2. Compiler Generating System

藤野喜一（早稲田大学）

Compiler Generating System（C.G.S.）

1. Introduction
2. Compilerの表現
3. Compiler Generating Systemの表現
4. Compilerの構成
5. C.G.S.の方式
6. C.G.S.の文法
7. INPUT DATA作成の注意
8. C.G.S.本体の設計について
9. CS(A, B)作成の手順

§1. INTRODUCTION

特定のDigital Computerに対してCompilation System（C.S.と略記する．Compilerという）を作成するには多くの労力とかなりの時日（5～30 man years）を必要とする．

一般に使われて来た方法は，対象とするComputerのMachine language（Assemblerも含む）でC.S.のProgramを記述する．この方法は，そのComputerのmachineの特性（即ちmachine codeのもつ特性）を最大限に利用できるから作成されるC.S.は極めてoptimalである長所を有するが，時間，労力その他の困難を伴うものである．

したがつて，もつと容易に，かつ短時日でできるだけ高い機能をもつC.S.を作成するための方式の研究が望まれる．

この考えを実現するためのProgram即ちC.S.の自動設計を行うProgramをCompiler Generating System（C.G.S.と略記する）とよぶ．

この論文はC.G.S.の基本的な考え方とその実現の方式及び例題を述べたものである．

尚この研究に当つて種々有益な御指導御助言を戴いている早大理工学部数学教室野口広助教授に深く感謝します．

又この実験に当り多くの計算機時間を割当て下さった早大電子計算室長難波正人教授に

感謝します．

§2. Compilerの表現

§2.1 C.S.の表現

ここでは，Compiler（i.e. C.S.）の性格を明白にするための記号的な表現を次のように行う．

まず．Automatic Programing languageの一つを$L(i)$（ただし，iはi番目の種類を示す）とかき，$L(i)$のsetを

$$\{L\}=\{L(i) \mid i\in I,\ I=(1,2,\cdots,l)\}$$

とかく．ALGOL，FORTRAN及びNUMERIC等は$L(i)$の例である．

Digital Computerの一つを$M(i)$，そのmachine languageを$\tilde{L}(i)$とし，$M(i)$，$\tilde{L}(i)$のsetを夫々，

$$\{M\}=\{M(i) \mid i\in J,\ J=(1,2,\cdots,m)\}$$
$$\{\tilde{L}\}=\{\tilde{L}(i) \mid i\in J,\ J=(1,2,\cdots,m)\}$$

とかく．ここで一つの$M(i)$に対する$\tilde{L}(i)$は一つだけと仮定する．

以上の記号を使用すれば，Compiler（C.S.）は一般に$CS(x,y,z)$で表現される．たゞし添字(x,y,z)は次の意味をもつ．

xの意味：C.S.のInput Dataを記述するlanguage（Source language）が$L(x)$である．

yの意味：C.S.の対象とするMachineが$M(y)$である．即ちC.S.自身が$\tilde{L}(y)$で記述されている．

zの意味：C.S.のOutput Data（Object program）が$\tilde{L}(z)$でかゝれる．

Machine $M(j)$に対する通常のCompilerは$CS(i,j,j)$で表現される．即ちC.S.自身，及びそのobject programの記述languageが同一であることを示す．この場合には単に$CS(i,j)$とかくことがある．

§2.2 C.S.の機能

次に$L(i)$が$CS(i,j,k)$のinput languageであるとき，特に$L(i,j)$とかく．$M(j)$の許容するlanguage $\tilde{L}(j)$のsymbolでかかれることを示す．

あるProgram Pが$L(i,j)$でかかれるとき$P(i,j)$とかく．$CS(i,j,k)$は，$P(i,j)$を変換してobject program $\tilde{P}(i,k)$を作り，$\tilde{P}(i,k)$は$\tilde{L}(k)$で記述されそのsource languageが$L(i)$なることを示す．よつて$CS(i,j,k)$の機能を$f(i,j,k)$とすれば次の式を得る．

$f(i,j,k)P(i,j)=\widetilde{P}(i,k)\subset P(\cdot k)$

$\left(P(i,j)\ \boxed{\xrightarrow{CS(i,j,k)}}\ \widetilde{P}(i,k)\right)$

$P(\cdot k)$ は $\widetilde{L}(k)$ でかかれたProgramを示す.

いま数学的に同一の問題 P を $L(i,j)$, $L(k,j)$ でかいたとき，それぞれ $P(i,j)$, $P(k,j)$ とし，これが同一問題を示すことを，$P(i,j)\equiv P(k,j)$ とかく.

$P(i,j)$ と $P(k,j)$ とではprogramの構造が一般に異るから $P(i,j)\to P(k,j)$ への変換がなければならない.（P を表わす数学的な flow chartを仲立にすればよい）これを $\varphi(i,k\,|\,j)$ とかく.

Compiler $CS(i,j,j)$ と $CS(k,j,j)$ の変換を夫々 $f(i,j,j)$, $f(k,j,j)$ とすれば

$$f(i,j,j)\,P(i,j)=\widetilde{P}(i,j)$$
$$f(k,j,j)\,P(k,j)=\widetilde{P}(k,j)$$

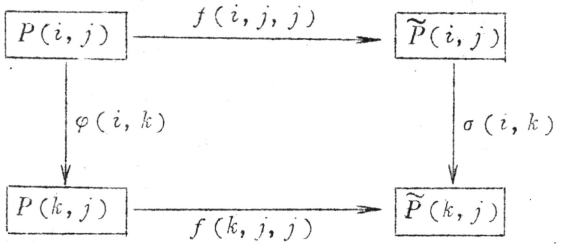
であるから，$\widetilde{P}(i,j)$ と $\widetilde{P}(k,j)$ は数学的に同一であるはずである．よつて $\widetilde{P}(i,j)\to\widetilde{P}(k,j)$ の変換を $\sigma(i,k,j)$ とすれば，

$$\sigma(i,k,j)\,f(i,j,j)\,P(i,j)=\sigma(i,k,j)\,\widetilde{P}(i,j)=\widetilde{P}(k,j)$$
$$f(k,j,j)\,\varphi(i,k,j)\,P(i,j)=f(k,j,j)\,P(k,j)=\widetilde{P}(k,j)$$

が成立しなければならない．この関係から

$$\underline{\sigma(i,k,j)\,f(i,j,j)=f(k,j,j)\,\varphi(i,k,j)}\quad\circledast$$

となる.



$\underline{P(i,j)\to P(k,j)\text{の変換}\varphi(i,k)}$ 及び

$\underline{\widetilde{P}(i,j)\to\widetilde{P}(k,j)\text{の変換}\sigma(i,k)}$ の存在を仮定すれば，二つの Compiler $CS(i,j,k)$ と $CS(k,j,j)$ の機能 $f(i,j,j)$, $f(k,j,j)$ は関係⊛を満足しなければならない.

この関係は，任意のMachineに対するPrograming System 作成の際の基本的な仮定である.

特に $\underline{\varphi(i,k)\text{ を Language Translation}}$

$\sigma(i,k)$ を Machine Language Translation

といい，$\varphi(i,k)$, $\sigma(i,k)$ を実現したProgramを夫々Language translator, Machine language translatorと呼び，$\Psi(i,k)$, $\Sigma(i,k)$ とかく，Ψ, Σ はtranslationの実際的な証明といえる．

§2.3 Selfexpressibleな C.S.

あるComputer $M(j)$ に対する一つのCompiler $CS(i,j,j)$ を仮定するとき，$CS(i,j,j)$ 自身をその source language $L(i,j)$ によつて，記述したProgram $P(CS(i,j,j),i,j)$ を $CS(i,j,j)$ にinputすれば，そのobject program $CS^*(i,j,j)=\widetilde{P}(CS(i,j,j),i,j)$ が $CS(i,j,j)$ と同一の機能と構造を有するCompilerであるとき，この $CS(i,j,j)$ は自己表現可能(self expressible) であるという．

通常のCompilerは必ずしもself−expressibleではない．但し $M(j)$ による特性及びmemoryの制限を一切無視すれば，Compiler は全て広い意味での自己表現可能であると考えられる．

広義でない自己表現可能なCompilerであるためには，Source language $L(i,j)$ が $\widetilde{L}(j)$ を含めば十分である．

Compilerの自動設計にとつて，self−expressibleな性質は重要な手がかりである．

§3. Compiler Generating Systemの表現

§3.1 C.G.S.の一般的な表現

Compiler Generating System C.G.S.は記号的に五つのparameterを用いて

$$CGS(x,y,z|U,j)$$

と表現される．parameter x,y,z,U,j は次の意味をもつ．

U はC.G.S.の入力言語が $L(u)$ であること，

j はC.G.S.を記述するlanguageが $\widetilde{L}(j)$ 即ちC.G.S.がMachine $M(j)$ をbaseとすることを示す．又 x,y 及び z はC.G.Sにとつてformal parameterであつて，x, y, z にactual parameter A,B,C が代入されるとき，Compiler $CS(A,B,C)$ がC.G.S.によつてproduceされることを意味する．

C.G.S.の入力Dataは次のものから成立つ．

(i) $CS(x,y,z)$ の $L(u)$ による一般的な表現

(ii) x,y,z のactual parameter A,B,C

(iii) A,B,C に対してC.G.S.が要求するactual parameterのset $\alpha(A,B,C)$

C.G.S.の出力Dataは

(i) $CS(A, B, C)$

(ii) $CS(A, B, C)$のword数その他必要な情報

から成る.

$CGS(x, y, z|U, j)$においてinput language $L(u)$, Machine $M(j)$は一般に固定されるから便宜上$CGS(x, y, z|U, j)$の代りに単に

$$CGS(x, y, z)$$

とかく.

$CGS(x, y, z)$の出力としてえられる$CS(x, y, z)$は$\widetilde{L}(y)$で記述されたprogramであるから, 見方を変えれば, $CGS(x, y, z|U, j)$はinput languageが$L(u)$, baseが$M(j)$, output languageが$\widetilde{L}(y)$である.

従つてyを固定するときには, $CGS(x, y, z|U, j)$の機能はCompiler $CS(U, j, y)$と一致する.

この性質によつてC.G.S.はC.S.の機能の他に更に, output language $\widetilde{L}(z)$に関係するpartが自由に変更され得る機能をもたねばならないことがわかる.

又parameter x, zによつて定まる情報は全て$L(u)$によつてかかれる$CGS(x, y, z)$のsource programの中に於て, constantではなくvariableとして表現される事の必要性が分る.

かくして, $CGS(x, y, z)$の実現の問題にとつては, $CS(x, y, z)$の構造の明確化が必要である.

§4. Compilerの構成と機能

この節では, Compiler $CS(X, Y, Y)=CS(X, Y)$の構成と機能について$CGS(X, Y)$に必要な範囲内で取扱う.

§4.1 $CS(X, Y)$の構成

$CS(X, Y)$の性格決定要素は次のものから成る.

(1) $L(X, Y)$の文法

(2) $L(X, Y)$のProgramming Word

(3) $\widetilde{L}(Y)$の文法

(4) $\widetilde{L}(Y)$のProgramming Word

(5) $\widetilde{L}(Y)$のLoading subroutine

(6) Library (X, Y)

(7) $\widetilde{\text{Library}}$ (Y)

以上7個の要素を基礎にして$\widetilde{L}(y)$で記述されたprogram $CS(X, Y)$が構成される.

又$CS(X,Y)$が$CGS(X,Y)$の入力Dataとして使用されるとき，上記(1)〜(7)の条件に関係するdataは全てvariableとして記述されることが望ましい．（§7参照）

§4.2　$CS(X,Y)$の許容するsource program

Automatic coding language $L(X,Y)$で記述された，ある計算の手順を示すflow−chartを$s=(X,S)$とすればsは一つのgraphである．但し$X=b_1\cup b_2\cup\cdots\cup b_n$，$S=s_1\cup s_2\cup\cdots\cup s_n$とかかれ，$b_i$はbox，$s_i$はbox b_iと　任意のbox b_jとの関係を示すswitchと呼ばれ，X,Sをsを構成するbox及びswitchの集合という．

box b_iにはbox coordinate（label）をつける．各box b_iは次の五のcomponentから成立つ．

(1) Declaration component d_i

box b_i内のIdentifierの性質を規定する．

(2) Operation component Op_i

box b_i内のcomputationを表わす部分

(3) Machine component mI_i

box b_i内のmachine codeでかかれた部分

(4) Substitution component SB_i

box b_i内のsubroutine callingを表わす部分

(5) Switch component SW_i

box b_iとbox b_j（$1\leqq j\leqq n$）の関係を示す，boolean functionでかかれた部分

次に各Componentは$L(X,Y)$のprogramming wordが$L(X,Y)$の文法に従つて配列されたrelationであるsubroutineのsequentialな集合である．

box b_iのcomponentを構成するsubroutineをs_{ij}とすれば，$b_i=(s_{i,1},s_{i,2},\cdots,s_{i,n(i)})$とかける．このとき$\{s_{ij}\}_{j,i}=p$を$s$のsource program (s,p)という．

たゞしpは，もしbox b_iのDeclaration componentがbox b_jのprogramming wordをdeclareする関係にあるときは，$b_i>b_j$（b_iはb_jより先に配列される）及びb_i内においてdecralation subroutine s_{ij}が$s_{i,k}$をdeclareするとき，$s_{ij}>s_{i,k}$であるという条件（compilation condition）を充している．

この条件を充す限りにおいて，box b_iの順序は任意である．

b_iにおける隣り合うs_{ij}のgroupが同一のcomponentに属するとき，component Mark（DC,OP,MI,SB及びSW）をそのgroupの先頭につける．

Program declaration　　source program pの前にpの性質を規定するprogram declarationを，最後にProgram end Mark %をつける．

PRG NAME INITIALADDRESS 及び
REG NAME (Formal parameter list)
TSB NAME (Formal parameter list)

の三種類あつて，PRGはsを直ちに実行可能なProgramに作成して，INITIAL ADRESSから格納できる様にし，REGはsをFormal parameter listにかゝれたformal parameterをもつLibrary (X)に登録されLibrary ($\tilde{Y}$)に格納されたsub-routineに，又TSBはLibraryに登録されない一時的なsubroutineとして作成することを示す．尚NAMEはsに対する記号である．又source program pはsのmainpartと呼ばれる．

§4.3 $CS(i,j)$の構成

Compiler CSはsource programを変換して，*P.M.*の要求するObject programを作成する目的を実行するために，次の四つの部分に大別される．

INTRODUCTION
ANALYSER
ALLOCATION 及び
CONTROL DIVISION

(1) INTRODUCTION

$s=\{s_{ij}\}$を構成するProgramming woadは$L(i)$の文法に合う限り表現は自由である．但し，$L(i)$に固有な形をもつOperator，delimiterは除く．Programming Wordを*C.S.*の規定する標準コードに変換し，分類してCompilerの標準集合にmappingすることによつて，CSは全てのProgramのwordを一定のRuleに従つて処理できる．

Compiler standard set

(i) Process Mark, Component Markのset
(ii) type declarationのためのidentifier
(iii) 文字，数字のset
(iv) $L(i)$に含まれるOperatorのset
(v) $L(i)$に含まれるSubroutineのset
(vi) Variableのset
(vii) Constantのset
(viii) Program Nameのset
(ix) Coordinateのset

以上のsetの中で，(i)(ii)(iii)(iv)は$L(i,j)$毎に一定である．即ちMachine $M(j)$

によつて多少の変更を受ける．(v)(vi)(vii)(viii) はCompilerが作用するProgram毎に登録されて作成される．特に (vi)～(viii) は適当な指示がなければ一回毎にCancel される．

(2) Declaration Analyser (DC)

s_{ij} に使用されるIdentifierの typeについてのinformationを作成し，Identifierに附加する．

一般にvariableは①classification，②type，③分類されたset における登録番号，④分類されたsetにおける相対的なinitial address，⑤word length，⑤arrayの場合はStructure格納番地から構成される．従つてsource program におけるVariable identifierはCompiler setにおいては一つのベクトルでその要素が上記の如く五つあることを示す．

相対Addressの決定には，single precisionは1，double precision，Complex numberは2 wordによつて計算される．登録番号のcounterには s, f, c (Symbolic variable, formal variable and constant) initial addressのcounterには s_1 (symbolic variable) s_2 (array variable) f (formal variable) c_1 (constant) がある．

Declaration part又はvariable，constantの登録とSearchを行う．

(3) ANALYSER

Operation Analyser	OP
Machine Instruction Analyser	MI
Substitution Analyser	SB
Switching Analyser	SW

の四個のPartに分れる．

INTRO及びDeclaration Partを通した s，すなわちCompiler の標準コードに直された s を $\overline{s}$ とかく $\overline{s}=\{\overline{s}_{ij}\}$ 但し混乱のない場合には $\overline{s}$ の代りに s を使う．

$$\overline{s}=\{\overline{s}_{ij}\}=\{\mathrm{INT\ and\ DC}(s_{ij})\}=\mathrm{INT,\ DC}(s)$$

変換の手順

$\overline{s}=\{\overline{s}_{ij}\}$ の $\overline{s}_{ij}$ が属するComponentに対応するAnalyser (Al) によつて，次の段階を経て，$\{\overline{s}_{ij}\}$ は変換される．

① $\overline{s}=\{\overline{s}_{ij}\}$ の $\overline{s}_{ij}$ を変換可能なrelation $\overline{s}_{ijk}$ にreduceする．

② reduceされた $\overline{s}_{ijk}$ に必要なinformationを作成する．

③ reducedされた $\overline{s}_{ijk}$ に対応するMachine $M(j)$ のlanguage $\widetilde{L}(j)$ でかかれるObject relation $\widetilde{s}_{ijk}$ を作る．

このとき $\widetilde{s}_{ijk}$ は未定係数として

(i) Variable, ConstantのIdentifierのparameter sy, cs, sa, wm, f

(ii) Used subroutineのparameter (u, s, p)をsubroutionのsetの中へ記入する．このときstructual subroutine (complex, index, double pecision etc) もautomaticallyに記入される．

④ $\tilde{s}_{ijk}$にAllocationの未定係数として

$$\tilde{a}_{ijk}=\mathrm{ALC}(\mathrm{PNOP}(\tilde{s}_{ijk})-\mathrm{BNW}(\tilde{b}_i))^{[i]}\ ;\ \mathrm{op}\ z$$

をつけて，Tapeに格納又はPunch outする．

⑤ s=REGのとき，sのformal variableを使用したinstructionの$\tilde{s}_{ijk}$内の相対順位をFT(Formal variable address Table)に格納する．used subroutineの情報はUSTに記録する．

以上によつて，source program $s=\{s_{ij}\}$はsequence $\{\tilde{s}_{ijk}\backslash\tilde{a}_{ijk}\}$に変換される．このとき，

$$\tilde{s}_{ij}=\sum_k(\tilde{s}_{ijk}\backslash\tilde{a}_{ijk})$$

$$\tilde{b}_i=\sum_j\tilde{s}_{ij}$$

$$OP=\sum_i\tilde{b}_i$$

(4) Allocation Part

(1)〜(3)で作成されたsequence $\{\tilde{s}_{ijk}\backslash\tilde{a}_{ijk}\}$における$\tilde{a}_{ijk}$の決定に必要な計算を行い，$\tilde{a}_{ijk}$を確立した係数に変えるための情報(AL, I)を作成する．

尚Allocation Partは

REGISTRATER, ALLOCATER, REDUCTER　の三つのPartよりなる．

(1) REGISTRATER

s=REGのとき$\tilde{s}$をsubroutine化するために$\widetilde{OP}$に次の機能をもつ$\widetilde{RG}$をつける．

(i) $\tilde{s}$の$\widetilde{OP}$に使用されたformal variableとそれに与えられるActual variableとのconnectionのための

Actual address reception $\widetilde{RG}$ 1 (pr)

Formal address modification $\widetilde{RG}$ 2 (se)

これらは主として，FTの情報を使用して作る．

(ii) Index Register，その他のRegister又はVariableを必要に応じて待避し，復活するための

$\widetilde{RG}$ 3 (rs)　$\widetilde{RG}$ 4 (rr)　(Register set及びRegister reset)

(iii) Main Partとのlink operation $\widetilde{RG}$ 5 (lo)

s=PRGならば$\widetilde{RG}$は附加しない．

(2) ALLOCATER

① u, s, p以外のparameterをpr, se, rs, op, rr, lo, sba, cs, sy, wm, sa

の順序でpr=0として相対的に確定する．この際opのwords数はNLOT=NOP−1である．但しNOPはOperation part変換終了時の値である．

② $\tilde{b}_i$ のopに対する相対address $[i]$ の確定

$[1]=0$, $[i]=[i-1]+(\tilde{b}_{i-1}$ のwords数$)$ このとき

$$\tilde{a}_{ijk}=\mathrm{op}+[i]+(\mathrm{PNOP}(\tilde{s}_{ijk})-\mathrm{BNW}(\tilde{b}_i))$$

③ CST(constant table)の内容の打出し又はTapeへの格納

④ $\tilde{b}_i$ のAllocation headingの作成と打出し

ALI i $[i]$ $(1\leqq i\leqq n)$

⑤ usp以外のparameterのALLOCATING Heading

ALI $\theta_{j-1}(+)$ parameter (j) の打出し

ただし $\theta_0=0$, $11\geqq j\geqq 1$, θ_j はparameter (j) で表現されるsetの語数．

⑥ s=REGならば

ALI 0000° ; parameter (s)

及びLibraryへの登録の情報として

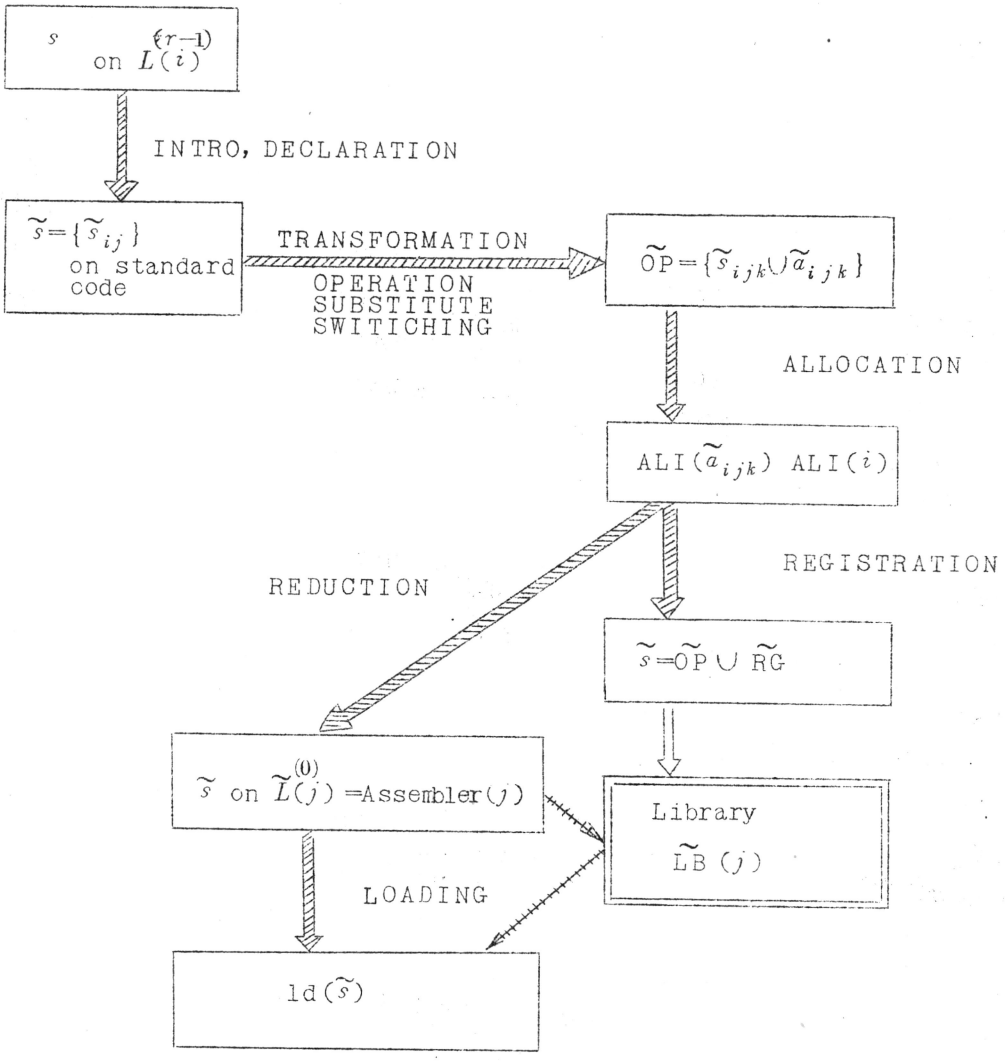
(i) 第1代のused subroutineの名称のset

(ii) program s のNAME，入力，出力変数の数，$\tilde{s}$ の語数，及びそのparameterの打出し．

(3) Reductor

s=PRGのとき，$\tilde{s}$ を実行可能な段階まで移し，そこで全used subroutines を重複せず配列し，各subroutineのparameterにInitial $(\tilde{s})$ を基準とする番地を与え，かつLibraryからの転送に必要な情報を打出す．

Compiler の働き



§5. C.G.S.の方式

C.G.S.の方式には次の三つを考えることができる.

(1) Self—Growing方式

(2) 変換方式

(3) 混合方式

§5.1 Self Growing方式

Compiler $CS(i,j)$ self expressible の性質を充すときは，その機能の拡張を $L(i)$ で表現し，$CS(i,j)$ 自身で行うことができるから一つの self—growingの性質をもつ．この性質をCSの設計に利用すれば，目標である$CS(i,j)$の全部を $\tilde{L}(j)$ でかかなくても，Basicな部分（self growingに必要な部分）を$\tilde{L}(j)$でかき，これをbaseとして拡張して行く方式ができる.

§5.2 Transformation方式

Machine $M(j)$に対して一つの$CS(i,j)$があるとき，ことなるMachine $M(k)$に対して$CS(i,j)$と同じ機能をもつ$CS(i,k)$を次のstepによつて作成する.

(i) $CS(i,j,j) \rightarrow CS(i,j,k)$

(ii) $CS(i,j,k) \rightarrow CS(i,k,k)$

(i)は次のように行われる.

① 出力コード表の変更

$CS(i,j,j)$の出力の部分に関係するコード表すなわち，$\tilde{L}(j)$のInstruction group table及びそのGroup No. tableを$M(k)$の$\tilde{L}(k)$によるMIGT(k)及びGNT(k)に変更する.

② 出力部分に関係するAddress変更

①によるMIGTの変更により生ずるCSの各partの出力部分におけるMIGT に関係するInstructionのAddress PartをMIGT(k)に合う様に変更する.

③ 2進，10進による変更

$M(i)$, $M(k)$の数表現のbaseが異なるときは，Variable, Constantの相対address計算のcounter, Allocation のための未定係数の計算及び情報決定のためのcounterを$M(k)$の方式に合う様に変更する．もしこの変更をしないときは，$\tilde{L}(k)$ におけるcomputation timeにおける$\tilde{L}(k)$のloading routineに必要な変換機能を与えればよい.

(ii)は次のように行われる.

① $CS(i,j,k)$はsource languageの$L(i,j)$であるから，Compiler $CS(i,j)$

を$L(i,j)$で表現したProgramを作成し，$P(CS(i,j)\mid L(i))$とし，これを$CS(i,j,k)$のsource programとする．object programは$CS(i,k,k)$である．

（註）　$P(CS(i,j)\mid L(i))$については　§7　を参照．

§5.3　混合方式

(1)と(2)を組合せたものである．$CS(i,j)$のBasic Partを$CS_B(i,j)$とかくとき次の段階によつて$CS(j,k,k)$を作成する．

(i)　$CS_B(i,j,j) \rightarrow CS_B(i,j,k)$

(ii)　$CS_B(i,j,k) \rightarrow CS_B(i,k,k)$

(iii)　$CS_B(i,k,k) \Rightarrow CS_B(i,k,k)$

§6. C.G.S.の文法

C.G.S.は

(1) C.G.L.（C.G.S.のsource language）

(2) C.G.S.本体のProgram

(3) Input Data　(i) $CS(x,y,z)$の$L(u)$による一般的な表現

(ii) (x,y,z)のactual parameter (A,B,C)による本体の要求するactual parameterのset　$a(A,B,C)$

より成る．

§6.1　C.G.L.

(1) C.G.L.のletter

Alphabet，数字，記号

(2) Variable

letterのstring　但し5文字以下で数字で始まらない．

(3) Constant

integer typeの数値で12ケタ以下の正の数である．

(4) Coodinate

？で始まる5文字以内のletterで作られる．<?>が他のwordとの区別をする．

(5) Process Mark

PRG, REG, TSB, %

(6) DC(declaration partのMark), OP(Operation partのMark), SW(Switching partのMark), SB(Substitution partのMark), K(Component End Mark)

(7) Operator

(A) Arithmetic operator

+, −, ×

(B) Relational operator

>(gtr), ≧(geq),

=, ≠(not equal)

(C) Contraction operator

¥(contraction)

(D) Boolean operator

"(or), $(and), not

(E) Special operator

// word指定operator (word lengthが2以上であることが，定義された variableに対して用いる．(例) A//2 はAの第2番目の wordを指定する．)

lsf, rsf lsfは left shift, rsfは right shiftを示す．

ext, lxt, rxt

extはregisterの内容の extraction

lxtは extractionされた結果をregisterのMSDまで移動する．

rxtは extractionされた結果をregisterのLSDまで移動する．

ind index指定operator Array variableの subscriptによるelement指示の他に，直接indexを指定してその変更等を行うための命令である．

example E ¥ ind(2)は expression Eの current valueをindex 2に代入する．

@ index modification operator

example A@2+B@1¥x , index(2) ;

はArray variable A, Bの夫々 index 2, 1で指定されるelementの和をx及びindex(2)に入れるexpressionである．

§6.2 Declaration (*DC*)

Declaration mark

area	variable arrayのelement数及びelementのwords数指定
array	array variableの構造指定
wl	0−dim variableのlength指定

Variableの性質

C.G.S.におけるVariableは全てinteger typeとする．又1～9までの倍長を自由に指定できる．よつて1 elementがn(1～9)lengthをもつという．

Area　Variableがarrayの構造をもつときはあらかじめVariableのとり得るelementの上限，及びそのword lengthを与え，variableのarrayのinitial addressを定めるMarkである．

尚，Area declarationのみで，Array structureを declare しなくてもよい．

Array　C.G.S.のArrayは二次元までとする．

example

IC　Area　$A, B(100/\!/2)$;
　　Array　$A(10, 5), B(10, 10)$;
　　wl　　$X, Y, Z(2)$;

§6.3　Operation Part　OP

Expression

Variable，Constant，Operator及び( )で作られるoperationを示す式をexpressionという．expressionにはArithmetic expressionとBoolean expressionとがある．

formula

E　¥$X_1, X_2, \cdots X_n$;

の形をformulaという．但しEはArithmetic又はBoolean expression，$X_1, X_2, \cdots X_n$はVariable又はVariable $/\!/n$である．

example

OP　　$S+T$ ¥C;
　　$X/\!/3+Y/\!/2$ ¥$Z/\!/3$;

§6.4　Switching function SW

coordinate α_iをもつblock b_iとcoordinate α_jをもつblock b_jとの関係を示すSwitching function SW_iは次の表現ができる．

SW　　goto α_2 ;

SW　　(B_1)¥$(C_1), (B_2)$¥$(C_2), \cdots,$ NO ¥(C_n); K

但しC_iはOP，SB，SWのcomponentのsequenceで必ず最後はSW Componentをもつ．

ただし，B_i はBoolean expressionであつてB_i がtrueならば(C_i)の内容を行い，B_i がfalseならばB_{i+1} を調べる．B_i ($1 \leq i \leq n-1$) がfalseならば(C_n)の内容を行う．

§6.5 Substitution SB

C.G.S.はその目的がCompilerの作成であるから，Utility routine及びTSB (Temporary subroutine declaration)によつて定義されたsubroutineのみSBでCallingできることにする．(§9. program例参照)

§6.6 Utility subroutine

C.G.L. languageの中にあらかじめ定義されているsubroutineであつて，次の種類がある．

entrance Program又は Temporary subroutineの先頭につける．同様に

exit はその最後につける．

read print はread(1)，print (3)の如く，数値で指定された種類のinstruc-tionを打出す．

empty (n) nは数値で，empty (5) とかけば無効命令(例えば000000000000等)が5 word分打出される．これは後に正式の命令を入れたり，特別な機械語をかきたい時に使用する．

§7. Input Data作成の注意

$CS(x, y, y) = CS(x, y)$ が$L(u)$で記述されるとき $CGS\ (U, j, x, y, z)$ の入力Dataとなる．よつてC.G.S.へのactual parameterが変るたびに，この $P(CS(x, y, y) \mid L(u))$ が一々変更することは不経済であるから，parameterの変更により変化する部分は，初めにFormal variableで表現すればよい．(但しconstantを使用しなければ，variableでもよい．)

特に actual parameter (A, B, B) である場合について考察する．

§7.1 Machine $M(B)$ に関係する部分

(1) $CS(x, y, y)$ の使用するTableの大きさ

Compilerはその baseとするMachine $M(B)$ によつて当然memoryの容量は異るから，Tableの深さは予め決定できない．$CS(A, B)$ がOutputされてから$M(B)$ のmemory等を考慮して決定すればよい．従つてTableの深さの限界はformal variable (又は単にvariable)とする．従つて$CS(A, B)$ の許容するVariable, Constant,

formal variable, coordinate (Label), Array variableの個数等は全てvariableで与えられる.

(2) $M(B)$のwordの構成に関係するData

CSの各tableの内容の形式, 大きさ等は$M(B)$のwordの種類, 大きさによつて変更される. 特にTable内容の一部抽出等のためのShift, extraction 等に関係するDataはvariableであることが必要である.

(3) Programming wordの表現形式及び標準コードの示すData

$M(k)$の入出力装置及びwordの構造等により第1類のProgramming word (Declaration word, Component Mark, Process Mark, Operator, utility−subroutineの記号, separator記号 etc.) の表現の形式, コードは可成り影響される. 即ち$CS(A,B)$の$L(A)$は$M(B)$によつて影響をうける.

(4) その他のProgramming wordの構造を規定する定数

boxのcoordinate (label), Variable, Constant (数値)の表現, Programの Name, $L^r(r\geqq 1)$に属するsubroutineの名称, parameterの形式及び文字の判定限度数等を規定するDataは全てvariableで記述する.

(5) Allocating Informationに使用するstructual parameterの表現

Structual parameterの表現は$M(k)$, i.e. $\widetilde{L}(k)$によつて一般に異なる.

(6) VariableのArray化

以上の(1)〜(5)までの注意による(formalな)variableは可成く系統別にまとめ, 番号づけた表現である事が望ましい.

理由は, variableに対応するActual variableのData又はAddressを$CS(X,Y,Y)$に与える時, その作成が容易であり, 同時に誤りの発見を容易にし, $CS(X,Y,Y)$の表現の見通しがよくなる.

§8. $CGS(x,y,z|U,j)$本体の設計について

特にOperation Part OPについての注意をのべる. Operation Part は2部に分れOP−IとOP−IIと呼ぶ. OP−Iは, operation partに属するformulaを分解してtransformableなexpressionを作成し, OP−IIがこれに対応するmachine instructionのsetを打出す.

OP−IIはこのとき, MIGT上に指定されたInstructionのsetを取り出してその性質に応じた処理を行つて, Instructionを完成する.

この際OP−IIが, 対象とするMachine $M(B)$によつて変更をうけないようにMIGTを作成する必要がある.

§8.1 OP−Ⅱの方式

OP−Iから送られたtransformable expressionを$E(k_0)$とする．一般に$E(k_0)$は

Left operand LS, Right operand RS,

operator Op(k_0)よりなる．又このexpressionのoperandのAC(register又はaccumulator)使用のinformation ACBが送られる．

$E(k_0)$に対応するmachine expression $\widetilde{E}(k_0)$はACB=00(0), 01(1) 又は10(2)の値によつて変化をうける(ここで( )内の数値は10進法表現を示す)．従つてOP−Ⅱが全ての$E(k_0)$を統一的にかつ一定の方法で$\widetilde{E}(k_0)$に変換しうる様に次のような配列を行う．

$\widetilde{E}(k_0, \alpha)$のinstructionがMIGT上のいくつかの連続したgroupで作られるとき，即ち

$(s_1(k_0,\alpha)\sim t_1(k_0,\alpha))$, $(s_2(k_0,\alpha)\sim t_2(k_0,\alpha))$, ····,

$(s_m(k_0,\alpha)\sim t_m(k_0,\alpha))$　　(αはACBの値)

で指定されるとき，$\widetilde{E}(k_0,\alpha)$に対応してoperation store Number table(OSNT)のOSN($3k_0+\alpha$)の上のosnの数値の組を配列する．

example　　<+>, <−>, <×>の場合

MIGT

1	12		1
2	13		2
3	13		1
4	13		2
5	12		1
6	17		2
7	22	3d0021	0
8	12		1
9	17	3d0022	0
10	17		2
11	42		1
12	33		2
13	15	3d0026	0

OSNT

+ 00	1	2
01	3	3
10	4	4
− 00	5	6
01	7	9
10	10	10
× 00	11	13
01	14	16
10	17	19

§8.2 Instructionの分類と構成

MIGTに配列されるInstructionはその性質によつて次の四種類に分類される．

(1) Address PartにOperandを要求しないもの(分類コード0)

example　12 00 3d 0021 00

この命令は12＊000a；としてpunchoutされる．以下の構成は同じである．

Instruction	Punchout form
5900e90002 00	59 ＋0002；
33003d0022 00	33 ＊000b；
120003000100	12 ＊0001(cs)；

(2) Address PartにOperandを要求するもの

Left operandを要求すれば分類コード　1

Right operandを要求すれば分類コード　2

この場合はAddress Partはall zeroである．

example

12 0000000001,

この命令は，$A(i)$−table（OP−Iの Analysis table）上に表現されたLeft operandの状態によつて，次の如く分類されてpunch outされる．

Left operand のAddress Part	打出し form
2000200	12 ＊0002 (sy)；
2003010	12 ＊0030 (sy), 1 ；
3000500	12 ＊0005 (cs)；
5040110	12 ＊0401 (sa), 1 ；
6001000	12 ＊0010 (fm)；
7000300	12 ＊0003 (wm)；

Right operandの場合も同様である．

□印はindex指定である．

(3) utility routineのparameterを要求するもの

example

	output
42 01 ext 03	42 ＋0001,＊0(ext)；

(4) Address Partのみ打出すもの

example	output
000003005 04	＊0005(cs)；

§9. $CS(A,B)$ 作成の手順

(1) Actual parameter A, Bを定める.

(2) MIGT(B), OSNT(B)を作る.

(3) $A \neq X$ならば, $L(X)$, $L(A)$が異なることによるactual parameter $l(A)$を作る.(注1)

$A=X$ならばこの手続は不要.

(4) $CS(A,B)$に必要な機能を定めて, Input Data $P(CS(X,Y,Z),U)$から, もし不要の部分があれば取り除く.

(5) $CGS(X,Y,Z)$ +MIGT(B)+OSNT(B)

をsetして, $P+l(A)$を入力すれば, その結果$\tilde{L}(B)$でかかれた$\tilde{P}(CS(X,Y,Z),B)$が出力となる.

これと同時に$CS(A,B)$に必要なactual parameter $a(A,B)$をきめる資料となるDataが作られる. これを検討して, $a(A,B)$を実際に作成する.

(6) $\tilde{P}(CS(X,Y,Z),B)+a(A,B)$が求める$CS(A,B)=CS(A,B,B)$である.

(注1) $l(A)$は主としてPのControl Divisionを変更するものである.

尚, $\tilde{P}(CS(X,Y,Z),B)$はC.G.S.のbaseであるMachine $M(j)$の出力機械によつて打出されたものであるから, $\tilde{L}(B)$で記述されていても, 例えば, その紙テープ等はそのままMachine $M(B)$にloadingできない場合がある.

この様な場合は, 作成されたProgram $CS(A,B)$をそのまま, $M(B)$のtypewriterでPunchしなおせばよい.

もし$M(j)$と$M(B)$のCodeが異るだけの時はCode変換のルーチンを作り, $\tilde{P}(CS(X,Y,Z),B)$をこれに通してから使用する.

尚, 次にあげたものは, 実験的に作成中のCompilerのOperation Partの一部(OP−I)をC.G.L.でかいたものである.

```
        PRG     OP-1      ( 200 )

?C00    SB  entrance ; K
        DC  area     A ( 200 // 1 ) , CP ( 4 // 1 ) , ID ( 3 // 1 ) ,
                       DATA ( 10 // 1 ) ;
            array    A ( 200 ) ; K

            TSB ATSET ( )

?ATS1   SB  entrance ; K
        OP  0 ¥ PDF , PD , CR , CL , RO , LO , UFL , ACB , ACF , PREAC ;
            100 ¥ i ;
            400 ¥ wm , pwm ; K
        SB  exit ; K

%

?C0     OP  ind ( 1 ) ¥ ID // 1 ; ind ( 2 ) ¥ ID // 2 ;
            ind ( 3 ) ¥ ID // 3 ; K

?C01    SB  ATSET ; K

?C1     SB  INTRO ; K

?C11    OP  RA ¥ CP // 3 ; RB ¥ CP // 4 ; K
        SW  ( RA = SYMBOL ) ¥ ?C//1 ;
            ( RA = const  ) ¥ ?CONS ;
                        NOT ¥ ?C3 ; K

?C3     OP  RB ¥ RA ; K
        SW  ( RA = DATA // 6 ) ¥ ?CPD1 ,
            ( RA = DATA // 7 ) ¥ ?CPD2 ,
            ( RA = DATA // 8 ) ¥ ?CD;  ,
            ( RA = DATA // 9 ) ¥ ?CD,  ,
                           NOT ¥ ?C4      ; K

?C4     OP  CP // 3 ¥ RA ; K
        SW  ( RA = 0 ) ¥ ?C8 ,
                  NOT ¥ ?CE01   ; K

?C//1   SW  ( RB = 0 ) ¥ ?C//2 ,
                  NOT ¥ ( SB  STS ; K

                              SW  goto ?C73 ; K ) ; K

?C//2   SB  INTRO ; K
        SW  ( RA = const ) ¥ ( OP RB ¥ RA ;
                                  RA lsf ( 2 ) - 100 + A ( 1 - 1 ) ¥
                                                       A ( 1 - 1 ) ; K
                               SW goto ?C1 ; K ) ,
                                  NOT ¥ ? CE02 ; K

?CONS   SB  CTS .; K
        SW  goto ?C73 ; K

?C73    OP  RA ¥ A ( 1 ) ; 1 ¥ PDF ; K
        SW  goto ?CP10 ; K

?CP10   OP  1 + 1 ¥ 1 ; K
        SW  goto ?C1 ; K

?CPDF   SW  ( PDF = 0 ) ¥ ( OP  PD + 10 ¥ PD ; K
                             SW got  ?C1 ; K ) ,
                    NOT ¥ ( OP  PDS ¥ ASN ; K
                              SB  index ; K
                              SW  goto ?C1 ; K ) ; K

?CD)    OP  PD - 1000 ¥ PD ; K
        SW  goto ?C1 ; K

?CD;    OP  1 ¥ CR ; 0 ¥ RO ; K
        SW  ( LO = 0 ) ¥ ( OP  CAW // 2 ¥ CSWX ; K
                             SW  goto ?C20 ; K ) ,
                  NOT ¥ ?CE03 ; K

?CD,    OP  OPSTR ¥ RA ; K
        SW  goto ?CP8 ; K

?CD8    OP  CP // 4 ¥ RA ; K
        SW  ( RB = OPERATOR ) ¥ (OP  0 ¥ PDF ; RA + PD ¥ RA ; K
                                   SW  goto ?CP8 ; K ) ,
                         NOT ¥ CE04 ; K

?CP8    OP RA ¥ A ( 1 ) ; RA ext ADR ¥ RO ; 1 ¥ CR ; K
        SW  ( LO geq RO ) ¥ ( OP CSW // 1 ¥ CSWX ; K
                               SW goto ?C20 ; K ) ,
                      NOT ¥ goto ?CP9 ; K
```

```
?CP9    OP  CR ¥ CL ; RO ¥ LO ; K
        SW goto ?CP10 ; K

?C20    OP  KO - 100 ¥ j ; K

?C21    SW  ( j = 0 ) ¥ ?CU ,
            ( A ( j ) ¥ ( OP j - 1 ¥ j ; K
                          SW goto ?C21 ; K ) ,
            ( A ( j ) ext SIGN = 0 ) ¥ ?CU ,
                                NOT ¥ ?C22 ; K
?C22    OP   0 ¥ UFL ; j ¥ LS ; K
        SW  ( A ( LS ) ext SIGN = ACSIGN ¥ ( OP 2 ¥ ACB ;
                                             ACF - 1 ¥ ACF ; K ) ,
                                    NOT ¥ ?CU ; K

?CU     OP  KO + 1 ¥ j ; K

?CU1    SW  ( j gtr 1 ) ¥ ?CE06 ,
            ( A ( j ) = 0 ) ¥ ( OP j + 1 ¥ j ; K
                                SW goto ?CU1 ; K ) ,
            ( A ( j ) ext SIGN neq 0 ) ¥ ?CU3 ,
                                NOT ¥ ?CE07 ; K

?CU3    OP  j ¥ RS ; K
        SW  ( A ( RS ) ext SIGN = ACSIGN ) ¥ ?U1 ,
            NOT ¥ ( OP ACB + 01 ¥ ACB ; ACF - 1 ¥ ACF ; K
                    SW goto ?U1 ; K ) ; K

?U1     SB  OP-2 ; K
        OP  0 ¥ ACB ; ACF + 1 ¥ ACF ; K
        SW  ( UFL = 1 ) ¥ ( OP  ko ¥ ind ( 2 ) ; K
                            SW  goto ?U2 ; K ) ,
                    NOT ¥ ( OP  LS ¥ ind ( 2 ) ; K
                            SW  goto ?U2 ; K ) ; K

?U2     OP  ACSIGN ¥ A @ 2 ; ind ( 2 ) ¥ PREAC ;
            RS ¥ ind ( 3 ) ; K


?U3     OP  ind ( 2 ) + 1 ¥ ind ( 2 ) ; 0 ¥ A @ 2 ; K
        SW  ( ind ( 3 ) gtr ind ( 2 ) ¥ ?U3 ,
                NOT ¥ ( OP  LS ¥ ind ( 2 ) ;
                            ind ( 2 ) - 1 ¥ ind ( 2 ) ;
                            0 ¥ ind ( 3 ) ; K ) ; K

?U4     OP  ind ( 2 ) ¥ j ; K

?U5     SW  ( j = 0 ) ¥ ?CSWX ,
            ( A ( j ) = 0 ) ¥ (  OP  j - 1 ¥ j ; K
                                 SW  goto ?CU5 ; K ) ,
            ( A ( j ) ext SIGN = operator ) ¥
                                 ( OP A ( j ) ext ADR ¥ LO ;
                                        j ¥ CL ; K
                                   SW goto ?COD ; K ) ,
                            NOT ¥ ?CE08 ; K

?COD    SW  ( LO geq RO ) ¥ ? CB ,
                        NOT ¥ ?CE ; K

?CSWX   SW  ( UFL = 1 ) ¥ ( OP  ko ¥ LS ; K
                            SW  goto ?CE ; K ) ,
                    NOT ¥ ?CE ; K

?CE     OP  RO ¥ LO ; A ( 1 ) ¥ A ( LS + 1 ) ; LS + 1 ¥ CL ;
            LS + 2 ¥ 1 ; K
        SW  goto ?C1 ; K

?CF1    SB  INTFG ; K
        SW  ( RB = K ) ¥ ?CF2 ,
                    NOT ¥ ?CF3 ; K

?CF3    OP  RA ¥ CP // 3 ; RB ¥ CP // 4 ; K
        SB  ATSET ; K
        SW  goto ?C11 ; K

?CF2    SW  ( wm geq pwm ) ¥ ( OP  wm ¥ pwm ; K
                               SW  goto ?CF5 ; K ) ,
                        NOT ¥ ?CF5 ; K
?CF5    OP  IWM ¥ wm ; ID // 1 ¥ ind ( 1 ) ;
            ID // 2 ¥ ind ( 2 ) ; ID // 3 ¥ ind ( 3 ) ; K
        SB  exit ; K

%
```

且し ?CE01 ～ ?CE0F は error stop 文字。

参 考 文 献

(1) 野口広，藤野喜一，渡部和，若月宏「NUMERIC—Automatic Coding System for the NEAC 1103 (NEAC 1103 自動プログラム体系)」1963年7月25日，電子計算機研究会資料電気通信学会

(2) 藤野喜一「NUMERICにおけるALLOCATIONの問題」昭和38年12月5日，情報処理第4回全国大会予稿

本 PDF ファイルは 1965 年発行の「第 6 回プログラミング―シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトの https://www.ipsj.or.jp/topics/Past_reports.html に下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載して、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間：2020 年 12 月 18 日 〜 2021 年 3 月 19 日
掲載日：2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html