対応する述語フレームにあるスロットなどを埋めることにより，図2-2 のような中間表現を生成する．

このときに用いる情報源としては，単語辞書(約250単語)，活用語語尾辞書(500程度)，述語フレーム辞書(40程度)，係り受け関係ルール(50例程度)がある．

５．プログラム合成部

５．１ 部品同定部

我々がプログラムを作成するときには，現在与えられている仕様が，自分の持っている知識の中で，もっとも多く重なり合ったものを得ようと思案している．

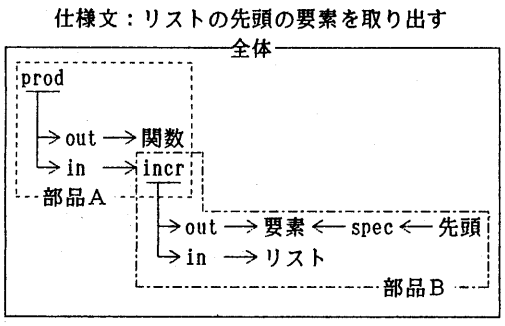
表5-1　関数の型の種類[6]

分類記号	分類基準	Lispによる関数例	初期部品の数
f	関数である		
→ds	リスト･シンボル用関数		
→ds_l	リスト用関数		
→l_c	リストを生成する	cons, list, append	3
→l_p	リスト用の述語	listp, length	2
→l_a	リストにアクセスする	car, cdr	2
→l_m	リストを直接操作する	rplaca, rplacd	2
→ds_s	シンボル用関数		
→s_c	シンボルを生成する	concat	1
→s_p	シンボル用の述語	atom, boundp, stringp	4
→s_a	シンボルにアクセスする	getd	1
→s_m	シンボルを直接操作する	set, explode, putprop	6
→ds_p	リスト･アトム用の述語	equal, null, member	3
→ds_etc	その他のリスト･アトム用関数	quote	1
→ma	算術関数	add, diff, minus	4
→ma_p	算術述語	numberp, zerop, lessp	5
→ma_etc	その他の算術関数	abs, mod, expt, sqrt	4
→sf	制御を司る関数	cond, mapcar, defun	5
→io	入出力関数	princ, read, terpri	6
n	関数とならないもの		

本システムもこのことを模倣して，部品の中間表現の規模がなるべく大きくなるようにして部品を同定している．このとき，原則として，入力仕様の中間表現をＳ，部品の中間表現をＰとすると，ＳがＰを包含していなければならない．

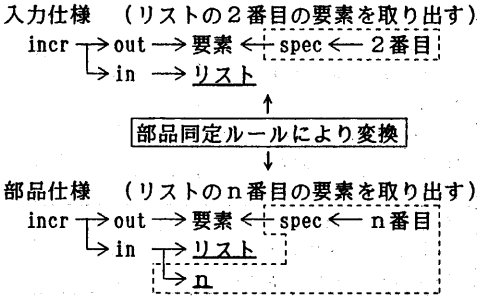
図5-1 において，始めprod中間表現とincr中間表現の両方を含んだ部品がないかを部品データベース内に探しにいくが，初期状態の部品データベースにはこのような部品がない．したがって，入力仕様に対する中間表現全体の部品同定は失敗する．次に中間表現の始点となる中間表現からtop-down的に部品を同定する．

部品によっては，引数の数が違うが処理概要から見れば，同じであると見なすことができる場合がある．このような場合に対処するために，部品同定ルールを設けている．部品同定ルールはプロダクションルール形式に記述してあり，条件部にある述語関数がすべて非ｎｉｌであった場合だけ，行動部にあるような関数が類似部品として同定される．



部品Ａ　（関数を定義する）
prod ┬→out ⟶関数
　　└→in ⟶動作を表す述語フレーム
部品Ｂ　（リストから先頭の要素を取り出す）
incr ┬→out ⟶要素 ⟵spec ⟵先頭
　　└→in ⟶リスト

図5-1　中間表現と類似部品



※　下線部がそれぞれの中間表現での入力

図5-2　部品同定ルールの適用

図5-2 で，部品仕様の内容が入力仕様の内容を含んでいることは明白である．すなわち，部品仕様では，数字変数「ｎ」のため引数の数が増えているにすぎない．上の例では，部品仕様に入力仕様の具体的な値である「２」を代入して，それを入力仕様の類似部品として同定している．

部品同定部の処理で用いる情報源には，単語辞書，部品同定ルール(50例程度)，部品データベースがある．

また，部品同定部の出力としては，中間表現に出現する順番で，同定した部品を１つのリストにした類似部品リストである．中間表現の入出力部分にくる形態によっては，部品同定ルールも適用することができないため，類似部品が同定できない場合がある．このような場合には，同定した部品の代わりに，その中間表現を類似部品リストの中に入れ，次の部品合成部に処理を任せる．

５．２ 部品合成部

目的プログラムへの変換は，類似リストにある順序を反転して，すなわち，中間表現からみれば末端に当たる中間表現からbottom-up的にプログラム化していく．これは，ある部品の出力が後続する部品の入力となり得るかどうかを調べながらプログラムを合成していくためである．

図5-1 の例では，先に部品Ｂがプログラム実体となり，次に部品Ａのプログラム実体に部品Ｂを埋め込むことにより，次のような最終的な目的プログラムを得ることができる．

類似部品リスト中に中間表現が現れた場合には，プログラム変換ルールによってプログラム化できるかどうかを調べる．プログラム変換ルールも，部品同定ルールと同様にプロダクションルール形式のルールで，当てはめることができるルールがあれば，それを用いてプログラム化する．

図5-4は，リストの「各要素に～する」というmap関数を生成するプログラム変換ルールの適用例である．

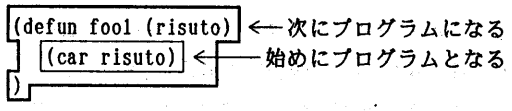


図5-3　プログラムの合成

仕様文：リストの各要素に２を足したリストを求める

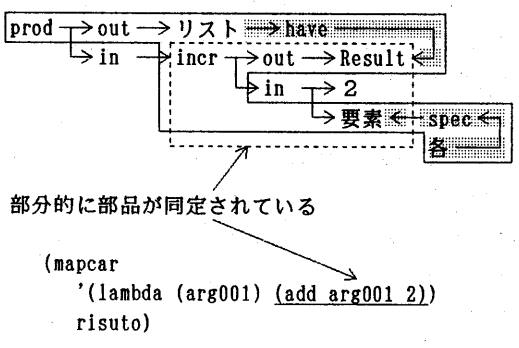


図5-4　プログラム変換ルールの適用

incr中間表現は，部品同定部により部分的に足し算を行う部品と同定されている(図5-4の点線で囲んだ部分). しかし，まだspecアークとhaveアーク部分が部品として表現されずに残っている(図5-4の網がけ部分).

この文を構文解析すると抽象名詞「各」は，「足す」の格を修飾する名詞句とされるが，プログラムにしたときには，自分の属している中間表現より上位のprod中間表現と一緒になってmapcar関数の構成要因となっている(図5-4 の実線で囲んだ部分). 本システムでは，中間表現を１つの処理単位と考え，部品を同定するため，他の中間表現に現れる名詞単語と共同して１つのプログラムを形成している場合に対応できないが，プログラム変換ルールを用いることによって，このような問題を解決している.

プログラム変換ルールが適用できない場合には，システムの処理は，自然言語処理部の構文解析部まで戻り，まだ処理していない句表現と構文木の組がないかを調べる. もし，他の句表現と構文木の組が存在しない場合には，入力仕様に対する目的プログラムを得ることはできない.

以上の処理で，用いている情報源は，単語辞書，プログラム変換ルール(50例程度)，部品データベースがある. また，部品合成部の出力としては，入力仕様に対応する目的プログラムである.

５.３ 部品登録部

前節のプログラム合成部までの処理で，入力仕様に対する目的プログラムを得ることができたが，次に，このプログラムを再利用するときのこと(部品の同定)を考えて，目的プログラムを部品化し，今までにない新しい部品であれば，部品データベース内に登録しなければならない.

本システムは，部品をblack box部品として扱っているので，部品の処理概要と，入出力の情報を部品データベース内に登録することによってプログラムを部品化する. 部品の処理概要は入力仕様の中間表現をそのまま用いるが，入出力情報は，関数定義部分を取り除いた中間表現で始点に位置する中間表現の出力を部品の出力とし，終点に位置する中間表現の入力すべてを部品に対する入力としている(図5-5参照).

部品データベースに登録する情報は，入力仕様の中間表現と入出力情報の他に，部品をLispシステム上で実行するときのことを考えて，部品のソースコードが必要となる.

部品化した結果，新しい部品である場合には，部品同定ルールをあてはめてみて，得られた中間表現が部品データベース内に登録されていないかどうかを調べる. これは，複数の部品が類似部品として上がることを防ぐために行うものである.

入力仕様によっては，部品データベース内にある部品にある部品の処理をより一般化したものがある.

次ページの図5-6 では，明らかに仕様２の方が一般的である. 先に仕様２を部品として持っているならば，

仕様文：リストの先頭に要素を付け足す

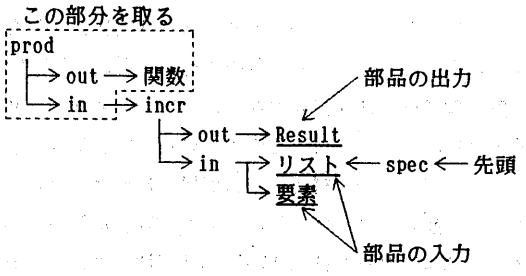


図5-5　プログラムの部品化

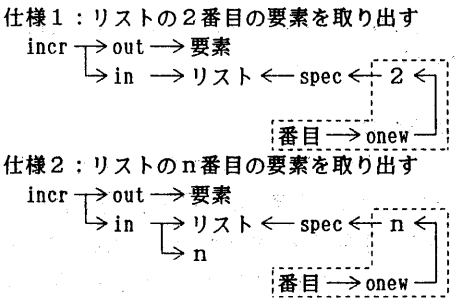


図5-6　中間表現の抽象化

次に仕様１がきても問題はない．しかし，先に仕様１を部品として持っていて，次に仕様２がきた場合には，先の部品同定ルールが適用でき問題が生じる．ここで，図5-6 で点線で囲んだ部分を見れば，仕様２の中間表現は数字変数がくることを意味し，引数の意味や個数を考えても仕様１の中間表現を含んでいると見なすことができる．

このように，部品仕様の中間表現を入力仕様の中間表現が変数を用いて含んでいて，その分だけ引数の数が多くなっている場合には，今まで部品データベースに登録してあった部品を棄却して，新しい，より抽象化された入力仕様の中間表現と部品を登録することにより，部品データベース内での抽象化を行っている．

６．例題

以下に，平均値のプログラムを合成することにより，本システムを用いた例を示す．

６．１ 平均値の問題

・総和の定義

平均値を求めるには，まずデータの総和を求めなくてはならない．このために，総和を求める関数を定義しておく必要がある．

図6-1 において，「作る」の対象格では関数に足した結果がすべて係っている．このため，「関数」と「足す」の間の関係は，何か行動を伴った結果であるという意味でactionアークによって両者を結合している．

一方，「足す」の格構造は，起点格と対象格が存在している．起点格の方は，「リスト」が「最初」に係っているが，起点となり句は場所的な意味を持っていなければならない．ここでは，「リスト」に位置を指定する句「最初」が係るようにアークの指す句を変更している．このため，「リスト」と「最初」はspecアークで結合される．対象格の方は，「最後」が「要素」に係っている．ここにある係り受け関係は，「最後」が「要素」を指定しているため，両者はspecアークで結合される．

なお，アークの種類の決定は，プロダクションルールで記述された係り受け関係ルールを用いている．

次に，図6-1で生成した格構造を用いて，図6-2のような中間表現を生成する．

図6-1 の格構造では，specアークで結ばれている最初と最後は意味的に考えてみると両者ともリストの要素の位置を指定しているものである．したがって，この両者をまとめて要素を指定するようにアークをつなぎ直す．また，要素はリストの構成要素であるのでこの両者は新たにhaveアークでつながれる(図6-2の点線で囲んだ部分)．

図6-1 ①の仕様文には，足した結果，何が出力されているか明記されていない．これは，一般的に算術演算を表す述語の格構造には，出力を表す単語を記述しないために起こる現象である．このため，中間表現上には，Resultを補いこの問題を解決している(図6-2の

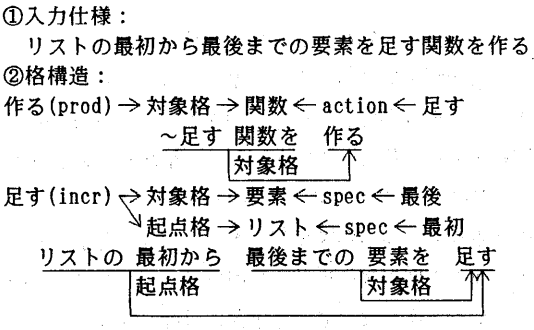


図6-1　総和の仕様と格構造

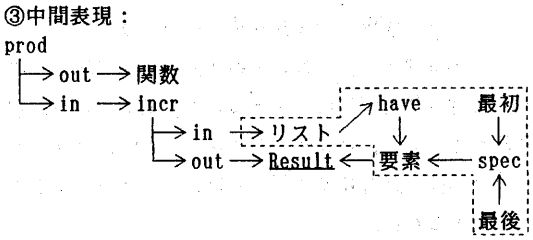


図6-2　総和の中間表現

下線部).

以上の処理は，述語フレーム辞書のスロットを埋めることにより行なわれ，図6-2 の中間表現を次のプログラム合成部に送る.

まず，部品の同定を行なう．prod中間表現に対する部品は，

(defun sf_ma (リスト) incr中間表現)

となる．incr中間表現の方は，部品データベースが初期登録状態であれば，このような関数は存在しない．したがって，同定される部品はなく，類似部品リストに中間表現がそのまま代入される.

部品合成部では，類似部品リストの内容を受け，プログラム変換ルールの条件部とincr中間表現で照合を取る．図6-2 では，リストの持っている要素を最初から最後まで足し，その結果を返すと解釈できる．よって，再帰関数を生成するプログラム変換ルールの中で算術的な部品addを用いるものと照合が取れ，最終的に図6-3のようなプログラムを出力する.

再帰する対象となる関数が算術的な意味を持つので，再帰終了条件としては，リストがnullであれば，０を返すことになる(図6-3 の下線部)．また，最初から最後までに対応する部分は，再帰呼出に対応し，car，cdr，自分自身の関数をincr中間表現より同定される部品add に合成してプログラム化される(図6-3で実線により囲んだ部分).

仕様文に対する目的プログラムができたときに，システムは，このプログラムの処理概要が１つの単語で言い表すことができるか聞いてくる．この場合には，総計，合計，総和などを入力する．この作業により，今後は「総和を求める」だけで，このプログラムを部品化したものを活用する．なお，この言い替えの情報は，ある単語が１つのプログラムにある場合を表したプログラム変換ルールの補助的な情報源として，その都度登録される.

```
(defun fool (リスト)
  (cond ((null リスト) 0)
        (t (add (car リスト)
                (fool (cdr リスト))))))
```

図6-3　総和のプログラム

・平均値の定義

次に，総和の部品を用いて平均値を求めるプログラムを合成する.

図6-4 の中間表現において算術的な特徴があるため，出力にResultを補っている．また，総数に係る句が省略しているとみなされ中間表現上では，データを補っている.

部品同定時において，部品quotientと部分的に部品が同定できるが(点線で囲んだ部分)，sepa中間表現の入力にあるfunvアークの照合が取れていない．funvアークはある関数を実行した値を示すアークであるので，プログラム変換ルールを用いて該当する部品をここに埋め込む．図6-4 では，総和が先に作った部品と，総数がリストの長さを数える部品lengthとを利用し，最終的に次のプログラムを合成する.

最後に，総和を定義したときと同じように，平均値，平均などの単語でプログラムの処理概要を言い替えてやることにより，次回からは，この平均値を求めるプログラムを部品として活用することができる.

平均値のように，数式で明示でき，それを利用して問題を解くような数学的な問題領域については，定義の与え方次第でほとんどの問題が解けると思う．しかしながら，それ以外の問題領域では，解ける問題と解けない問題がはっきりとしている．これは，主に部品が同定できない場合に用いるプログラム変換ルールがあるかどうかに起因している.

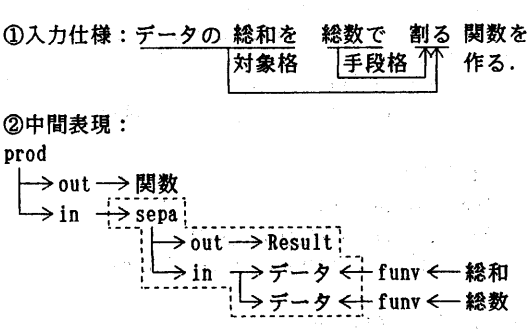


図6-4　平均値の仕様と中間表現

```
(defun fool (データ)
  (quotient (総和　データ) (総数　データ)))
```

図6-5　平均値のプログラム

6.2 n番目の問題

リストの各要素に対して何らかの処理を施したリストを返す関数については，図5-4 のmapcarを例に取り，すでに説明した．ここでは，リストを再帰的に用いる例として，n番目の要素に対して何か処理を行う問題を示す．例として，図5-2 の部品同定ルールの適用や図5-6の 部品データベース内での抽象化の説明に用いた「リストのn番目の要素を取り出す」という仕様文に対しての処理過程を示す．

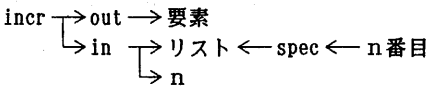
ここの例では，n＝1になったときに，取り出すという動作を行う．このためには，部品を同定するときに，ある要素を取り出すということで部品carが部分的に同定されていなければならない．残りの中間表現に対しては，再帰関数を生成するプログラム変換ルールによってプログラム化される．

プログラム変換ルールの適用条件としては，入力したリストのn番目の要素に対して何か動作を伴う処理をすることと，出力が入力したリストの要素であることが要求される．この時に，図6-6 の③に示すようなプログラムに変換される．なお，部分的に同定された部品は，再帰終了条件を満たしたときの行動になって現れている(図6-6の下線部)．

我々は，上記のような再帰関数にするようなプログラミング方法は，主に経験から得ているが，本システムでは，このようなプログラム変換ルールを持つことによって対応している．

①入力仕様：リストの n番目の 要素を 取り出す
（対象格）

②中間表現



③プログラム

```
(defun fool (n risuto)
  (cond ((equal n 1) (car risuto]
        (t (fool (sub1 n) (cdr risuto]
```

・プログラム変換ルール
　条件　1．リストのn番目に対して何かを行う
　　　　2．出力がその要素である
・同定部品
　部品ｃａｒ：リストのある要素を取り出す

図6-6 n番目の問題例(取り出す)

7．おわりに

本稿では，我々にとって一番身近な自然言語により，プログラムの仕様文を入力し，その目的プログラムを合成すると共に，合成したプログラムとその中間表現を，新しい部品として登録し，次のプログラム合成に用いることができるようにすることを目標としてきた．この特徴を活かし，使用者が仕様文の与える順序を考えて入力してやれば，徐々にシステムが解くことのできる問題が増えていくことになる．

プログラムへの変換には，各種プロダクションルールを用意して，新たな問題が生じても，プロダクションルールだけを追加することにより対応できる．

本システムが自動プログラミングの理想的なものであるとは言いがたく，このように呼べるようにするためには呼べるようにするには，

①複雑な問題になるにつれ仕様文だけで表すことが不可能となるため，日本語に代わる仕様提示方法が必要となる．

②全く新しい概念を，できるだけ使用者の負担とならないような概念獲得部が必要となる．

の２点が大きな問題となってくる．

＜参考文献＞

[1] A.Barr,E.A.Feigenbaum編,田中幸吉,淵一博監訳: 人工知能ハンドブック，共立出版，pp.447-518, Vol.2(1984)

[2] 菅原昌久，重永実：日本語による自動プログラミングシステム，情報処理学会研究会報告，AI64-8 (1989)

[3] 黒川利明：LISP入門，培風館(1982)

[4] 伊藤潔,本位田真一,内平直志：プロトタイピングツール，啓学出版(1987)

[5] 情報処理振興事業協会技術センタ編：計算機用日本語基本動詞辞書-初版-(1986)

[6] Franz Lisp Referance Manual Opus43.1

[7] R.Davis：Interactive Transfer of Expertise: Acquisition of New Inference Rules, Artificial Intelligence, Vol.12,No.2, pp.121-158(1979)