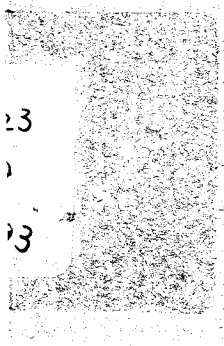
拡張型言語とその処理系に関する研究

山　本　米　雄



昭和４８年１２月

論 文 目 録　　大阪大学

報告番号・甲第1701号　　山本米雄

主論文　拡張型言語とその処理系に関する研究

（主論文のうち印刷公表したもの）

1. ダイナミックコンパイラシステムに関する一考察
   電子通信学会　電子計算機研究会資料
   昭和46年2月23日

1. DCS言語とその処理系
   電子通信学会　オートマトンと言語・パターン
   認識と学習研究会資料
   昭和47年7月5日

1. 拡張型文法の曖昧性について
   電子通信学会　オートマトンと言語・パターン
   認識と学習研究会資料
   昭和48年1月21日

1. ダイナミックコンパイラシステムとその評価
   電子通信学会　電子計算機研究会資料
   昭和48年6月21日

1. ＡＬＧＯＬ文法の一構成法
電子通信学会　オートマトンと言語研究会資料
昭和４８年９月２５日

1. 階層構造をもつコンパイラ・コンパイラとその構成法
電子通信学会　電子計算機研究会資料
昭和４８年１１月２２日

（主論文のうち未公表のもの）

1. 拡張型言語の一形式とその処理系－ダイナミックコンパイラシステム－
電子通信学会論文誌Ｄ　57巻３号
昭和４９年３月２５日　掲載予定

論文採録決定通知状

昭和48年11月22日

山本米雄 殿

〒105 東京都港区芝公園3-5-8
機械振興会館内
社団法人 電子通信学会
編集委員長

論文題名 拡張型言語の一形式とその処理系ダイナミックコンパイラシステム

ご寄稿いただきました上記論文（受付番号247，10分冊）は昭和48年11月20日に採録と決定いたしましたのでご通知申し上げます。

拡張型言語とその処理系に関する研究

山　本　米　雄

昭和４８年１２月

内　容　梗　概

本論文は筆者が大阪大学大学院工学研究科（通信工学専攻）在学中に行なった拡張型言語とその処理系に関する研究の成果をまとめたものであり，次の７章をもって構成する。

第１章は緒論であり，拡張型言語とその処理系に関するこれまでの研究のあらましを述べるとともに，本論文の占める位置を明らかにしている。

第２章では拡張型言語と自己拡張型コンパイラについての考察を行い，その実現の可能性について論及している。また従来の欠点をおぎない処理系まで考察し，文命令の拡張が行えるダイナミックコンパイラシステムを提案し，そのシステム構成について述べている。ダイナミックコンパイラシステムはコンパイラ・インタプリタでコンパイラを実時間で変更，拡張してゆく自己拡張型コンパイラシステムである。本章ではこのシステムの機能について述べている。

第３章では，ダイナミックコンパイラシステムの言語系について述べている。言語系はメタ言語，ダイナミック言語よりなる。メタ言語はダイナミック言語に新しく命令を付加する定義命令であり，ダイナミック言語は問題を記述する手続き向き言語であり，それらの構文，意味について述べている。

第４章では第３章で述べた言語系の無曖昧さについて述べている。すなわち拡張型言語では常に拡張に際して無曖昧さが要求される。そこで本章では筆者の提案した言語系が曖昧さを持たないことを証明し，この言語系が充分拡張型言語として使用し得ることを保証している。

第５章では処理系について述べている。処理方式，コンパイラ・インタプリタ，ダイナミックコンパイラの構成方針を示し，各フェイズをルーチンと

表に分割している．特にダイナミックコンパイラは表駆動型になっており，コンパイラ・インタプリタは表を変更作成することにより，ダイナミックコンパイラを拡張する．本章ではそれらの各ルーチン，表の構成について述べている．

第６章では第５章で述べた処理系の構成に基づき，NEAC2206を用いて試作実験した結果について述べ，その評価を行っている．処理系の評価には静的なステップ数と動的なステップ数を用いて本システムが十分有用であることを保証した．さらにダイナミックコンパイラシステム全体の検討，評価についても述べている．

第７章は結論であり，本研究により得られた諸結果を検討し，今後の方向について二，三言及している．

関　連　発　表　論　文

1）山本，打浪，手塚：「自己拡張型コンパイラに関する一考察」信学全国大会　936（昭45）

2）山本，打浪，手塚：「ダイナミックコンパイラシステムに関する一考察」信学会　電子計算機研資　EC70－48（1971－02）

3）山本，打浪，手塚：「ダイナミックコンパイラに関する一考察」信学全国大会　1023（昭46）

4）山本，打浪，手塚：「ダイナミックコンパイラシステムに関する一考察その2」信学会全国大会　1211（昭47）

5）海尻，山本，打浪，手塚：「ＤＣＳ言語とその処理系」信学会オートマトンと言語・パターン認識と学習研資ＡＬ72－41・ＰＲＬ72－39（1972－07）

6）山本，打浪，手塚：「拡張型文法の曖昧性について」信学会オートマトンと言語・パターン認識と学習研資ＡＬ72－98・ＰＲＬ72－39（1973－01）

7）山本，打浪，手塚：「ダイナミックコンパイラシステムにおける無曖昧文法について」信学会全国大会　1182（昭48）

8）海尻，山本，打浪，手塚：「ＤＣＳプロセッサの構成について」信学会全国大会　1386（昭48）

9）山本，海尻，妹尾，打浪，手塚：「ダイナミックコンパイラシステムとその評価」信学会　電子計算機研資　ＥＣ73－19（1973－06）

10）山本，打浪，手塚：「ＡＬＧＯＬ文法の一構成法」信学会　オートマトンと言語研資ＡＬ73－41（1973－09）

11） 海尻，山本，打浪，手塚：「コンパイラ記述言語システムの一方式」電気関係学会関西支部連合大会　G６－１　（昭48）

12） 妹尾，海尻，山本，打浪，手塚：「汎用命令を持つ，小型計算機用システム記述言語の一方式」電気関係学会関西支部連合大会　G６－２（昭48）

13） 海尻，山本，打浪，手塚：「階層構造を持つコンパイラ・コンパイラとその構成法」信学会　電子計算機研資　EC73－47（1973－11）

14） 山本，海尻，打浪，手塚：「拡張型言語の一形式とその処理系－ダイナミックコンパイラシステム－」信学会論文誌D（採録決定）

目　　次

第１章　緒　　論

高速大容量の大型電子計算機が発達普及し，計算機システムが複雑になりつつある．計算機システムが複雑になるにつれて，ハードウェアにましてソフトウェアの重要性が認識されてきた．ソフトウェアとは利用技術と訳されているが，それは人間と機械を結びつけるマン・マシンの通信手段である．この人間と機械における通信手段は，電子計算機が情報処理機械として重要な役割を果すにつれて活発に研究，開発されるようになった．しかし，大型汎用計算機の生産が遅れる大きな要因の一つがソフトウェアシステムの作成にあることからも，現在ソフトウェアの立遅れが目立っており，“ソフトウェア危機”が各所でいわれている．この“ソフトウェア危機”をいかに克服するかは，情報処理科学において当面する重大な問題の一つである．

ソフトウェアにおけるプログラミング言語についても，現在使用者の専門分野が細分化し，問題も多種多様になり，それに応じて，各問題向きの多くの計算機言語が研究開発されてきた．図１．１にそれら計算機言語の種類を示す．このように計算機言語の多様化に伴って，その処理系であるコンパイラをいかに作成するかが問題になる．

筆者は計算機言語として拡張型言語が将来この問題に対して大きな意義をもつものと考えている．すなわち一つの拡張型言語は

○　各問題領域に適した「自然な」形式で，データ構造，処理，記述が行なえる．

○　標準的な問題に対しては核言語および拡張するメタ言語を標準化できる．

という利点があり，個々の計算機言語を作成する必要がなくなる．

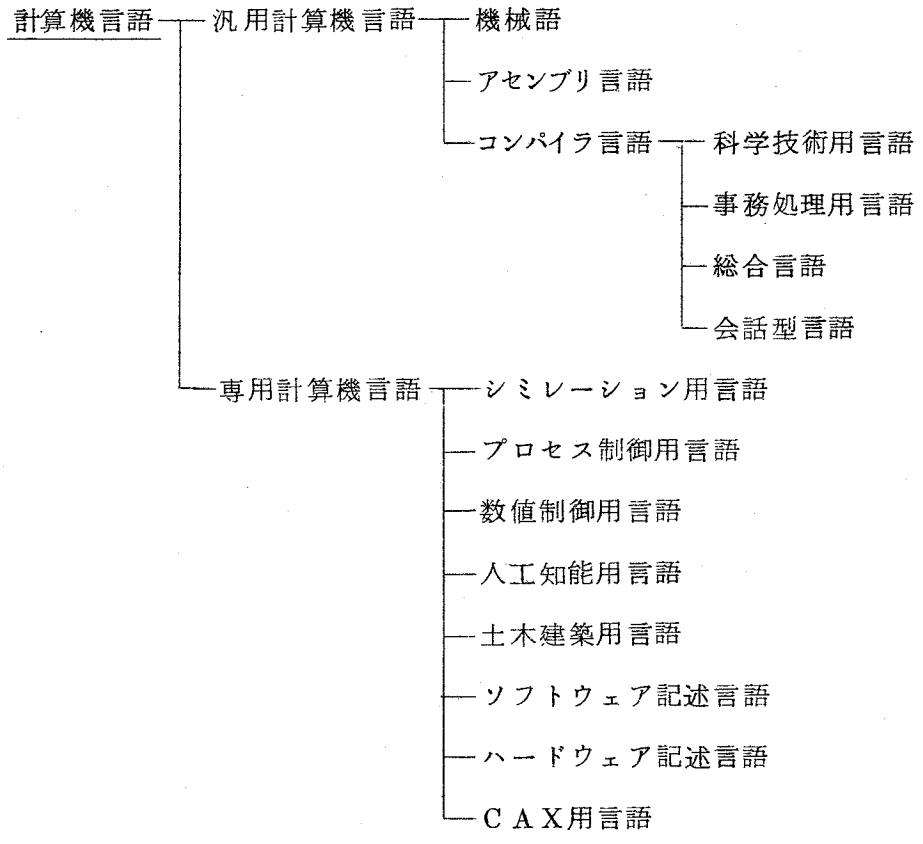


図１．１　計算機言語の種類

拡張型言語の研究はPL/I等の総合言語の反動，またはその経験の結果として１９６０年代の中頃から始まった．その研究段階は次の３つに分けられる．

1）第１段階はマクロ命令を用いての拡張であり，[1][2]既存の言語の文字面のはり合せである．

2）第２段階はALGOL60 に拡張性をもつように再編成したことである．[3]例えばALGOL６０の変形としてALGOL C−D[4], GPL[5], IMP[6] 等がこれである．

3）第３段階は拡張性に要求される構造，一般性，効率に非常に注意を払

って開発された全く新しい言語の作成である．その例としてはBASEL，PPL，GEDANKEN，ECL，PASCAL等[7]～[16]がある．これらの研究は７０年頃から始まった．筆者も７０年以来拡張型言語システムに着目し，文命令を構文的に拡張することを目的とし，言語とその処理系の両者にわたって研究してきた．処理系（コンパイラ）そのものを逐次的に拡張し，文命令を構文的に拡張する機能は他の上記の言語にはないようである．

筆者はＴＳＳの普及を考えて，端末から一般使用者がコンパイラの変更，拡張が行なえるシステム，すなわちオンライン・リアルタイムで端末から各自のコンパイラの使用と作成が同時に行なえるシステムの考察と検討を行ない，ダイナミックコンパイラシステム（ＤＣＳ）の提案を行なった．[17]～[19]このＤＣＳの入力言語系は拡張型言語系となる．本論文ではコンパイラを逐次的に拡張し，文命令を構文的に拡張するシステムについて考察，実験を行なっている．

ＤＣＳの処理系は従来のプリプロセッサを用いた方法（マクロによる拡張）[1][2]と異なり，図１．２に示すコンパイラ・インタプリタを用いる方式を採用した．これにより実時間でコンパイラを変更，拡張できる．

図１．２のコンパイラ・インタプリタの入力言語はコンパイラ記述言語とみなすことができる．一般にコンパイラ記述言語は

1）コンパイラそのものを記述する言語．

2）コンパイラの作成法を記述する言語．

3）コンパイラの仕様を記述する言語．

に分類できる．1）は機械語ないしはアセンブラ言語であって，コンパイラそのものがこの言語で記述されている．初期のコンパイラは直接この言語で

書かれた。しかしコンパイラが大きくなるとこの方法ではシステムが膨大なものとなり，ほとんどその作成が不可能になる。2）はＣＯＬ[20],[21] ＢＰＬ[22] 等のコンパイラ記述言語であり，コンパイラ・コンパイラ[23],[24] もこれに属す。コンパイラの作成法の記述は非常に複雑であり，一般使用者むきの言語ではない。一方3）は単にコンパイラの仕様を記述するための言語であって，記述，使用が比較的容易になり，一般使用者でも使用が可能になる。3）のコンパイラはコンパイラ・ジェネレータとみなすことができる。ＤＣＳは言語の構文と意味を入力して，出力としてその言語のコンパイラを出す，コンパイラ・ジェネレータである。

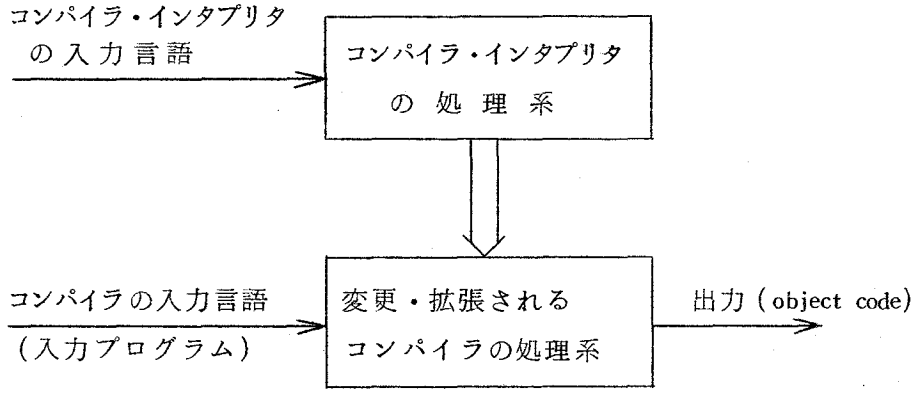


図１．２　コンパイラ・インタプリタと拡張されるコンパイラ

以上述べてきたように，ＤＣＳは入力言語としての拡張型言語システム，処理方式として自己拡張型コンパイラシステムの中のコンパイラ・インタプリタ方式，拡張能力としてのコンパイラ記述言語に対しての考察である。以下第２章では拡張型言語，自己拡張型コンパイラ，コンパイラ記述言語の各々に対して考察[25],[26] を行ない，筆者が提案したダイナミックコンパイラシステムの特徴を述べている。ＤＣＳはコンパイラ・インタプリタ（ＣＩ）

でもってダイナミックコンパイラ（ＤＣ）を実時間で拡張する自己拡張型コンパイラシステムである。

第３章ではＤＣＳの入力言語で言語系[27]すなわち拡張型言語系について述べる。言語系はメタ言語（ＭＬ），ダイナミック言語（ＤＬ）よりなり，ＭＬでもってＤＬに新しく文命令を構文的に定義する定義命令である。

拡張型言語系において，拡張される言語は拡張に際してその構文が常に無曖昧であることが要求される。書換規則が定義（付加）された場合の一般的な無曖昧さを論じた論文は現在みあたらない。第４章においてこの事を論じ，ＤＣＳにおいて行なった拡張に際して曖昧さが生じないことを保証する。[28],[29] また曖昧さが生じない算術式の構成法を述べる。

第５章においてはＤＣＳの処理系の構成[26],[27],[30]について述べる。ＤＣの構成は表駆動型のコンパイラにした。ＣＩはＤＣの各種表を変更，作成することにより拡張を行なう。

第６章では前章で述べた構成法に基づき，ＮＥＡＣ２２０６を用いて試作，実験を行なった結果と評価について述べる。[26],[31] 評価に際しては静的なステップ数と動的なステップ数を用い，本システムが十分有用であることを保証している。さらにＤＣＳ全体の評価，検討を行ない，今後の展望について述べている。

第７章では本論文で得た諸結果を検討し，今後の方向について述べている。

第 2 章　拡張型言語システム

2．1　緒　　言

拡張型言語システムは言算機言語システムとして次の３つの角度から論じることができる。

1）入力言語としての拡張型言語。

2）処理系としての自己拡張型コンパイラ。

3）拡張能力としてのコンパイラ記述言語。

本章において上記のそれぞれについて考察を行ない。筆者が提案したダイナミックコンパイラシステム（ＤＣＳ）を位置づける。最後にＤＣＳのシステム構成，機能について述べる。

2．2　拡張型言語，自己拡張型コンパイラ，コンパイラ記述言語

拡張型言語は広義には PL/I [32]，ＡＬＧＯＬ６８等[33]～[37] の総合計算機言語を含めるが一般には

1）核言語

2）拡張機能

の２つが存在して初めてその機能を果す。その機能とは次のⅰ～ⅳ）を拡張もしくは修正する機能である。

ⅰ）演算子

ⅱ）構文（文命令）

ⅲ）データ構造

ⅳ）制御機構

従来の拡張型言語をまとめると表２．１になる。[5]～[16],[38]

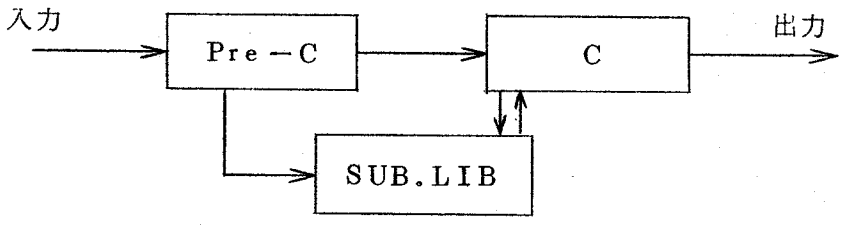
表２．１　拡張型言語の分類

	分　類	小　項　目		例
Ⅰ	定義できる対象による分類	1	演　算　子	（全　　て）
		2	文　命　令	ESDL，ALGOL N
		3	データ構造	EL1，PPL，ALGOL 68
		4	制　御　文	PPL，EL1
Ⅱ	拡張方式による分類	1	核（コア）方式	GPL
		2	殻（ジェル）方式	PL/I，ALGOL 68
Ⅲ	定義の方法による分類	1	一般使用者向き	EL1，PL/I，ALGOL 68
		2	システム・デザイナ向き	ESDL，SEL
Ⅳ	メタ言語による分類	1	アセンブリ言語	ESDL
		2	高水準言語	EL1，GPL，ALGOL N
Ⅴ	処理系による分類	1	プリプロセッサ（マクロ）	ESDL，ALGOL C−D
		2	プロセッサの変更	EL1，ALGOL N

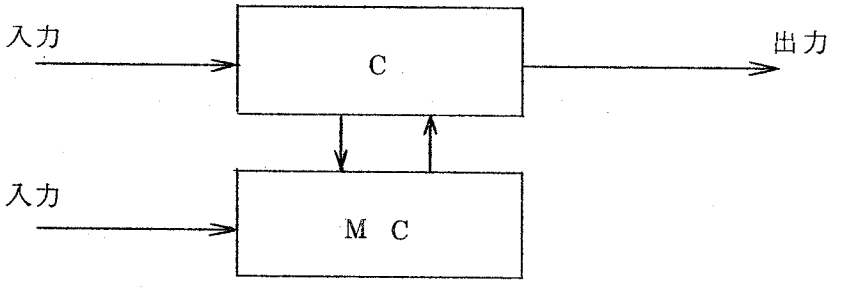
拡張型言語の処理系すなわちコンパイラは自己拡張型コンパイラとみなすことができる．自己拡張型コンパイラのモデルとして図２．１の型が考えられる．(17)

（a）　従来のC（コンパイラ）の前にPre−Cを付加するシステム．Pre−Cはサブルーチンを自動的に登録し，登録ずみの命令は実行時にPre−Cが自動的に呼びだす．

この方法はマクロ的な拡張であって，もとの処理系であるコンパイラ自体は変化しない．従ってみかけ上拡張されたコンパイラが拡張された言語を反映しない．



(a)



(b)

図２．１　自己拡張型コンパイラのクラス

(b)　拡張されるC（コンパイラ）と拡張するMC（メタコンパイラ）の二段に分けるシステム．MCは一定不変であり，Cの拡張，変更だけを行なう．

この方法によると(a)の方法と比し，コンパイラが拡張された言語を反映した処理系となる．(b)には構文(syntax)の拡張と意味(formal semantics)の拡張がある．後者の拡張はマイクロプログラミング等，命令と機械動作(machine action)の対応の変更，拡張の場合である．本論文は前者の場合を取扱っている．(b)では新しく付加する命令等は定義のみを与える

だけでよい．その時メタコンパイラは与えられた定義から処理プログラムを自動的に作成し，コンパイラを拡張する．そのためにはメタ表現と対象表現との変換処理機能と次に示すものが必要である．

i）新しい構文を定義する命令

ii）その定義を解釈しながら，新しく付加した構文の処理ルーチンを作成する機能．

iii）新しく付加した構文を解釈し，実行するようにコンパイラを拡張する機能．

iv）付加した構文がすでに存在する系と矛盾しないかを調べる機能．

筆者が提案した拡張型言語システム（ダイナミックコンパイラシステム，ＤＣＳ）は自己拡張型コンパイラとしては(b)の方法を採用し，特にＭＣとしてコンパイラ・インタプリタ（ＣＩ）を用いる．

一方コンパイラ記述言語として次の３つが考えられる．

1）コンパイラそのものを記述する言語．

2）コンパイラの作成法を記述する言語．

3）コンパイラの仕様を記述する言語．

1）は機械語ないしはアセンブラ言語であって，コンパイラそのものがこの言語で記述されている．初期のコンパイラは直接この言語で書かれた．しかしコンパイラが大きくなると，この方法ではシステムが膨大なものとなり，ほとんどその作成が不可能になる．2）はCOL[20],[21]，BPL[22]等のコンパイラ記述言語であり，コンパイラ・コンパイラもこれに属する．コンパイラの作成法の記述は非常に複雑であり，一般使用者向きの言語ではない．一方3）は単にコンパイラの仕様を記述するための言語であり，記述および使用が比較的容易になる．しかし処理系は複雑となり，すっきりとした形で

論じることはむずかしくなる。 3) の処理系は一般に図２．２のように表現でき，コンパイラ・ジェネレータとみなすことができる。また図２．２は言語Ａのコンパイラ・ジェネレータの中に含めて考えると，拡張型言語の処理系となる。

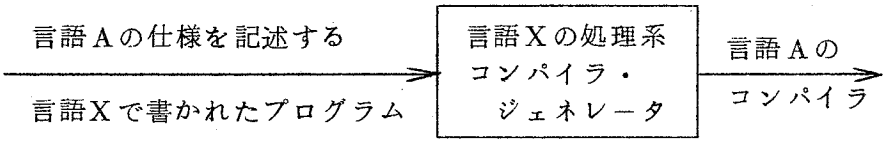


図２．２　コンパイラ・ジェネレータ

２．３　ダイナミックコンパイラシステム（ＤＣＳ）[17]～[19]

２．３．１　ＤＣＳの特徴

ＤＣＳは使用者がオンライン・リアルタイムで端末から各自のコンパイラの使用と作成が同時に行なえる自己拡張型コンパイラシステムである。ＤＣＳは前節の自己拡張型コンパイラとしては b）の型であり，ＭＣとしてＣＩ（コンパイラ・インタプリタ）を用い，表の変更，作成を行なう。拡張されるコンパイラＣは表によって制御される表駆動型（Table driven）のコンパイラである。それらの表はメタ表現と対象表現の二重の性格を持つ。またＤＣＳは拡張型言語としては前節の表２．１のⅠ－２，Ⅱ－１，Ⅳ－１，Ⅳ－２，Ⅴ－２に属する。ＤＣＳでは現在のところ演算子の拡張は，構文的な文命令の拡張により比較的容易に行なえるので，演算子の定義は含めていない。またＤＣＳではコンパイラそのものを拡張するが，従来のアセンブラ言語を用いたマクロ定義ではプリプロセッサを用いてオープンなサブルーチンを埋め込む方法をとっており，本方式とは本質的に異なる。言語面から見ると，

ＤＣＳは書式自由な文命令が定義できる言語と考えられる．また従来の＜文＞を引数にもつ手続きが，宣言できる言語に比し，定義する言語が高水準言語で一般使用者向きになっているところに特徴がある．

２．３．２　ＤＣＳの構成

ＤＣＳの設計の基本方針は次の通りである．

a）オンライン・リアルタイムで使用者が適宜コンパイラを変更拡張できる．

b）コンパイラを変更拡張するインタプリタ（コンパイラ・インタプリタ）と，それの対象となるコンパイラの二段に分ける．

c）拡張されるコンパイラは拡張変更が容易な様に可能な限り表形式にする．

具体的には図２に示す様なシステムである．ＤＣＳは以下の様な各ブロックよりなる．

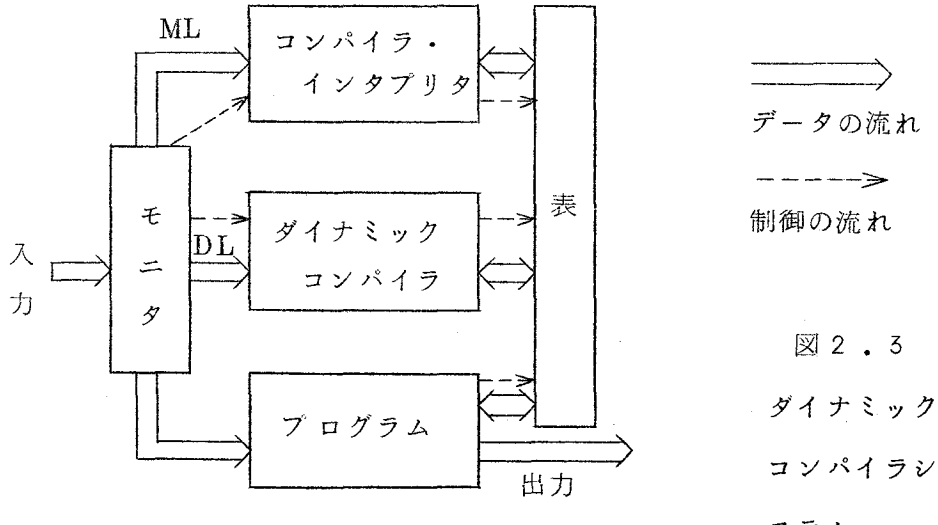


図２．３
ダイナミック
コンパイラシ
ステム

i) 固定されたコンパイラ・インタプリタ（ＣＩ）．このＣＩの入力言語をメタ言語（ＭＬ）と呼ぶ．

ii) ＣＩにより拡張変更されるダイナミックコンパイラ（ＤＣ）．このＤＣの入力言語をダイナミック言語（ＤＬ）と呼ぶ．

iii) プログラム(P)．

iv) ＤＣＳ全体の制御を行うモニタ．

v) ＣＩ，ＤＣ，Ｐの動作を規定する表．

ＭＬ，ＤＬをＤＣＳ言語系と呼び，それ等に対する処理プログラムをＤＣＳ処理系と呼ぶ．

２．４　結　　言

本章において拡張型言語，自己拡張型コンパイラ，コンパイラ記述言語の考察を行ない，筆者が提案したダイナミックコンパイラシステムの構成について述べた．

第３章　ダイナミックコンパイラシステム（ＤＣＳ）の言語系[17),(26),(27)]

３．１　緒　　言

本章においてＣＩ（コンパイラ・インタプリタ）の入力言語であるＭＬ（メタ言語）とＤＣ（ダイナミックコンパイラ）の入力言語であるＤＬ（ダイナミック言語）について述べ，最後にＤＣＳプログラムについて述べる．

ＭＬはＤＬの変更，拡張を行なうので一般には定義命令，消去命令，消去質問命令，文法打出し命令等が必要であるが，本論文では拡張型言語システムにおいて本質をなす定義命令に限定する．またコンパイル命令，実行命令，休止命令等もＤＣＳ以外のモニタまたはＯＳで行なうものとしてここでは取扱わない．ＤＬにはＡＬＧＯＬ風言語を用いた．使用者はＤＬで問題を記述する．ＭＬ，ＤＬの構文の記述には表３．１に示すＢＮＦを用いる．それらのメタ記号の意味は従来のＢＮＦと同様である．意味の記述には日本語を用いる．最後にＤＣＳプログラムについて述べる．

表３．１　ＭＬ，ＤＬの文法記述のためのＢＮＦ

	第１種ＢＮＦ	第２種ＢＮＦ
メタ記号	::＝，｜，＜，＞	＝::，⊥，≦，≧
超言語変数	＜・・・＞	≦・・・≧
記述する対象	ＤＬ	ＭＬ
始記号	＜ＤＣプログラム＞	≦ＣＩプログラム≧

３．２　メタ言語（ＭＬ）

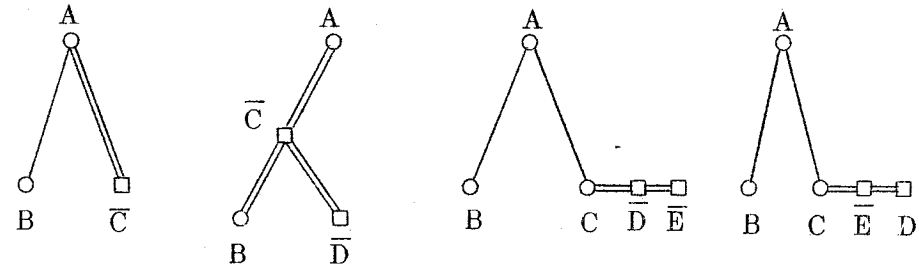
前節で述べたように定義命令だけを扱かう。

３．２．１　定義命令

定義命令でＤＬに新しく命令を付加する。

1）構文の拡張

構文の拡張は図３．１に示すように４種類考えられる。しかし，実際新しく命令を定義する場合，既存の命令の変更を行なうよりも，従来の命令と，変更を行なった命令とを同時に使用する場合が多い。そこで定義命令として，　a）並列付加型を考える。



a）並列付加型　b）並列挿入型　c）直列付加型　d）直列挿入型

$A ::= B$ ↓ $A ::= B \mid \overline{C}$

$A ::= B$ ↓ $A ::= \overline{C}$, $\overline{C} ::= B \mid \overline{D}$

$A ::= B \mid C$ ↓ $A ::= B \mid C\overline{D}\overline{E}$

$A ::= B \mid CD$ ↓ $A ::= B \mid C\overline{E}D$

$\overline{\ }$の付いた語は新しく付加される非終端語または終端語

図３．１　シンタクティカルチャートとＢＮＦによる文法の拡張

2）定義形式

命令の定義は既存の計算機言語でも実現されている．例えばＡＬＧＯＬ６０での手続き，ＦＯＲＴＲＡＮでの関数，サブルーチン，文関数があり，他にＡＬＧＯＬ６８等における演算子の定義がある．ＤＣＳで構文的な定義をさらに一般化して次の６種類の定義形式を考える．

Ⅰ　単項演算子（UNARY　OPERATOR）

Ⅱ　二項演算子（BINARY　OPERATOR）

Ⅲ　関　　数（FUNCTION）

Ⅳ　手　続　き（SUBROUTINE）

Ⅴ　文　命　令（STATEMENT）

Ⅵ　複合命令（COMPOUND　STATEMENTS）

文命令とはＦＯＲ文，ＩＦ文のように文としてまとまった意味をもつもの，複合命令とは文の複合体としての命令，手順をあらわすものである．現在，ＤＣＳではⅤの文命令を扱う．

3）定義命令における語の分類

定義命令で使用する語，記号の分類を表３．２に示す．

①：ＭＬの構文を記述するためのメタ記号，超言語変数（非終端語）であって直接定義命令の中に出現しない．

②～⑧：ＭＬの終端語であって言語の中に出現する．②，③は決まった意味をもち不変であるが，④～⑥はＭＬからみて変数である．④，⑤は一度ある意味が登録されると，それが取消されるまで有効であるが，⑥はそれが使用されている定義命令の中だけで有効である．④，⑤はALGOL６０の場合と違って，ＤＣＳでは使用者によって定義される．（但し核命令を構成している語は意味が固定されてい

表３．２　ＭＬの語・記号の分類

種類	表現する対象	使用される記号
①	ＭＬを記述するメタ記号 超言語変数	::= ，⊥，≦，≧ ≦ＭＬプログラム≧≦定義命令≧等
②	第２種ＢＮＦの終端語 （予約語）	DEFINE，DEFEND，ADD， STATEMENT，SYNTAX， ASSIGNMENT，SEMANTICS
③	ＭＬの終端記号 区切り記号 （予約語）	¥，＜，＞，::=
④	ＭＬの終端語 変数，ＤＣの非終端語 （予約語）	＜DL program＞，＜statement＞等
⑤	ＭＬの終端語 変数，ＤＣの終端語 （予約語）	begin，end 等ＤＬの基本記号
⑥	ＭＬの終端語 変数，局所的変数，識別子	Ｖ，Ｎ等の変数，識別子
⑦	数	5，10.0 等の数
⑧	特殊記号 演算子等になりうる記号	¥，@ 等

る.) MLからみると④,⑤は単なる変数であるが，DLからではこれらの変数がDLの新しい区切り記号等となり，DLの拡張となる.

3.2.2 構　　文

MLはDLを記述するので，DLの書換え規則と混乱を起さないように，BNFにおける::=，<・・・>，|の各々に対応する=::，≤・・・≥，⊥を用いる．これらのメタ記号，メタ変数の意味はBNFと全く同じである．以下にMLの構文を示す.

≤MLプログラム≥=:: ≤定義命令≥

≤定義命令≥=:: DEFINE ≤定義命令本体≥ DEFEND

≤定義命令本体≥=:: ≤宣言部∨≥≤本体∨≥

≤宣言部∨≥=:: STATEMENT ≤文命令名≥¥

≤文命令名≥=:: *me*

≤本体∨≥=:: ≤構文部≥≤割当部≥≤意味部≥

≤構文部≥=:: SYNTAX ADD ¥ ≤構文接続部≥≤構文定義部≥

≤構文接続部≥:: *NT* :: =*UNT* ¥

≤構文定義部≥=:: *UNT* :: =*UT* *NT* 〔{*UY* ⊥ *T* }*NT*〕$^{※}$ *UT* ¥

≤割当部≥=:: ASSIGNMENT部≤割当本体≥

≤割当本体≥=:: 〔*NT* :: =*mv* ¥〕$^{+}$

≤意味部≥ SEMANTICS ≤意味本体≥

≤意味本体≥=:: <DLプログラム>¥

但し，{ A⊥B }：A，Bのいずれかが存在

〔　　〕$^{+}$：1回以上存在

〔　〕　：0または1回存在

〔　〕※　：0回以上存在

MLを記述している書換え規則の終端語にDLの非端語，終端語が含まれるが，これらを各々NT，Tで表わし，新しく定義されてそれらになる語をUNT，UTとした．これらは表3．2の④,⑤の各々に対応する．mv，meはMLでの変数および識別子であり，⑥に対応する．＜DLプログラム＞は3．3．2節で定義する

3．2．3　意　　味

ML（定義命令）は図3．2に示す構造になっている．

○　DEFINEの後の≤宣言部≥でSTATEMENTと宣言し，次に文命令名を与える．

○　構文部（SYNTAX部）の≤構文接続部≥はすでに存在するDLの書換規則のどの部分に接続されるかを示す．

○　≤構文定義部≥は拡張する部分がどのような構文を持つか定義する．

○　割当部（ASSIGNMENT部）では，構文部で導入されたDLの非終端語に終端語（局所的な変数）を割当てる．

DEFINE　≤宣言部≥
　SYNTAX
　　≤構文接続部≥
　　≤構文定義部≥
　ASSIGNMENT
　　≤割当本体≥
　SEMANTICS
　　≤意味本体≥
DEFEND

図3．2　定義命令の構造

○　意味部（SEMANTICS部）では，割当部で割当られた変数を用いてDL上で，定義される命令の意味を記述する．

3．3　ダイナミック言語（DL）

使用者はこのDLを用いて問題を記述する．

3．3．1　核　言　語

拡張される前のもとの基本となるDLを核言語（DL_B）と呼ぶ．ここではDL_Bに次の4種の命令を持たせる．

1）入出力文（READ，WRITE 文）

2）代入文

3）無条件飛び越し文（GO　TO＜ラベル＞）

4）条件付飛び越し文（IF　POSI＜変数＞THEN＜ラベル＞ELSE＜ラベル＞）．（＜変数＞の値が0でもPOSIの＜ラベル＞を実行する．）

命令としては上記の1）～4）の命令でALGOL60の命令まで拡張できる．

3．3．2　構　　文

DL_Bの構文を次に示す．

＜DLプログラム＞ ::＝＜ブロック＞｜＜複合文＞

＜ブロック＞ ::＝〔＜名札＞：〕※ begin ＜宣言＞
〔；＜文＞〕※ end

＜複合文＞ ::＝〔＜名札＞：〕※ begin ＜文＞〔；＜文＞〕※ end

＜宣言＞ ::＝dec ＜変数＞〔，＜変数＞〕※

＜文＞::＝＜無条件文＞｜＜条件文＞｜＜入出力文＞

＜無条件文＞::＝＜複合文＞｜＜基本文＞

＜基本文＞::＝＜代入文＞｜＜飛越文＞｜＜空文＞

＜条件文＞::＝〔＜名札＞:〕※ ifposi ＜変数＞

then ＜名札＞ else ＜名札＞

＜入出力文＞::＝＜入力文＞｜＜出力文＞

＜入力文＞::＝〔＜名札＞:〕※ read ＜変数＞〔,＜変数＞〕※

＜出力文＞::＝〔＜名札＞:〕※ write ＜変数＞〔,＜変数＞〕※

＜飛越文＞::＝〔＜名札＞:〕※ goto ＜名札＞

＜空文＞::＝〔＜名札＞:〕※

＜代入文＞::＝〔＜名札＞:〕※＜変数＞::＝＜算術式＞

＜算術式＞::＝〔＜加減作用素＞〕＜項＞〔＜加減作用素＞＜項＞〕※

＜項＞::＝＜因子＞〔＜乗除作用素＞＜因子＞〕※

＜因子＞::＝＜一次子＞〔↑＜一次子＞〕※

＜一次子＞::＝＜数＞｜＜変数＞｜＜カッコ付算術式＞

＜カッコ付算術式＞::＝(＜算術式＞)

＜数＞::＝＜十進数＞〔＜指数部＞〕

＜十進数＞::＝＜小数＞｜＜十進＞

＜十進＞::＝＜整数＞〔＜小数＞〕

＜整数＞::＝〔＜数字＞〕+

＜小数＞::＝・＜整数＞

＜名札＞::＝＜名前＞

＜変数＞::＝＜名前＞

＜名前＞::＝＜英字＞〔＜英字又は数字＞〕※

＜英字又は数字＞::＝＜英字＞｜＜数字＞

＜英字＞::＝A｜B｜C｜・・・｜X｜Y｜Z

＜数字＞::＝0｜1｜2｜・・・｜7｜8｜9

ＤＬの構文は直列型（＜A＞::＝＜B＞＜C＞＜D＞）と並列型（＜A＞::＝＜B＞｜＜C＞｜＜D＞）の書換規則で記述されている．この事に関しては第５章の処理系で詳しく述べる．

３．３．３　意　　味

核言語DL_BはALGOL風の言語である．その意味はALGOLに準ずる．

３．４　ＤＣＳプログラム

ＤＣＳではＤＬを拡張するＭＬは使用者のプログラムで実行される．使用者は端末から図３．３に示すようなＤＣＳプログラムを入力する．

→	P	定義命令	P′	コンパイル命令	リンク命令	実行命令	→

図３．３　プログラム列の一例

この例では，ＰとP′でＤＬで書かれた一つの問題を解くプログラムであり，Ｐはその時点において使用可能な命令からなるプログラムであり，P′は定義命令において定義された命令を含むプログラムである．Ｐ，P′はＤＣによって解釈，ほん訳される．定義命令はＣＩによって処理され，ＤＣのコンパイル機能を拡張する．コンパイル命令，リンク命令，実行命令はＤＣＳ以外のコマンド命令を通じて使用者が入力する．

図３．４にa）ＩＦ文，b）ＦＯＲ文の定義例を示す．次にその特徴を述べ

```
DEFINE STATEMENT ZIFST ¥
  SYNTAX ADD ¥
    ZST # ZIFST ¥
    ZIFST # IF ZAE POSI ZST ZERO ZST NEGA ZST FI ¥
  ASSIGNMENT
    ZAE # N ¥ ZST # X1 ¥
    ZST # X2 ¥ ZST # X3 ¥
  SEMANTICS
    BEGIN DEC A,B % A # N %
     IFPOSI A THEN L1 ELSE L2 %
    L1 @ B # -A % IFPOSI B THEN L3 ELSE L4 %
    L4 @ X1 % GOTO L5 %
    L3 @ X2 % GOTO L5 %
    L2 @ X3 % L5 @ END ¥
DEFEND
```

a) ＩＦ文の定義例

```
DEFINE STATEMENT ZFST ¥
  SYNTAX ADD ¥
    ZST # ZFST ¥
    ZFST # FOR ZSV # ZAE STEP ZAE UNTIL ZAE DO ZST ROF ¥
  ASSIGNMENT
    ZSV # I ¥ ZAE # N1 ¥
    ZAE # N2 ¥ ZAE # N3 ¥
    ZST # X ¥
  SEMANTICS
    BEGIN DEC A % I # N1 %
    LA @ A # N3-I %
    IFPOSI A THEN L1 ELSE L2 %
    L1 @ X % I # I+N2 %
    GOTO LA % L2 @ END ¥
DEFEND
```

b) ＦＯＲ文の定義例　　図３．４　定義例

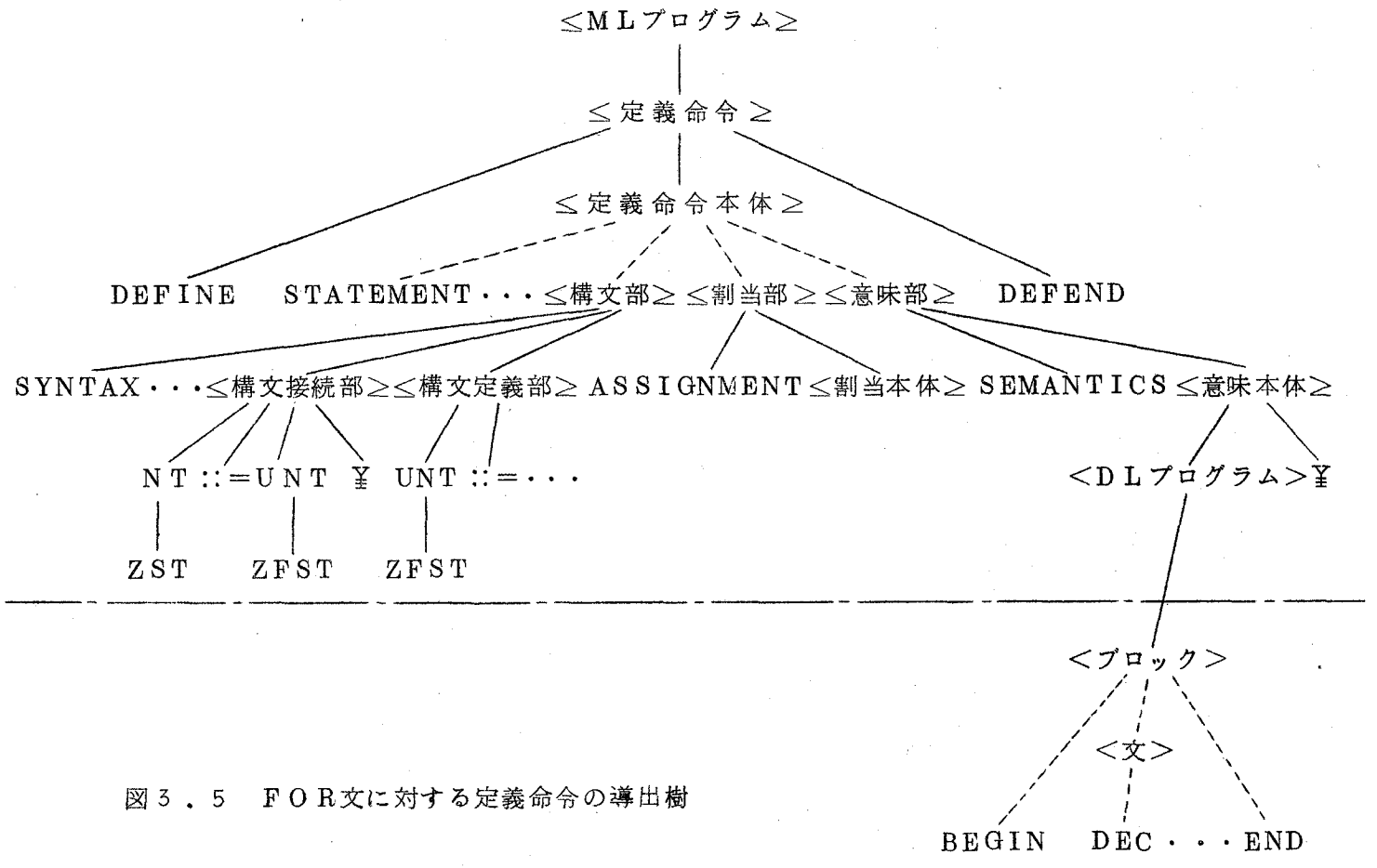


図３．５　ＦＯＲ文に対する定義命令の導出樹

る．

○　Ｚで始まる名前はＭＬに対する変数でＤＬに対して非終端語を表わす．¥はＭＬでの区切り記号，％はＤＬでの区切り記号を表わす．

○　割当部において割当られた変数は意味部において標準手続きとして使用できる．

○　ＤＬの非終端語はＭＬからみると終端語となり，二重の性格（メタ表現対象表現）をもつ．すなわちＣＩが動作しているときには，ＤＣの構文解析における構文表の要素となり，重要な役割を果たす．

○　図３．５にｂ）のＦＯＲ文に対する定義命令の導出樹を示す．構文が二段になっているのが特徴である．

３．５　結　　言

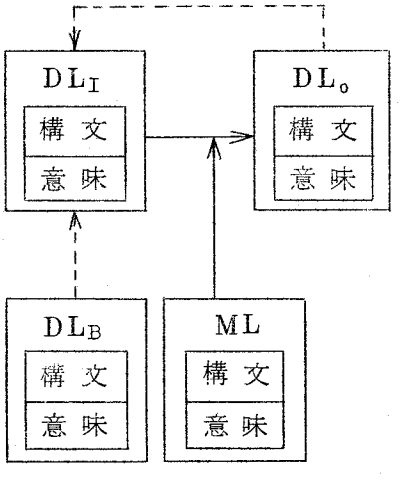
本章においてＤＣＳ言語系について述べた．ＭＬについては文命令の定義命令で，並列付加型の場合を考察した．複合命令では他の型も必要であるので今後調べる必要があろう．DL_BとしてここではＡＬＧＯＬのサブセットと類似の言語を選定したが，より広範な命令が定義できるためには，ビットレベル，文字レベル，語レベルの３レベルのDL_Bの選定が必要である．

第４章　ＤＣＳ言語系の無曖昧さ

４．１　緒　　言

本章ではＤＣＳ言語系において拡張される構文の無曖昧さについて論じる．すなわち，拡張される構文の無曖昧さが保存されるための構文の制限条件（付加される書換規則の制約条件）を求める．次に無曖昧さが保存されるように，算術式を構成する方法について述べる．従来，曖昧さの問題については，曖昧な言語の存在性，任意の文法における曖昧さの存在判定などの基本的問題が数多く発表されてきている．(39)～(43) これらはあるクラスに属する与えられた言語，文法における曖昧さおよび無曖昧さについて論じたものである．

ＤＣＳはＭＬでもってＤＬに新しく書換規則を付加するが，その様子を図４．１に示す．最初核言語DL_Bから出発する．そしてMLによってDL_Iを拡張してDL_0にする．ここで拡張とはDL_Iに書換規則を付加することによって



DL_B：ＭＬによって拡張されるもとの核の言語ＤＬ
DL_I：ＭＬによって拡張される前の言語
DL_0：ＭＬによって拡張された後の言語

図４．１　ＭＬによるＤＬの拡張

DL_0にすることである．そして本章ではその書換規則の付加によって無曖昧さが保存されるか否かを論じる．

4．2　基本概念

ここでは本章で取り扱う諸記号の定義を行なう．なお言語，文法とはすべて文脈自由言語（cfl），文脈自由文法（cfg）である．

〔定義4．1〕　Gとは次の四重組である．

$$G=(V_N, V_T, P, S)$$

ここで V_N：非終端語の有限集合，V_T：終端語の有限集合，$V(=V_N \cup V_T, V_N \cap V_T=\phi)$：全語いの集合，$P$：書換規則の集合で書換規則とは$A\rightarrow\alpha$で，$A\in V_N$，$\alpha\in V^*$，$S$：始記号である．

もし，$A\rightarrow\alpha\in P$，w_1，w_2　V^*ならば$w_1 A w_2\rightarrow w_1\alpha w_2$とかき，$\overset{*}{\rightarrow}$は$\rightarrow$の1回以上の繰返しとし，導出と呼ぶ．

〔定義4．2〕　言語Lとは

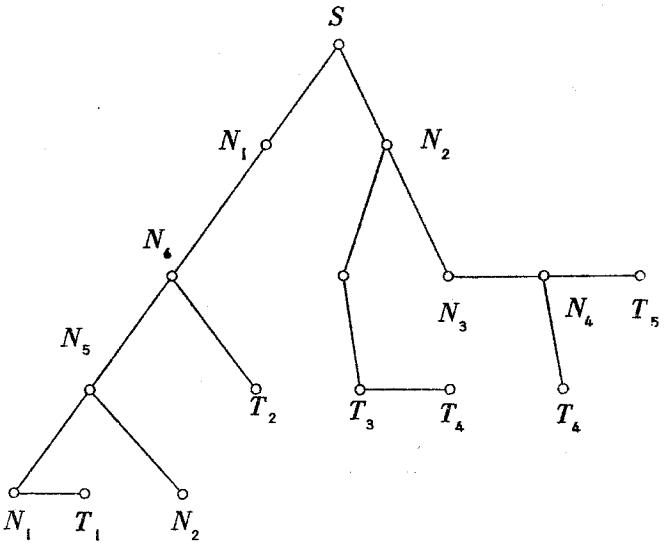
$$L(G)=\{w \mid S\overset{*}{\rightarrow} w,\ w\in V_T^*\}$$　である．

〔定義4．3〕　$G=(V_N, V_T, P, S)$は$L(G)$の中に二つ以上の相異なった最左導出を有する文が存在するとき，曖昧であるといい，そうでないとき，すなわち，$L(G)$の中のすべての文に対して最左導出が一意的に定まるとき，無曖昧であるという．

シンタルティカルチャート（ＳＣ）とはGの書換え規則を図式化したものであるが，Gが与えられればＳＣを作成することができる．例えば式（4．1）のようなGに対して図4．2のようなＳＣが作れる．

$$G=(V_N, V_T, P, S)$$

$$V_N=\{S, N_1, N_2, N_3, N_4, N_5, N_6\}$$



非終端語で終っているところは関数呼びだしになっている

図４．２　式（４．１）のシンタクティカルチャート（ＳＣ）

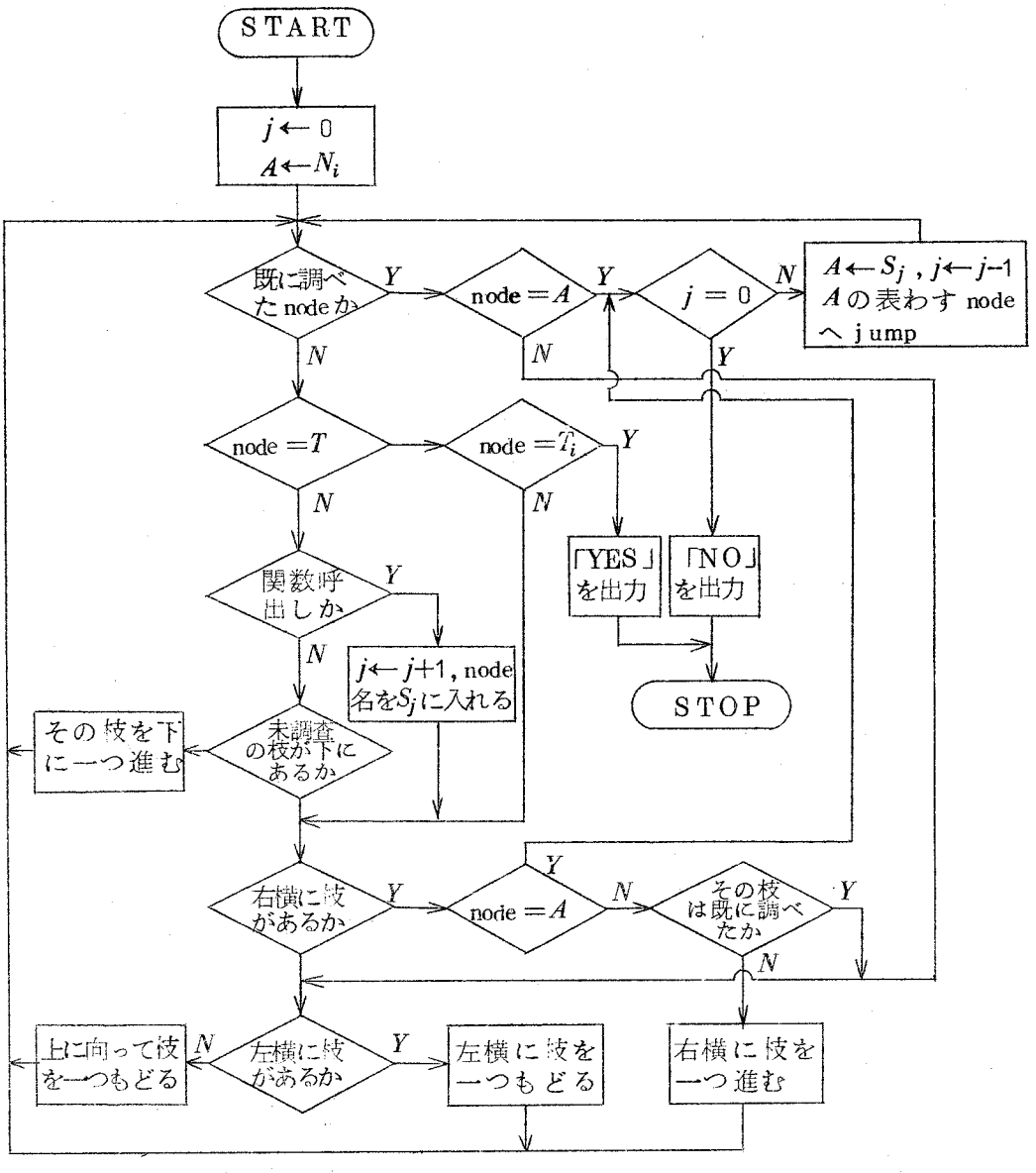
$$V_T=\{T_1, T_2, T_3, T_4, T_5\}$$

$$P=\{S\to N_1,\ S\to N_2,\ N_1\to N_6,\ N_2\to N_3,\ N_2\to N_3N_4T_5,\ N_3\to T_3T_4,$$
$$N_4\to T_4,\ N_6\to N_5,\ N_6\to T_2,\ N_5\to N_1T_1,\ N_5\to N_2\} \qquad (4.1)$$

次にGが与えられ，その中の$N_i\ (\in V_N)$に対して，N_iから導出されるstringの集合を，すなわち$\{w \mid N_i \overset{*}{\to} w,\ w\in V_T\}=L(GN_i)$とし，$w\ (\in L(GN_i))$を構成している記号の集合を$\sigma(N_i)$とする．これは$N_i$だけによって定まる集合である．

〔補題４．１〕　任意のGに対してN_i，$T_i\ (N_i\in V_N, T_i\in V_T)$が与えられると，$T_i\in\sigma(N_i)$であるか否か判定するアルゴリズムが存在する．

〔証　明〕　Gが与えられるとSCが求まり，そのSCに対して，N_iから出発して下にたどっていきすべての枝の分枝についてT_iがあるかどうか調べ



註：既に調べた節点とは

その節点から出る枝はすべて既に調べた枝であるもの

既に調べた枝とは

その枝を行きと帰りと２回たどったもの

図４．３のSC上の各節点は名前が同じものでもその位置によって区別がなされているものとする．

図４．３　$T_i \in \sigma(N_i)$を判定するアルゴリズム

る．そのとき関数呼び出しがあれば，それを記憶しておき，後に改めて呼びだされた非終端語から下へたどって調べる．そのときすでに調べた枝は調べない．このようなチエックは有限回の操作で終る．図４．３にその具体的な流れ図を示す．

４．３　文命令の拡張

ＤＣＳにおいてＭＬは核言語DL_Bから出発し，ＤＬを拡張するが，DL_Bとしてアルゴルのサブセットを用い，それに文命令（代入文，条件文等）レベルでの核拡張を行なう際，無曖昧さが保存されるための一つの十分条件を与える．ここで拡張とは，もとの無曖昧文法G_Iに$A \rightarrow \alpha$なる書換え規則δを付加することを意味する．

〔定理４．２〕　$G_I=(V_N, V_T, P, S)$が無曖昧なｃｆｇで，$V_T' \cap V=\phi$なるV_T'があって，付加される書換え規則δが

$$A \rightarrow a_b N_1 T_1 N_2 T_2 \cdots\cdots N_n T_n N_{n+1} a_e \qquad (4.2)$$

であるとき

$$G_0=(V_N, V_T \cup \{a_b, a_e\}, P \cup \{\delta\}, S)$$

は無曖昧なｃｆｇである．ただし

$$A \in V_N,\ a_b, a_e, \in V_T',\ N_i \in V_N,\ T_i \in V_T \cup V_T',$$

$$T_i \notin \sigma(N_i) \quad (n=1, 2, \cdots\cdots, n)$$

である．

〔証　明〕　G_0が曖昧であると仮定すると，$L(G_0)$の中に二つ以上の最左導出からなる文が存在する．その文をwとし今二つの異なった導出を$\xrightarrow[1]{*}$，$\xrightarrow[2]{*}$で表わす．

Ⅰ）wの中にa_b，a_eが含まれないとき

$S \xrightarrow[G_0]{*} w$において書換規則が使用されていないので，これは$S \xrightarrow[G_5]{*} w$と導

出が一致している．このことはG_Iが無曖昧文法であるから，wの導出が二様に存在することと矛盾する．

Ⅱ） wの中にa_b，a_eが含まれているとき，始めてδが適用されるとすると

$$S \xrightarrow[G_0]{*} \varphi A \psi \rightarrow \varphi\, a_b N_1 T_1 \cdots N_n T_n N_{n+1}\, a_e \psi \xrightarrow[G_0]{*} w \quad , \quad \varphi \in V_T^*$$

a） $a_b N_1 T_1 \cdots N_n T_n N_{n+1}\, a_e$から曖昧さのある導出が行なわれないとき，すなわち$\psi$から二様の導出があるとき

$$S \xrightarrow{*} \varphi\, a_b N_1 T_1 \cdots N_n T_n N_{n+1}\, a_e \psi \xrightarrow[1]{*} w$$

$$S \xrightarrow{*} \varphi\, a_b N_1 T_1 \cdots N_n T_n N_{n+1}\, a_e \psi \xrightarrow[2]{*} w \qquad \text{が存在する．}$$

いまwを$\xi a_e \eta$とすると，上の二式より，それぞれ次の二式が導かれる．すなわちηの二つの導出が存在する．

$$\psi \xrightarrow[1]{*} \eta$$
$$\psi \xrightarrow[2]{*} \eta \qquad (\eta \in V_T^*)$$

そして，ηの中にa_b，a_eが存在しなければⅠ）に帰着し，もし存在すれば，もう一度くりかえし，$\psi' \xrightarrow[1]{*} \eta'$となる．そして最終的に$\psi^{(n)} \xrightarrow[1]{*} \eta^{(n)}$，$\psi^{(n)} \xrightarrow[2]{*} \eta^{(n)}$となり，$\eta^{(n)}$の中に$a_b$，$a_e$が含まれていない形になる．この導出は$G_I$の書換規則の$\eta^{(n)}$で行なわれている．これは$G_I$が無曖昧であることに矛盾する．

b） $a_b N_1 T_1 \cdots N_n T_n N_{n+1}\, a_e$の中から曖昧さが生じる場合

$$S \xrightarrow{*} \varphi\, a_b N_1 T_1 \cdots N_n T_n N_{n+1} \psi \xrightarrow[1]{*} w$$

$$S \xrightarrow{*} \varphi\, a_b N_1 T_1 \cdots N_n T_n N_{n+1} \psi \xrightarrow[2]{*} w$$

いまT_1に着目してwを表現すると，$w=\varphi a_b \eta T_1 \pi a_e r$とおき

$$S \xrightarrow[1]{*} \varphi a_b \eta T_1 \pi a_e r$$

$$S \xrightarrow[2]{*} \varphi a_b \eta T_1 \pi a_e r$$

となり，N_1からT_1が導出されないことにより

$$N_1 \xrightarrow[1]{*} \eta$$

$$N_1 \xrightarrow[2]{*} \eta$$

の導出より曖昧さが生じている．ηの中にa_b，a_eが含まれていないときⅠ）に帰着し，a_b，a_eが含まれているときはⅡ）をくりかえし，最終的にはηの中にa_b，a_eが含まれていない状態にすることができる．しかしこれはG_Iの仮定と矛盾する．そしてこれは任意のN_i，T_i（$i=1, 2, \cdots, n$）に対しても成り立つ．〔証明終〕

前述の証明は厳密には超限帰納法を用いて行なわれなければならないが，ここでは簡略して行なった．しかし証明の本筋はこれとほとんど変わらない．このことは後述の定理４．６，４．７の証明においても同様である．

定理４．２において，a_b，a_eは書換規則が付加されるごとに新しく固有な終端記号が導入されることを意味しているが，これはかなりきびしい条件である．a_eは別に固有のものでなくてもよい．すなわち次の系が成り立つ．

〔系４．３〕　定理３．１において付加される書換規則δの式（４．２）は次の式（４．３）であってもよい。

$$A \rightarrow a_b N_1 T_1 N_2 T_2 \cdots N_n T_n \qquad (4.3)$$

a_eを取り除けば他は定理４．２と同じである．

証明は定理４．２とまったく同様に行なえる．

また定理４．２において$T_i \leqq \sigma(N_i)$としたが，これは$T_i \in \sigma(N_{i+1})$（i

＝1，2，・・・，n）としてもよい．すなわち次の系が成り立つ．この証明も定理４．２と同様である．

〔系４．４〕　定理４．２において

$$T_i \in \sigma(N_i) \text{のかわりに} T_i \in \sigma(N_{i+1}) \quad (i=1,2,\cdots,n)$$

としても成り立つ．

以上の定理の本質は$N_1T_1N_2T_2\cdots N_nT_n$のように非終端語が交互になっているのが重要な役割を果たしている．無曖昧さが保存されないような場合の定理を次に述べる．

〔定理４．５〕　$G_I=(V_N, V_T, P, S)$が無曖昧なｃｆｇで付加される書換規則δが

$$A \to a\alpha$$

ただし，$A \in V_N$，$a \in V_T'$，$V_T' \cap V = \phi$

ならば，　$G_0=(V_N, V_N \cup \{a\}, P \cup \{\delta\}, S)$

は無曖昧なｃｆｇとなるとは限らない．

〔証　明〕　ある無曖昧文法G_Iに対してこのようなδを付加したとき曖昧さのある文法G_0を作るとよい．例えば$G_I=(V_N, V_T, P, S)$として

$$\left.\begin{aligned} &V_N=\{S, N_1, N_2\}, \quad V_T=\{a, b, d\} \\ &P=\{S \to N_1 d N_2,\ N_1 \to a,\ N_1 \to aa,\ N_2 \to ab,\ N_2 \to b\} \end{aligned}\right\} \quad (4.4)$$

とすれば，$L(G_I)=\{adab,\ adb,\ aadab,\ aadb\}$となり図４．４のような導出樹が存在し，他の導出は存在しないから無曖昧である．

明　付加するδとして，$S \to CN_1N_2$としたとき，$caab$は図４．５のように二様に導出が存在し，曖昧さをもつことになる．〔証明終〕

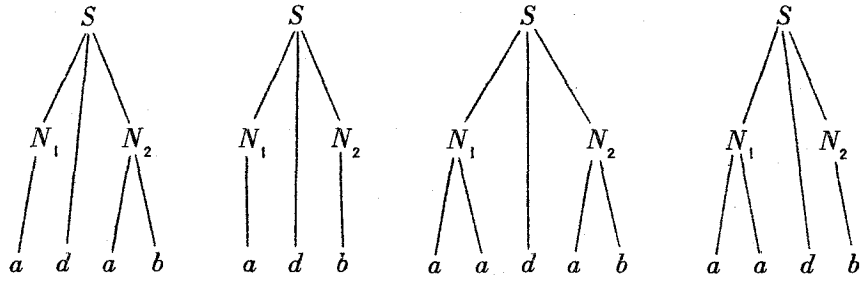


図４．４　式（４．４）の$L(G)$に対する導出樹

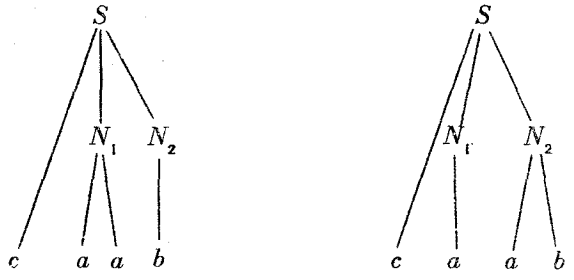


図４．５　$caab$に対する二つの導出樹

〔例　題〕　核言語DL_BとしてＡＬＧＯＬのサブセットを用い，DL_Iに繰り返し文に含まれていない場合，for文を付加するさいの書換規則は式（４．５）のようになる．

$$\left\{\begin{array}{l}\langle\text{statement}\rangle \rightarrow \text{for}\ \langle\text{simple variable}\rangle := \langle\text{arithmetic} \\ \qquad\qquad \text{expression}\rangle\ \text{step}\ \langle\text{arithmetic expres-} \\ \qquad\qquad \text{sion}\rangle\ \text{until}\ \langle\text{arithmetic expression}\rangle \\ \qquad\qquad \text{do}\ \langle\text{statement}\rangle\ \text{rof}\end{array}\right. \tag{4.5}$$

となる．ただし，各々の＜・・・＞はG_I（ここではDL_Bの文法）に属す非終端語である．ここでfor文の最後にrofがあるのは定理４．２を使用した為である．このrofを取り除くには系４．３を用いればよいが，しかしこのと

きも，既存の終端語で終っていなければならない．ＡＬＧＯＬではfor文の次に；で終らすようにすればよいが，実際そのような書換規則がみつかるかどうか現在わかっていない．

４．４　算術式の構成

本節では最初から無曖昧さを保存するよに書換規則を付加していく方法について述べる．現段階では算術式だけを考慮した．

算術式の構文はＡＬＧＯＬ６０では一般に以下のようになっている．

＜加減作用素＞→＋｜－

＜乗除作用素＞→×｜／

＜１次子＞　→＜符号のない数＞｜＜変数＞｜＜関数呼び出し＞｜
（＜算術式＞）

＜因子＞　→＜１次子＞｜＜因子＞｜＜１次子＞

＜項＞　→＜因子＞｜＜項＞＜乗除作用素＞＜因子＞

＜算術式＞　→＜項＞｜＜加減作用素＞＜項＞｜＜算術式＞
＜加減作用素＞＜項＞

(4.6)

しかし式（４．６）の書換規則には，見やすさ，記述の簡潔さはあるが，拡張する際，曖昧さの問題を考慮していない．そこでこれを新しい演算子が導入されやすいように，しかも無曖昧さが保障されるように，構成論的に組立てる．準備として，記述を簡単にするため，表４．１のような記号を導入する．

まず次式を出発点とする．

$$E \rightarrow n \mid v \mid f \mid (E) \qquad (4.7)$$

拡張する手順は次のⅰ），ⅱ）になる．

ⅰ）Eを新しい非終端記号におきかえる．ただし(E)はそのままにする．そして，たとえば新しい非終端記号をPとすれば，$E\rightarrow P$を作る．

ⅱa）付加する演算子が二項演算子の場合（例えば↑）$E\rightarrow E\uparrow P$（Pは適当な非終端語）でもって導入する．

ⅱb）付加する演算子が単項演算子の場合（例えば＋）$E\rightarrow +T$（Tは適当な非終端語）でもって導入する．

ⅱc）付加する演算子が単項演算子と二項演算子を兼ねる場合（例えば－）$E\rightarrow -T$，$E\rightarrow E-T$（二つの書換規則において同じ文字は同一物を表わす）でもって導入する．

表４．１　語い表

	BNFにおける記号	新しい記号	
非終端語	＜算術式＞	E	非終端語
	＜項＞	T	
	＜因子＞	F	
	＜１次子＞	P	
	＜符号のない数＞	n	終端語
	＜変数＞	v	
	＜関数呼び出し＞	f	
終端語	＋	＋	
	－	－	
	×	×	
	／	／	
	(	(	
	)	)	
	↑	↑	

式（４．７）にⅰ），ⅱ）を適用すると次の式（４．８）～（４．１０）が得られる．

$$P\rightarrow n \mid v \mid f \mid (E)$$
$$E\rightarrow P \mid E\uparrow P \qquad (4.8)$$

$$P \rightarrow n \mid v \mid f \mid (E)$$

$$F \rightarrow P \mid F \uparrow P$$

$$E \rightarrow F \mid E \times P \mid E / F \qquad (4.9)$$

$$P \rightarrow n \mid v \mid f \mid (E)$$

$$F \rightarrow P \mid F \uparrow P$$

$$T \rightarrow F \mid T \times F \mid T / F$$

$$E \rightarrow T \mid +T \mid -T \mid E+T \mid E-T \qquad (4.10)$$

ここで式（４．１０）にいたって式（４．６）の書換規則が得られる．

式（４．７）においては，表４．１からもわかるように＜符号のない数＞，＜変数＞等はそれぞれ別個の終端記号とした．次に手順ⅰ），ⅱ）をさらに詳しく説明し無曖昧さを保存することを証明する．

ⅰ）は非終端語を一つ付加して名前のつけかえ（renaming)を行なうことである．例えば式（４．７）から式（４．８）への過程においては図４．６のようになる．

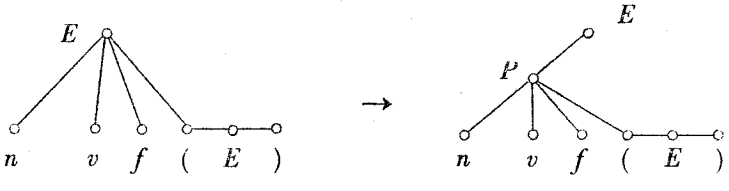


図４．６　renaming

ⅱa）　$A \rightarrow A\,T\,N$の形の書換規則を付加することになる．ただし，A，$N \in V_N$，$T \in V_T'$，$V_T' \cap V = \phi$ $(V = V_N \cup V_T)$，上の例では，$E \rightarrow E \uparrow P$となり図４．７のようになる．

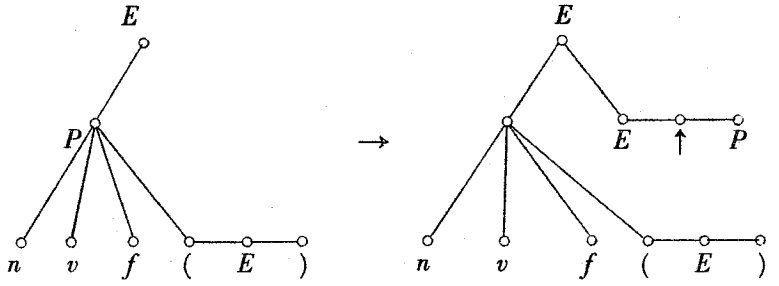


図４．７　ⅱa）型の書換規則の付加

ⅱc）　$A\rightarrow TN$，$A\rightarrow ATN$ の形の書換規則の付加，ただし，A，$N\in V_N$，$T\in V_T'$　式（４．９）から式（４．１０）への過程においては図４．８のようになる

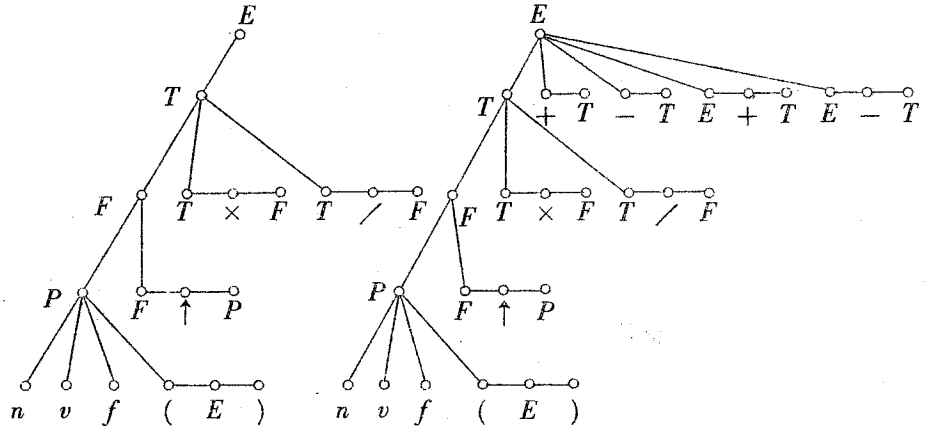


図４．８　ⅱc）の型の書換規則の付加

ⅰ）の renamingの操作はまったく新しい名前のつけかえだから曖昧さに関しては問題がない．ⅱa），ⅱb），ⅱc）に関して次の定理を証明する．

〔定理４．６〕　$G_I=(V_N, V_T, P, S)$ が無曖昧な cfgで付加される書換規則 δ が

$$A\rightarrow ATN$$

であるとき $G_0=(V_N, V_T\cup\{T\}, P\cup\{\delta\}, S$ は無曖昧なcfgとなる．ただし，$A, N\in V_N, T\in V_T', V_T'\cap V=\phi$

〔証　明〕G_0に曖昧さがあると仮定すると，$L(G_0)$の中に二つ以上の最左導出からなる文が存在する．その文をwとし，異った導出を$\xrightarrow[1]{*}$，$\xrightarrow[2]{*}$で表わす．

$S\xrightarrow[1]{*}w$，　$S\xrightarrow[2]{*}w$　において

Ⅰ）wの中にTを含まないとき

δを用いないで導出されているからG_Iに曖昧さが存在することになり矛盾する．

Ⅱ）wの中にTを含むとき

ψの中にはTを含まない形で$w=\varphi T\psi$とおくことができる．

$$S\xrightarrow{*}\varphi' A T N\psi'\xrightarrow{*}\varphi T\psi \qquad (4.11)$$

以下$\varphi' A T N\psi'$において

a）$N\psi'\xrightarrow{*}\psi$

b）$\varphi' A\xrightarrow{*}\varphi$

c）$S\xrightarrow{*}\varphi' A T N\psi'$

の各々にわけて証明を行なう．

a）$N\psi'\xrightarrow{*}\psi$　においてはψの中にTを含まないから，Ⅰ）によりこの導出において曖昧さを生じない．

b）$\varphi' A\xrightarrow{*}\varphi$　の導出において　$\varphi=\varphi'\varphi_0$　とおくことができる．（φ，$\varphi'\in V_T^*$）．すなわち，$\varphi' A\xrightarrow{*}\varphi'\varphi_0$となり，この導出において曖昧さが生ずるとすれば$\varphi'\in V_T^*$だから　$A\xrightarrow{*}\varphi_0$においてである．$\varphi_0(\in V_T^*)$は一般に$T$を含んでいるが，Ⅱb)をくりかえして，最終的にはⅠ）に帰着することができる．

c) $S \xrightarrow{*} \varphi' A T N \psi$ に曖昧さがある場合

$S \xrightarrow{*} \varphi' A \psi' \rightarrow \varphi' A T N \psi'$

となり，$S \xrightarrow{*} \varphi' A \psi'$ に曖昧さがあることになる．ここで，$\psi' \in V^*$ であるが a)，b) の場合と同様に I) に帰属できる． 〔証明終〕

〔定理 4．7〕 $G_I = (V_N, V_T, P, S)$ が無曖昧で付加される書換規則 δ_1，δ_2 が

$A \rightarrow T N$， $A \rightarrow A T N$

であるとき，$G_0 = (V_N, V_T \cup \{T\}, P \cup \{\delta_1, \delta_2\}, S)$ は無曖昧である．ただし，$A, N \in V_N, T \in V_T'$ で二つの書換規則の記号の同一名は同一物を表わす．

〔証 明〕 定理 4．6 と同様に右から最初の T に着目してわける．ψ に T を含まないように $w = \varphi T \psi$ とおく．

I) φ に T を含まないとき

$S \xrightarrow{*} \varphi' A \psi' \xrightarrow{*} \varphi T \psi$

は一意的に導出される．

II) φ に T を含むとき $S \xrightarrow{*} \varphi T \psi$ において φ の中の右端の T に着目し，$S \xrightarrow{*} \xi T \eta T \psi$ となり，η，ψ の中に T を含まない．一般的に ξ の中に T を含むが，これをくりかえすことにより ξ の中に T を含まないようにすることができる．ここで I) と同様にすれば無曖昧さが証明できる．

〔証明終〕

以上で i)，ii) の手順が無曖昧さを保存することを保証したが，式 (4．6) の算術式では演算子の優先順位は 3 種類で単項演算子は＋，－だけであったが，この方法によると優先順位の種類も任意で，単項演算子の種類も自由に入れられＢＮＦを構成することができる．しかもこのＢＮＦの無曖昧さが保

証されている。

4.5 結　　言

本章では次の2点において拡張型文法の曖昧さを論じた。すなわち

1) 文命令レベルでの命令の拡張

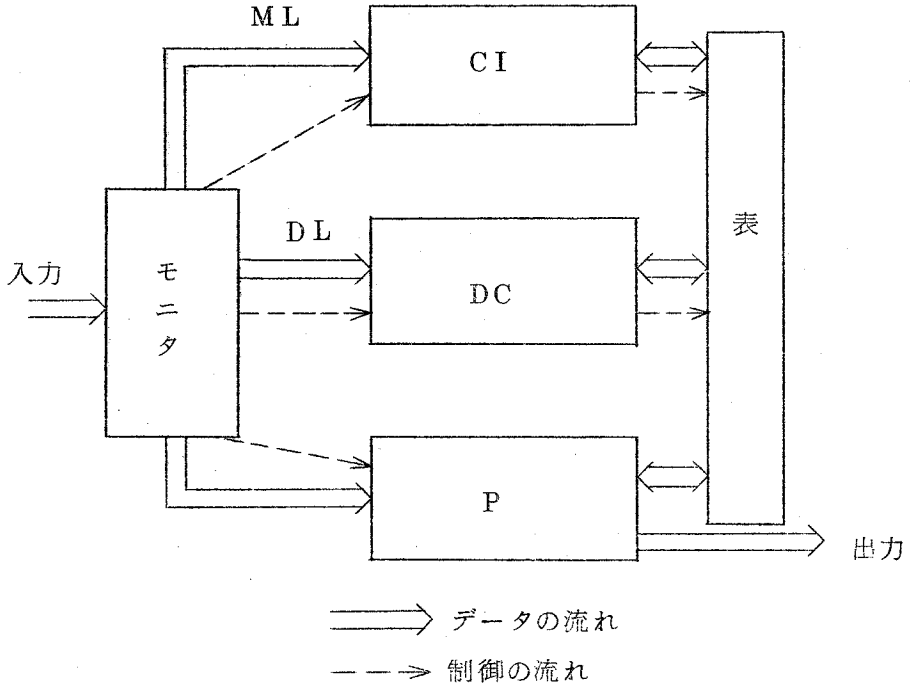
2) 算術式の範囲内で無曖昧な拡張可能な書換規則

1) については第3章のＤＣＳ言語の無曖昧さが保存される十分条件を与え，ＤＣＳの拡張において曖昧さが生じないことを保証した。2) については算術式を無曖昧な形で構成する手順を与えた。これによると無曖昧さが保証され，単項，二項演算子が任意の優先順位で構成できることが示された。さらにこの方法をすすめＡＬＧＯＬ60全体を構成する問題は今後の課題である。

第５章　ＤＣＳの処理系

５．１　緒　　言

第２章でＤＣＳの構成，機能について述べ，第３章でＤＣＳ言語系について述べてきた．すなわちＤＣＳは図５．１の構成になっている．



ＣＩ：Compiler・Interpreter

ＤＣ：Dynamic Compiler

Ｐ：Program

ＭＬ：Meta Language

ＤＬ：Dynamic Language

図５．１　ＤＣＳの構成

本章ではＤＣＳ言語系を処理する処理系，特にＣＩ，ＤＣを構成しているルーチンと表について述べる．

５．２　ＤＣＳ処理系 (27),(31)

５．２．１　構成方針

ＤＣＳ処理系の構成方針を以下に示す．

1）ＣＩは表のみの変更，作成を行なう．

2）ＤＣは表駆動型 (Table driven) 方式とする．

3）ＣＩを拡張が行ない易いようにモジュール化し，それに合わせＤＣもモジュール化する．各モジュールから表を作成する．

図５．２にＤＣＳの処理フローを示す．

各ルーチンの略語と名前の対応は次の通りである．

ＤＳＹＲ：DCS's Syllable read Routine

ＤＬＡＲ：DC's Lexial Analysis Routine

ＤＳＡＲ：DC's Syntax Analysis Routine

ＤＣＧＲ：DC's Code Generation Routine

ＣＩＩＲ：CI's Initialization Routine

ＣＩＣＲ：CI's Control Routine

ＣＤＥＲ：CI's DEclaration Routine

ＣＳＹＲ：CI's SYntax part Routine

ＳＣＰＲ：Syntax Conection Part Routine

ＳＤＰＲ：Syntax Description Part Routine

ＣＡＳＲ：CI's ASsignment part Routine

ＣＳＥＲ：CI's SEmantics part Routine

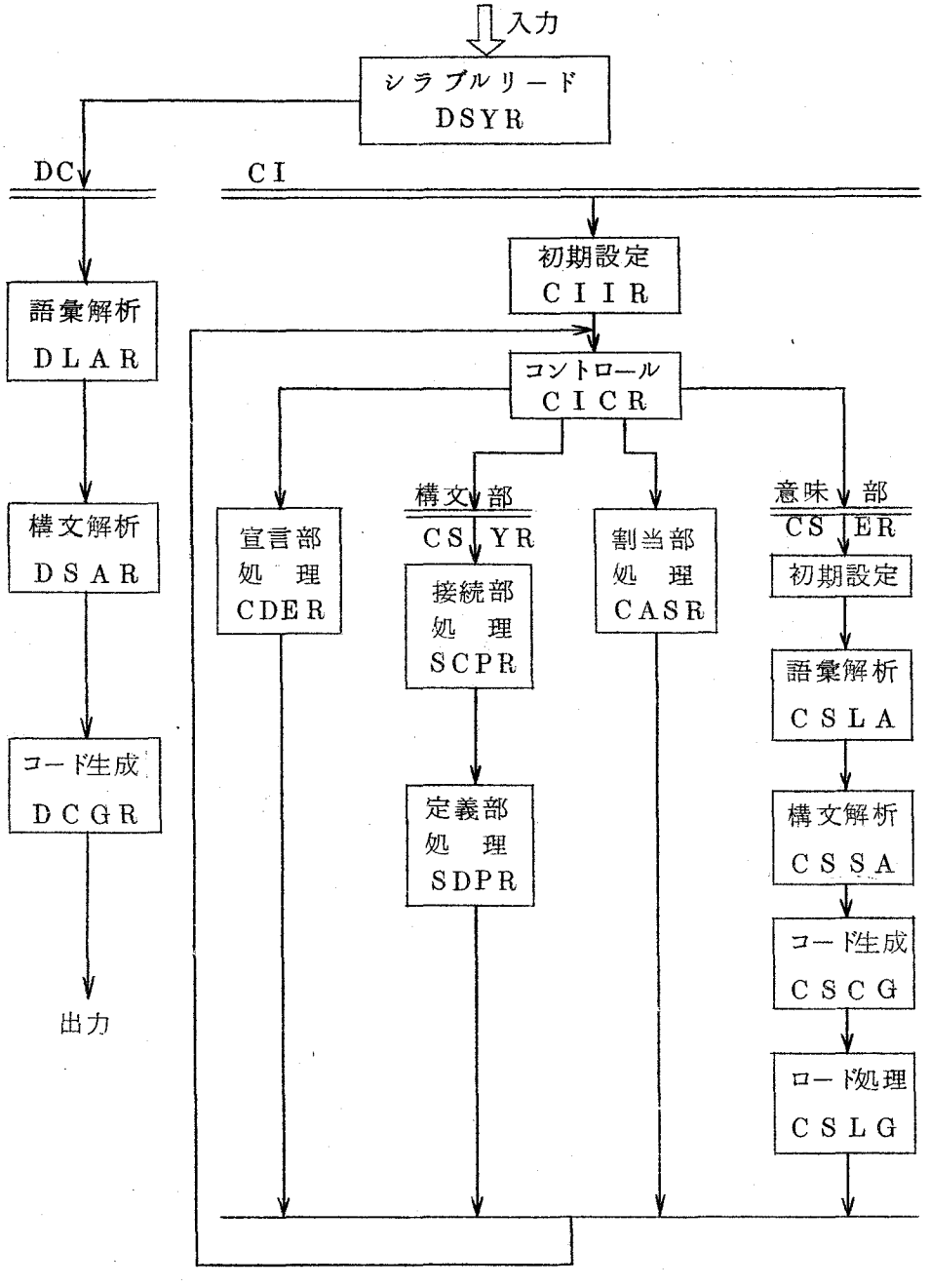


図５．２　ＤＣＳ処理フロー

ＣＳＬＡ：CIś Semantics part Lexial Analysis routine

ＣＳＳＡ：CIś Semantics part Syntax Analysis routine

ＣＳＣＧ：CIś Semantics part Code Generation routine

ＣＳＬＧ：CIś Semantics part Load table Generation routine

ＤＣの各ルーチンで使用される表は次の通りである．

ＤＳＹＲ*：文字，数字，記号表．

ＤＬＡＲ：ＤＣの予約語表，演算子表，区切り記号表；定数表，名札表，変数表．

ＤＳＡＲ：汎構文表（個々の構文表の制御表），構文表（個々の非終端語毎に存在）．

ＤＣＧＲ：ロード表，コード表，被演算子表．

ＤＳＹＲの表はＣＩ，ＤＣによって変更，作成されない．一方DLAR，ＤＳＡＲ，ＤＣＧＲの各ルーチンの表はＣＩによって変更，作成される．しかし，DLARの定数表，名札表，変数表はＤＣによって作成される．

５．２．２　ＤＣＳモニタ

ＤＣＳモニタはＤＣＳの入力をＣＩ，ＤＣＰの各々へ分配する機能をもつ．実際はＤＳＹＲ（ＤＣＳのシラブルリードルーチン）で入力を読み込みシラブル毎に分け，同時にＣＩ，ＤＣの入力の別を判断している．

５．３　ダイナミックコンパイラ（ＤＣ）

５．３．１　構成方針

ＤＣは以下の構成方針に従がう．

＊ DSYRはDCの一部でもあるが以後DCの中のルーチンとして説明してある．

1）図５．２に示したようにＤＳＹＲ，ＤＬＡＲ，ＤＳＡＲ，ＤＣＧＲのルーチンに分ける．各ルーチンは各々中間結果を出力し，それが次のルーチンの入力となる．

2）各ルーチンは表を引くことによって実行される形式になっている．

3）各ルーチンでの拡張は独立に，しかも表のみの拡張で行なえる．

５．３．２　ＤＣの処理系

各ルーチンの説明を行なう．

i）ＤＳＹＲ：シラブルリードの機能を持つ，他にＤＬ，ＭＬの判別としてＤＣ，ＣＩへの入力のふり分けの機能をもつ．この部分に対してＣＩによる拡張は行なわれない．この結果はネームプログラム（ＮＰ）領域（ＤＳＹＲの出力領域）に貯えられる．

ii）ＤＬＡＲ：ＤＳＹＲの出力をＮＰ領域より読み取り，各シラブルを内部コードに変換し，コードプログラム（ＣＰ）領域（ＤＬＡＲの出力領域）に貯える．同時に変数，名札の登録を行なう．

iii）ＤＳＡＲ：ＤＬＡＲの出力をＣＰ領域より読み取り，構文表に従って構文解析を行ない，結果をスタックを使ったポーランド記法の形式で出力する．この出力の形をストリームポリッシュデータ（ＳＰＤ）と名付ける．図５．３はＳＰＤの例である．

iv）ＤＣＧＲ：ＤＳＡＲの出力（ＳＰＤ）を読み取り目的プログラムを生成する．目的プログラムは最も深い位置のポリッシュスタックより作り始め，最も浅い位置のポリッシュスタックに致って終了する．再配置可能コードから終対番地コードへの交換はロード表を使って行なう．

構文表は変更，作成が行なわれ易いように直列型と並列型とがある．

(a)　直列型　＜×＞::＝＜A＞＜B＞＜C＞*

表５．１(a)の表は＜A＞＜B＞＜C＞の解析の順とその処理を示す．＜A＞が終端語のときはその位置情報をスタックに入れ，⊛$_A$ に飛んで＜A＞を解析する．表５．１(a)に示した真の場合の動作表示より，＜×＞は＜A＞，＜B＞，＜C＞の３つが解析されて初めて解析されることが分かる．

＊＜・・・＞でも以後，終端語を表わすことがある．

(b)　並列型　＜×＞::＝＜A＞｜＜B＞｜＜C＞

処理方法は直列型の場合と同じであるが，並列型では＜×＞の解析に於て＜A＞，＜B＞，＜C＞のどれか１つが解析されれば良い．３つとも確認されないとき初めてエラーとなる．

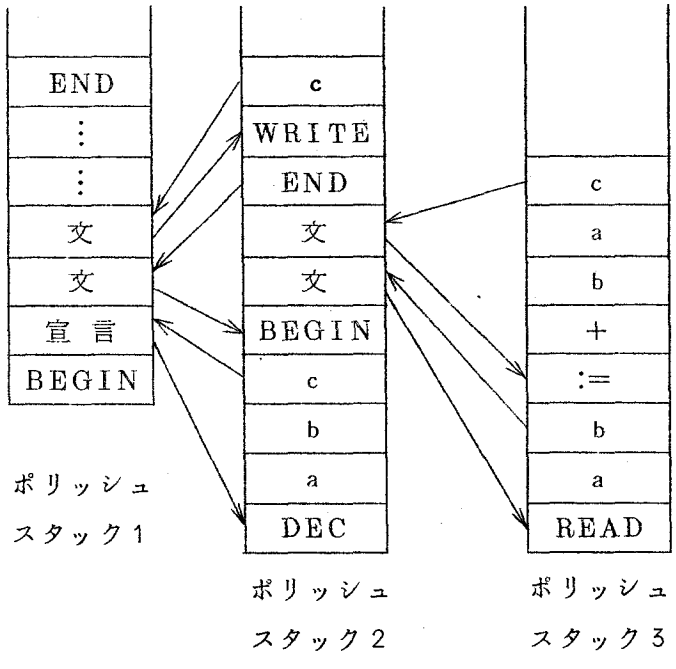


図５．３　ストリーム・ポリッシュデータの例

表５．１　構文表

(a)　直列型

判断ルーチン		真		偽	
ルーチン名	判断のための飛越	仕　事	飛　越	仕　事	飛　越
1.＜×＞	2	＜×＞意味	次のルーチン	エラー処理	
2.＜A＞	⊛$_A$	—	3	エラー処理	1
3.＜B＞	⊛$_B$	—	4	エラー処理	1
4.＜C＞	⊛$_C$	—	1	エラー処理	1

(b)　並列型

判断ルーチン		真		偽	
ルーチン名	判断のための飛越	仕　事	飛　越	仕　事	飛　越
1.＜×＞	2	—	次のルーチン	エラー処理	—
2.＜A＞	⊛$_A$	＜×＞::＝＜A＞意味	1	—	3
3.＜B＞	⊛$_B$	＜×＞::＝＜B＞意味	1	—	4
4.＜C＞	⊛$_C$	＜×＞::＝＜C＞意味	1	—	1

５．４　コンパイラ・インタプリタ（ＣＩ）

５．４．１　構成方針

ＣＩは以下の構成に従がう．

1）構文部，割当部，意味部の各々に対して図５．２に示したような独立な処理部を持つ．

2）ＣＩの行なう拡張操作は表の操作のみである．

次にＣＩがＤＣを拡張するときに，ＣＩが行なわなければならない処理を以下に列挙する．

Ⅰ．語彙解析ルーチンのための各種表の変更，登録．

Ⅱ．構文解析ルーチンのための各種表の変更，登録，作成．

Ⅲ．新しく定義された命令の構文と意味の間の引数表の作成．

Ⅳ．新しく定義された命令のオブジェクトコードの作成．

Ⅴ．再配置可能なオブジェクトコードを実行可能なコードにするための情報を，新しく定義された命令のオブジェクト生成ルーチンに付与．

ＣＩの各処理部は上記の必要な処理を次のように行なう．

○　構文部処理ルーチン・・・・・Ⅰ，Ⅱ

○　割当部処理ルーチン・・・・・Ⅲ

○　意味部処理ルーチン・・・・・Ⅳ，Ⅴ

以上が主な処理であるが，次に具体的に述べる．

５．４．２　ＣＩの処理系

ⅰ）ＣＩＩＲ：ＣＩのロード，作業領域の確保等を行なう．

ⅱ）ＣＩＣＲ：ＭＬで書かれたプログラムはシラブルリードによりＤＬと同じ形式で貯えられるが，このルーチンはそれを順次読み取り，どのルーチンへの入力かを判別し，対応するルーチンへ分配する．

ⅲ）ＣＤＥＲ：ＭＬの宣言部の処理を行なう．ＭＬの定義形式に対して処理ルーチンを決定する．

ⅳ）ＣＳＹＲ：ＭＬの構文部の処理を行なう．すなわちＤＣの語彙解析ルーチン及び構文解析ルーチンの変更，拡張を行なう．構文部は接続部と定義部より成るので，処理系も各々を処理するＳＣＰＲとＳＤＰＲがある．

○　ＳＣＰＲ：図５．４の表を変更，拡張する．非終端語表はＭＬの終

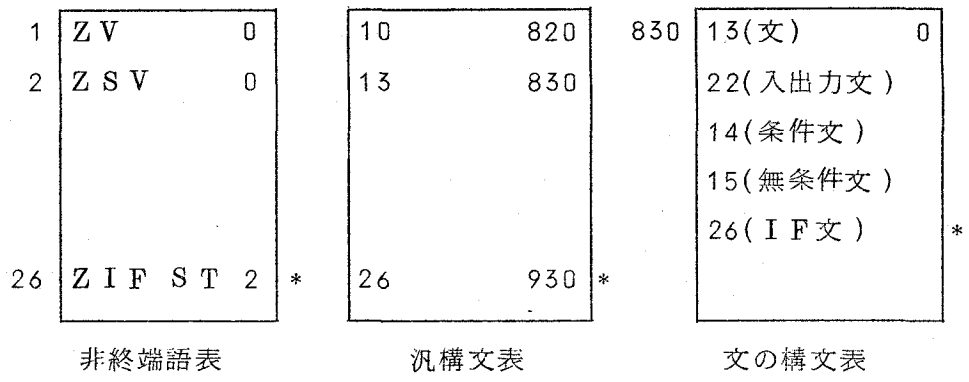
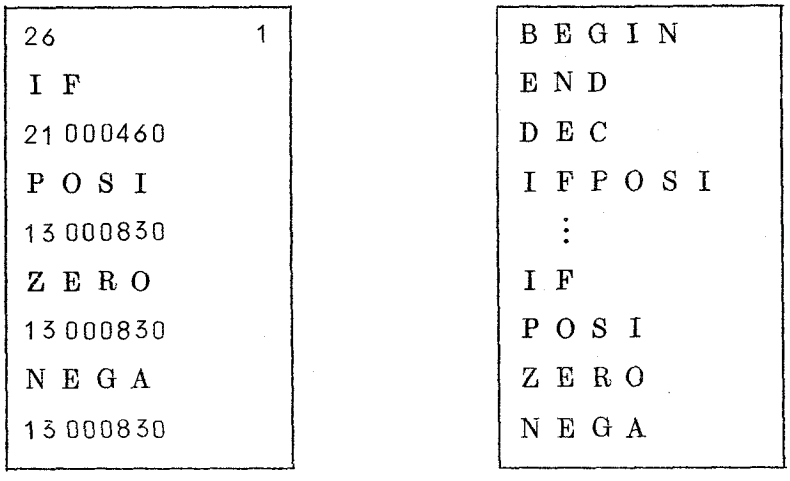


図５．４　ＳＣＰＲが変更する各種表

端語でかつＤＬの非終端語の表である．図３．４(a)の例について説明すると，先ず新しく定義したＩＦ文（ＺＩＦＳＴ）を登録する．このときＺＩＦＳＴに２６の内部コードが割当てられる．２は文レベルの非終端語を表わす．次にＩＦ文の上位の非終端語である文の構文表に２６（ＩＦ文）を登録する．そして新しくＩＦ文の構文表を作成する為の場所（930番地）を確保し，汎構文表に26，930 を，文の構文表には930を登録する．なお右肩の０はこの構文表が並列型であることを示している．

○　ＳＤＰＲ：図５．５に示すようにＤＣの予約語表を変更し，構文表を作成する．ＤＣの予約語表には新しく定義した語（ＩＦ，ＰＯＳＩ，ＺＥＲＯ，ＮＥＧＡ）を予約語として登録し，内部コードを割当てる．ＩＦ文の構文表における２６・・・１の２６は非終端語ＺＩＦＳＴの内部コードであり，１はこの構文表が直列型であることを示す．それ以下では終端語（ＩＦ，ＰＯＳＩ，ＺＥＲＯ，ＮＥＧＡ）は予約語表で定義した内部コードが入り，非終端語のところはその非終端語の内部コードと対応する構文表の先頭番地が入る．



IF文の構文表　　DCの予約語表

図５．５　ＳＤＰＲが変更拡張する表

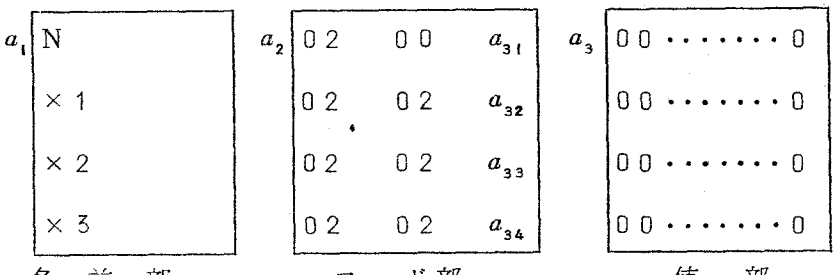


図５．６　仮引数表

v）ＣＡＳＲ：ＭＬの割当部の処理を行なう．すなわち新しく定義された構文の仮引数の処理を行なう．その結果は次のＣＳＥＲで利用される．図３．４の例の場合には図５．６の仮引数表を作成する．これによって仮引数の名前，内部コード，値をつなぎ，その仮引数の種類，すなわち単純変数（００），文（０２）等の情報を入れる．

vi）ＣＳＥＲ：ＭＬの意味部の処理を行なう．ＭＬの意味部は基本的に

はＤＬで記述するので処理もＤＣを使って行なう．これまでのルーチンと異り，ＤＣのルーチンを如何に結合するかが，このルーチンの役割となる．図５．２のＤＣの各ルーチンの制御をＣＩに行なわす．図５．７にその様子を示す．また割当部で定義した変数は意味部では標準手続きと考える．それにより意味部はＤＬからみて構文的に正しいプログラムとなり，ＤＣの各ルーチンの内部を変更しないでパラメータ（各種表）のみ変更すればよいことになる．この様にしてＣＳＣＧを通した結果の出力は絶対番地コードであるので，それを配置可能コードに変換するのがＣＳＬＧの役割である．その為に絶対番地の情報を図５．８のロード表に記す．これにより再配置可能な形になった目

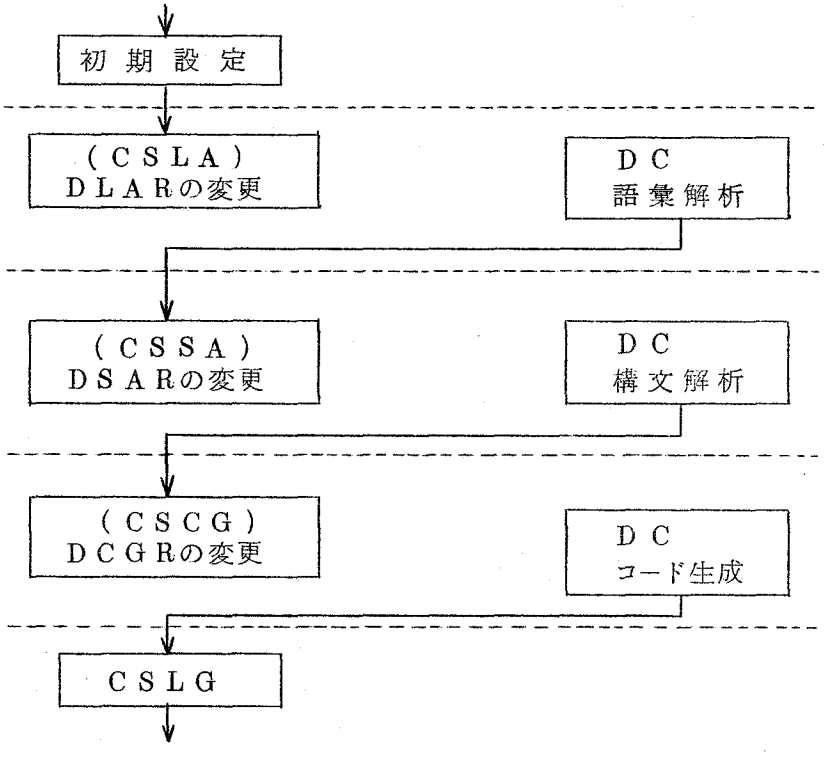


図５．７　ＣＩの意味部処理ルーチン

的コードを意味ルーチンとして登録する．これは新しく定義された命令の構文に対する意味である．

```
0 ···· 140000
0 ···· 160000
0 ···· 180000
1 ···· 050000
1 ···· 130000
1 ···· 120000
1 ···· 060000
1 ···· 150000
99 ··· ····· 9
```

図５．８　ロード表

５．５　結　　言

本章においてＤＣＳの処理系について述べた．処理系の記述においては一般性をもたすために，機械に依存しない形で述べた．そしてＤＣの目的言語についても，ＤＣＳの実現に一般性，汎用性をもたすために，あえて記述しないで，インプリメンターの自由にまかせた．そうすることにより，より計算機に密着した，効率のよい処理系ができると思われる．

第６章　試作システムと評価

６．１　緒　　言

前章までに提案してきたＤＣＳを試作実験した．試作にあたってはＮＥＡＣ２２０６のアセンブラ言語を用いた．また試作システムの評価には静的なステップ数と動的なステップ数を用いて行なう．前者は記憶容量に関係し，後者は実行時間に関係*する．最後にＤＣＳ全体の評価と今後の展望について述べている．

６．２　試作システムの概要

試作システムはNEAC２２０６を用いて行なった．

NEAC２２０６は１０進１２桁（符号ビットとパリティビットを含めて５３ビット）を１ワードとし，４Ｋワードの記憶容量を持つ計算機である．語には図６．１の４種がある．

アキュムレータは３個，インデックスレジスタは１８個あり，演算は浮動

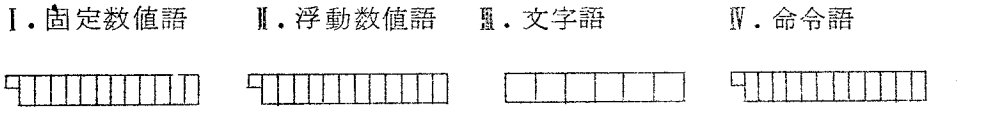


図６．１　NEAC ２２０６の命令

＊ＮＥＡＣ２２６の場合，記憶容量の単位はワードであり，一命令を実行するのに10μs～100μsと考えてよい．

小数点，固定小数点の両方式の演算が可能である．

インプリメンテーションはアセンブラ言語を使用した．この言語は機械語と一対一に対応し，相対番地，文字データの入力等が可能である．また，特殊命令として表のルックアップ命令をもつ．

第５章でのＤＣＳ処理系を上記の計算機で実現した，ＣＩ，ＤＣの本体は紙テープでもって主記憶装置（４Ｋワード）へ入力する．実験はキーボードより使用者が手でもって入力する．また使用者は紙テープでもって入力することもできる．

６．３　試作システムの評価

a）静的なステップ数による評価

表６．１に(a)ＤＣ，(b)ＣＩの静的なステップを示す．

静的なステップ数でみるとＣＩとDC_B^*のステップ数の和は３Ｋワードとなり，その比は１：４である．この比の値の小さいのはＭＬの能力として文命令の定義形式に限定した為である．これ以外に他の形式（例えば複合命令，制御機構の拡張，定義等）を加えるとこの比の値は大きくなる．

b）動的なステップ数による評価

ＣＩについて図３．４(a),(b) の例の場合，表６．２に示すようになる．

DC_E（付加された命令を処理するために増えたダイナミックコンパイラ）を作成する時間，すなわち動的なＣＩのステップ数は，

$$y = 170x + 200 + \alpha$$

＊ DL_Bに対するダイナミックコンパイラ

表６．１　静的なステップ数

(a)　　ＤＣ

	ステップ数	小　　計
ＤＳＹＲ（シラブルリード）	２４５	３０１
ＤＳＹＴ（シラブルリード表）	５６	
ＤＬＡＲ（語彙解析）	１９２	５２０
ＤＬＡＴ（語彙解析表）	３２８	
ＤＳＡＲ（構文解析）	５５３	６８５
ＤＳＡＴ（構文解析表）	１３２	
ＤＣＧＲ（コード生成）	３４４	７２６
ＤＣＧＴ（コード生成表）	３８２	
ＭＴＣ　（磁気テープ制御）		１５２
合　　計		2,３８４

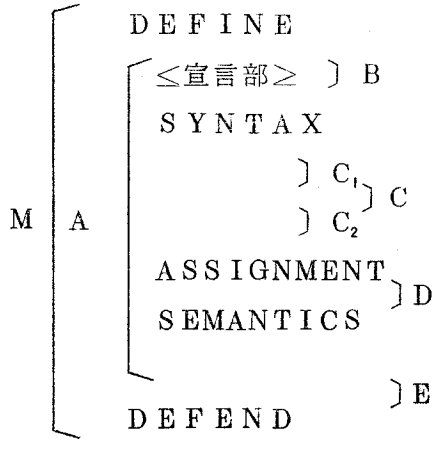
(b)　　ＣＩ

各　ル　ー　チ　ン	ステップ数
Ⅰ．ＣＤＥＲ　（宣言部処理）	１８
Ⅱ．ＣＩＣＲ　（コントロール）	４８
Ⅲ．ＣＳＹＲ　（構文部処理）	１２７
ＳＣＰＲ（接続部処理）	５２
ＳＤＰＲ（定義部処理）	７５
Ⅳ．ＣＡＳＲ　（割当部処理）	４５
Ⅴ．ＣＳＥＲ　（意味部処理）	１７６
初　期　設　定	３０
ＣＳＬＡ（語彙解析）	３０
ＣＳＳＡ（構文解析）	３５
ＣＳＣＧ（コード生成）	４１
ＣＳＬＧ（ロード処理）	４０
小　　計	４１４
飛　越　表	３０
非　終　端　語　表	２５
予　約　語　表	６
エ　ラ　ー　ル　ー　チ　ン	１０
ラ　ベ　ル　表	３０
小　　計	１０１
ＣＩ　合　　計	５１５

表６．２　ＣＩの動的なステップ数

各　ル　ー　チ　ン	プログラム部分	例　1	例　2
Ⅰ.読　み　込　み	M		
（ＤＣのＳＹＲ）	—	—	—
Ⅱ.ＣＩＣＲ	A	65	73
Ⅲ.ＣＤＥＲ	B	17	17
Ⅳ.ＣＳＹＲ	C	312	414
ＳＣＰＲ	C_1	50	50
ＳＤＰＲ	C_2	262	364
Ⅴ.ＣＡＳＲ	D	160	200
Ⅵ.ＣＳＥＲ	E	300	339
アドレス決定		30	30
ＣＳＬＡ	E	25	25
（＋ＤＬＡＲ）	—	—	—
ＣＳＳＡ	E	38	38
（＋ＤＳＡＲ）	—	—	—
ＣＳＣＧ	E	45	45
（＋ＤＣＧＲ）	—	—	—
ＣＳＬＧ		162	201
計		854＋α	1,043＋α

注：表でプログラム部分とは右図の定義プログラムのどの部分を対象とするかを示す．又，一線部分はＤＣの行なうべき動作の部分である．



で与えられる．但しxはプログラム中における語の数，定項項200は定義命令の共通部分であり，αは意味部でのＤＣの動的なステップ数である．例１のＩＦ文では（８５４＋α），例２のＦＯＲ文では（1043＋α）である．両者ともＭＬプログラムの処理は１秒以内で終り，ＤＣＳのオンラインでの使用は可能である．

次にＤＣのステップ数についてみると図６．２のようになる．

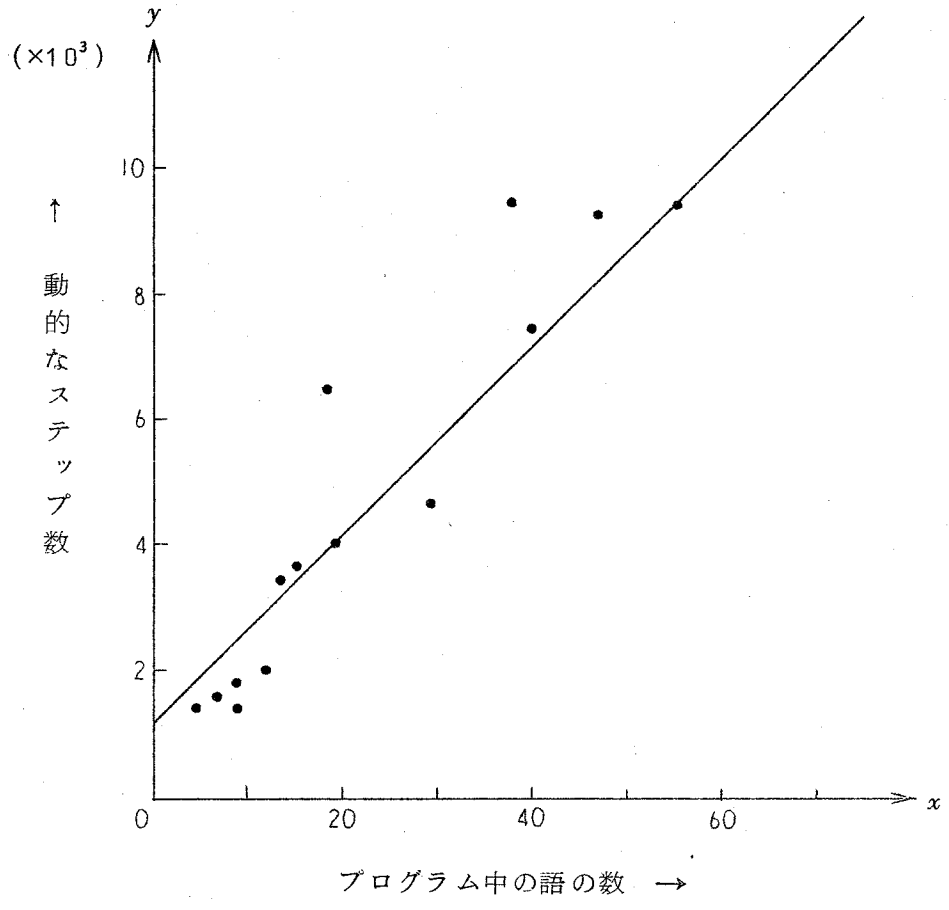


図６．２　ＤＣにおける入力語数と動的なステップ数の関係

グラフより，語数と動的ステップ数の関係はほぼ一次式

$$y = 155x + 1000$$

で近似できる．このことより，ＤＣは直列型，並列型の構文表にしたにかかわらずほぼ満足する動的なステップ数で動作することがわかった．

他の評価の結果をまとめると次のようになる．

1）DC_Bに対するDC_Eの比率はＩＦ文ＦＯＲ文合わせて５％弱である．これはＤＣの各々の文命令に対する共通部分が多く，新しく文命令を付加してもさほど記憶容量をとらないことを示している．

2）新しく付加した文命令と既存の文命令とのコンパイル時間はほぼ等しい．

3）新しい命令を次々と付加する場合，記憶容量（静的なステップ数）は線形に増加するのみであり，処理時間（動的なステップ数）は既存の命令と余り変らない．このことはＤＣＳの特徴として付加された命令は核となる命令と同レベルになることによる．

６．４　ＤＣＳの評価と将来の展望

評価をまとめると次のようになる．

1）コンパイラそのものを逐次的に変更，拡張することにより，新しく文命令を付加しうるという初期の目的は一応達成された．文命令を定義するにあたって構文部，割当部，意味部と分け，構文部，割当部においては対象言語の非終端語まで操作できるようにした．これにより付加する命令の構文まで使用者が定義でき，きめの細かい拡張が可能となった．

2）定義例ではＩＦ文，ＦＯＲ文を扱ったが，たとえば一つのサブプログラムを意味部で書き，それを呼び出す命令を定義することができる．例

として，二次方程式を解く文として

NIJI＜変数＞NIJIKO＜変数＞ICHIJI＜変数＞NOKAI＜変数＞DEARU

のように文を定義して，初めの三つの＜変数＞に数を与え，最後の＜変数＞に解を求める文命令が定義できる．

3）処理系は変更，拡張が行なわれやすいように，構文表等，種々の表を作成した．特に構文表は直列型と並列型にし，ＤＬの書換規則を全てこの形にした．それによってＤＣの構文の拡張は容易になり，表のみの拡張で行なえた．

4）試作した処理系の評価には静的なステップ数と動的なステップ数を与えた．動的なステップ数でみるとＣＩが一つの命令を拡張する時間はその定義命令の語数に比例し，さほど大きなステップ数とならない．このことにより，DCSを他の計算機で実現する場合，処理時間は試作の場合より早くなり，実用システムが作成可能である．

次に今後の展望をまとめると次のようになる．

1）３．２節で述べた他の拡張（複合命令等）および２．２節で述べた制御機構の拡張等をＤＣＳに付加する事が望まれる．その為にはＤＣＳを実現するための計算機の選定がまず必要であろう．

2）構文的な命令に限定しても，構文の接続の仕方（３．２節での a）並列付加型，b）並列挿入型，c）直列付加型，d）直列挿入型）で本論文では a）の並列付加型を採用したが他の型を使用した場合，また，構文の定義をさらに一般化した場合の拡張方法，およびそれらの構文接続，定義に対しての無曖昧さの保証等があり，今後さらに研究する必要がある．

3) 核言語として本論文ではアルゴルのサブセットを選び，ＡＬＧＯＬ風言語にしたが，各目的に応じ，核言語を選ぶことができる。理想的には各使用者が（または各端末毎に）各自の数種の核言語をもち，使用者自身に合った拡張を行ない，コンパイラを私有化する。いわゆるパーソナルコンパイラにするとよい。ＤＣＳはこの方向に対して一つの指唆を与えたと思う。

６．５　結　　言

本章において試作システムの評価とＤＣＳの評価，および将来の展望について述べた。

これによるとＤＣＳは既存のコンパイラとさほど変わらず，しかも使用者自身がコンパイラそのものを拡張できることがわかった。

第７章　結　　論

本論文の第２章において，使用者がＴＳＳを用いて端末から自分自身のコンパイラを実時間で拡張するシステム―ダイナミックコンパイラシステム（ＤＣＳ）―を提案した．ＤＣＳはコンパイラ・インタプリタを用いて，コンパイラを逐次的に書換えていく方式を採用した．ＤＣＳはＣＩ(Compiler Interpreter)，ＤＣ (Dynamic Compiler)，Ｐ (Program)の３つのレベルよりなり，他にＤＣＳモニタ，表よりなる．

ＣＩ，ＤＣの入力言語をＭＬ (Meta Language)，ＤＬ (Dynamic Language)と呼ぶが，第３章においてＭＬ，ＤＬの言語系について述べた．ＭＬはＤＬに文命令を付加する定義命令である．定義命令には構文部，割当部，意味部があり，それぞれにおいて定義する命令の構文，構文部と意味部での変数間の関係，意味を与えた．構文部，割当部ではＤＬの非終端語まで操作でき，きめの細かい拡張が行なえた．

第４章では，ＤＣＳ言語系での拡張において無曖昧さが保存されることを保証した．すなわち，ＭＬでもってＤＬの書換規則を定義するが，そのときＤＬの構文を記述している書換規則に制限をつけることにより，曖昧さが生じない事を証明し，ＤＣＳにおいて構文的に曖昧さのない拡張が行なえる保証を与えた．

第５章ではＤＣＳ処理系について述べた．ＣＩ，ＤＣの構成方針を示し，具体的に各ルーチン，表の構成方法を与えた．ＣＩはＤＣの構文表等の変更，作成を行ない，ＤＣはそれらの表に従って動作する表駆動型のコンパイラである．新しく定義する命令の意味は既存のＤＬを用い，処理もＤＣで行なった．その結果，意味部の記述，処理が容易になった．

上記システムの試作をNEAC2206で行なった．第6章においてその試作，実験の結果を述べ，評価を行なった．まとめると次のようになる．

○　新しい命令を次々と定義する場合，記憶容量は線形に増加するのみである．

○　ＣＩの定義命令の処理時間はだいたい1秒以内で終る．

○　新しく定義された文命令と既存の文命令とのコンパイル時間はほぼ等しい．

○　ＤＣの処理系は直列型，並列型の構文表にしたにもかかわらず，処理時間（コンパイル時間）は他のコンパイラとあまり変わらない．

以上のように試作システムでは満足できる結果が得られた．

本論文によって使用者自身がオンライン・リアルタイムで，コンパイラの使用と作成を同時に行ない，各目的に合ったパーソナルコンパイラへの方向を与えた．

今後ＤＣＳを実用システムにする際，記憶容量，処理時間の問題よりも，能力的な点，例えば複合文の定義，制御機構の定義等を導入するのが大きな問題として残こるだろう．

謝　　辞

本研究にあたって，終始懇切なる御指導，御鞭撻を賜わりました手塚慶一教授に対し衷心より御礼申し上げます．

大学院において御指導，御教授賜わった笠原芳郎名誉教授，ならびに電子工学教室の尾崎弘教授に厚く御礼申し上げます．

大学院修士，博士両課程において御指導，御教授いただきました通信工学教室の板倉清保教授，滑川敏彦教授，熊谷信昭教授，中西義郎教授に対し厚く厚く御礼申し上げます．

また有益な御助言，御討論いただいた笠原正雄助教授，真田英彦講師に深謝します．

筆者の属する手塚研究室の打浪清一助手には熱心な御討論，御教示をいただき，また後舜堂氏，手塚研究室の中西暉助手，近畿大学の梶谷講師をはじめ，研究室の諸氏には種々の面でお世話になった．特に，大学院学生広瀬嘉昭氏，篠原健氏，手塚正義氏，海尻賢二氏，福永正博氏，早瀬憲一氏，妹尾孝憲氏，大倉二郎氏および本学卒業生の近藤恵氏，森島浩人氏，小田光穂氏には有益な御討論と御協力をいただいた．また，京都大学の長谷川利治教授，基礎工学部の豊田順一助教授，北橋忠宏助手，大阪府立大学の鶴身征雄助手，京都大学の宮原秀夫助手，電々公社武蔵野通研の野村浩郷氏には有益な御助言をいただいた．また，応用物理学教室の安井裕助教授には文献でお世話になった．

ここに記して以上の方々に深く感謝の意を表する．

文　　献

(1) P.J.Brown: "A survey of macro processors" Annual Review in Automatic Programming, Vol.6 Pergamon Press. Oxford and New York, pp.37−88 (1969)

(2) B.M.Leavenworth: "Syntax macro and Extended translation" Comm. ACM, Vol.9, No.11, pp.790−793 (1966)

(3) B.A.Galler and A.J.Perlis: "A proposal for definitions in ALGOL" Comm. ACM, Vol.10, No.4, pp.204−219 (1967)

(4) T.E.Cheatham: "The introduction of definitional facilities into higher level programming languages" Proc. AFIPS 1966 Fall Joint Computer Conference, Vol.29, pp.623−637

(5) J.V.Garwick: "GPL, a truly general purpose language" Comm. ACM, Vol.11, No.9, pp.634−638 (1968)

(6) E.T.Irons: "Experience with an extensible language" Comm. ACM, Vol.13, No.1, pp.31−40 (1970)

(7) J.C.Reynolds: "GEDANKEN−a simple typeless language based on the principle of completeness and the reference concept" Comm. ACM, Vol.13, No.5, pp.308−319 (1970)

(8) B.Wegbreit: "The ECL programming system" proc. AFIPS 1971, Fall Joint Computer Conference, pp.253−

(9) S.A.Schuman and P.Jorrand: "Definition mechanisms in extensible programming languages" Proc. AFIPS 1970, Fall Joint Computer Conference, pp.9-20

(10) B.N.Dickman: "ETC-An extendible macro-based Compiler" Proc. AFIPS 1971, Spring Joint Computer Conference, pp.529-538

(11) M.C.Harrison: "BALM-An extendable list-processing language" Proc. AFIPS 1970, Spring Joint Computer Conference, pp.507-511

(12) B.Wegbreit: "Multiple evaluators in an extensible programming system" Proc. AFIPS 1972, Fall Joint Computer Conference, pp.905-915

(13) E.Milgrom and J.Katzenelson: "Date structuers in the extensible programming language AEPL" Proc. AFIPS 1972, Fall Joint Computer Conference, pp.515-523

(14) G.Molnar: "SEL-A self-extensible programming language" Comp. J., Vol.14, No.3 pp.238-242 (1970)

(15) V.Schneider: "Some syntactic methods for specifying extendible programming languages" Proc. AFIPS 1969, Fall Joint Computer Conference, pp.145-156

(16) T.E.Cheatham, JR., A.Fischer and P.Jorrand: "On the basis for ELF-An extensible language facility"

Proc. AFIPS 1968, Fall Joint Computer Conference, pp.937 -948

(17) 山本，打浪，手塚：“ダイナミックコンパイラシステムに関する一考察” 信学会 電子計算機研資 EC70-48 (1971-02)

(18) 山本，打浪，手塚：“ダイナミックコンパイラに関する一考察” 信学全国大会 1023 (昭 46)

(19) 山本，打浪，手塚：“ダイナミックコンパイラシステムに関する一考察 その２” 信学全国大会 1211 (昭 47)

(20) 萩原 他：“Compiler記述言語：COL” 情報処理，Vol.9，No.4 pp.187-196 (1968)

(21) 萩原 他：“COLで書かれたCompiler について” 情報処理，Vol.10，No.6 pp.375-383 (1969)

(22) 小久保 他：“コンパイラ記述用言語 BPL” 情報処理，Vol.6，No.11，pp.342-349 (1970)

(23) J.A.Feldman: “A formal semantics for computer language and its application in a compiler-compiler” Comm. ACM，Vol.9，No.1，pp.3-9 (1966)

(24) 井上：“コンパイラ・コンパイラ” コンピュータサイエンスシリーズ 産業図書 (1970)

(25) 山本，打浪，手塚：“自己拡張型コンパイラに関する一考察” 信学全国大会 936 (昭 45)

(26) 山本，海尻，打浪，手塚：“拡張型言語の一形式とその処理系－ダイナミックコンパイラシステム－”信学論文誌 D (1974)

(27) 海尻，山本，打浪，手塚：“ＤＣＳ言語とその処理系”信学会

オートマトンと言語・パターン認識と学習研資 AL72-41・PRL 72-39 (1972-07)

(28) 山本，打浪，手塚："拡張型文法の曖昧性について"信学会 オートマトンと言語・パターン認識と学習研資 AL72-98 PRL72-99 (1973-01)

(29) 山本，打浪，手塚："ダイナミックコンパイラシステムにおける無曖昧文法について"信学全国大会 1182 (昭 48)

(30) 海尻，山本，打浪，手塚："ＤＣＳプロセッサの構成について" 信学全国大会 1386 (昭 48)

(31) 山本，海尻，妹尾，打浪，手塚："ダイナミックコンパイラシステムとその評価"信学会 電子計算機研資 EC73-19 (1973-06)

(32) 竹下："PL/I 複合プログラミング言語"日本経営出版会 (1970)

(33) A Van Wijngaarden: "Report on the Algorithmic Language ALGOL 68" Numerische Mathematik, Vol.14, pp.79-218 (1969)

(34) C.H.Lindsey et al: "Informal introduction to ALGOL 68" North-Holland publishing company (1971)

(35) V.B.Schneider: "A translation grammar for ALGOL 68" Proc. AFIPS 1970, Spring Joint Computer Conference pp.493-505

(36) P.Branquart et al: "The composition of Semantics in ALGOL 68" Comm. ACM, Vol.14, No.11, pp.697-708 (1971)

(37) 島内："プログラム言語論—ALGOL 60からALGOL Nへ—"

電子計算機基礎講座5，共立出版 (1972)

(38) 斉藤 他："ESDL (ETL's Systems Description Language) について" 電気試験所彙報 第34巻 第5，6号 pp.354-367 (1970)

(39) N.Chomsky and M.P.Schützenberger: "The algebraic theory of context-free languages" Computer Programming and Formal System pp.118-161 (1963)

(40) M.Gross: "Inherent ambiguity of minimal linear grammars" Information and Control, Vol.7, pp.366-368 (1964)

(41) H.A.Maurer: "A direct proof of the inherent ambiguity of a simple context-free language" Jour. ACM, Vol.16, No.2, pp.252-260 (1969)

(42) V.Fabian: "Structual unambiguity of formal language" Czech. Math. J. Vol.89, pp.394-430 (1964)

(43) J.Gruska: "On structual unambiguity of formal languages" Czech. Math. J. Vol.90, pp.283-294 (1965)

(44) J.A.Feldman and D.Gries: "Translator writing systems" Comm. ACM, Vol.11, No.2, pp.77-113 (1968)

(45) T.E.Cheatham, Jr.: "Syntax-directed compiling" Proc. AFIPS 1964, Spring Joint Computer Conferens, Vol. 25, pp.31-57

(46) F.R.A.Hopgppd: "Compiling Techniques" American Elsevier Publishing Company, INC. (1969)

(47) D.Gries: "Compiler Construction for Digital Computers" John Wiley & Sons, INC. Chapter 1

(48) J.W.Backus et al: "Revised Report on the ALgorithmic Language ALGOL 60" Comm. ACM, Vol.6, No.1 pp.1-17 (1963)

(49) 井上："ソフトウェアの自動作成" 情報処理, Vol.10, No.4 pp.191-197 (1969)

(50) 中田："コンパイラの技法" 竹内書店 (1971)