C 2　コンパイラ・コンパイラ手法による問題向き言語Xのコンパイラ作成

森　洋之，中島　秀和，下村　津之（日本電気株式会社）

1. まえがき

この論文では，問題向き言語Xのコンパイラー作成に用いた「BHF分割法」について述べる．

この手法は，Syntax Graphにコントロールされた Parserを組合せて実用に耐えるParserを作ろうとするものである．

この手法の基礎は，先に発表した論文「コンパイラ自動作成の一実験[1]」である．

問題向き言語を問題に合せて開発すれば，記述能率の高い言語ができる．例えば，X言語では，一問題，２０行以内で記述できる．従ってコンパイル速度はさほど問題にはならない．問題向き言語はコンパイラー，コンパイラー手法を活用しやすい分野である．

問題向き言語を開発するときの問題は，

(1) 問題に適合した言語を定義する，

(2) 文法をBNF等で正確に記述する，

(3) コンパイラーの作成等である．

問題に適した言語を定義する過程は，既存の言語などを参考にしながら，人間のセンスに合った定義をするのが良いと思う．この過程では，特に定まった手法はないであろう．次のステップからコンパイラー・コンパイラーで研究されている手法を活用できると有効である．実際にBNFで文法を記述してみると１個のスタートシンボルで始まるBNFの中に全てをおさめるのはむずかしいことであった．

この困難を，BNFを分割することによって解決する「BNF分割法」を考えX言語のコンパイラーの作成に適用した．

2. 問題向き言語Xの定義

問題向き言語によってマン・マシン・コミュニケーションの効能を上げ計算機の活用範囲を広げる目的で，実装記述言語×を開発した．実装は，装置の組立工事に関係し，次の特徴を持つ，

(i)配置決定には必要な部品を最少のスペースに配置するばかりでなく，操作上，保守上等の要求が加わるので配置決定を完全にプログラム化するのは困難である．この配置は装置設計者が行っている．

(ii)実装図は組立図であるため，製図者にとって単調な製図作業のみを要求する仕事である．

(iii)単調であるとは言え，ミスは組立工事を不能にするので細心の注意を要する．

最近の発達した図形出力装置と言語を組合せて正確な実装図を出力し，製図者の能率向上を計るのが実装図記述言語のねらいである．言語×は図１の手順で用いられる．

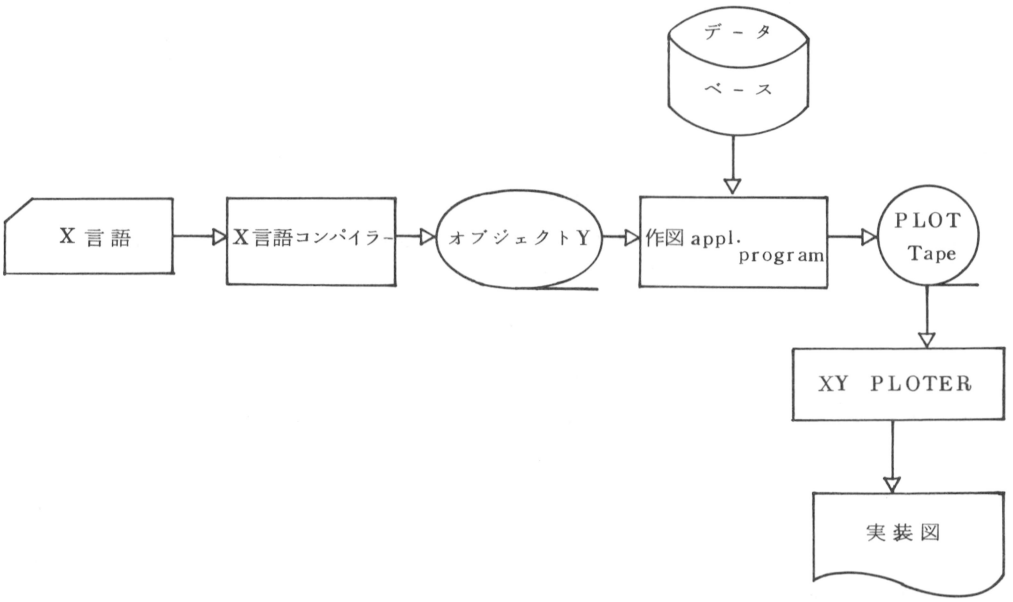

図1　言語Xによる作図手順

一枚の図面は，X言語の一BLOCKで表現される．

＜block＞：：=＜head＞END $

＜head＞：：=＜block - statement＞|＜head＞＜statement＞

＜block - statement＞：：=ID：BLOCK

(＜parameter - list＞)$

BLOCKは，BLOCK-STATEMENTで始まり，END-STATEMENTで終るSTATEMENT群である．IDは，図面につける名称で使用者が自由につけられる．PARAMETER-LISTは一枚の実装図の骨組になる図の種別の指定と，それをモデファイするパラメータである．

例えば

'FILTER-PNL'　：BLOCK(FIL)$

は図2の実装図の骨組を意味する．

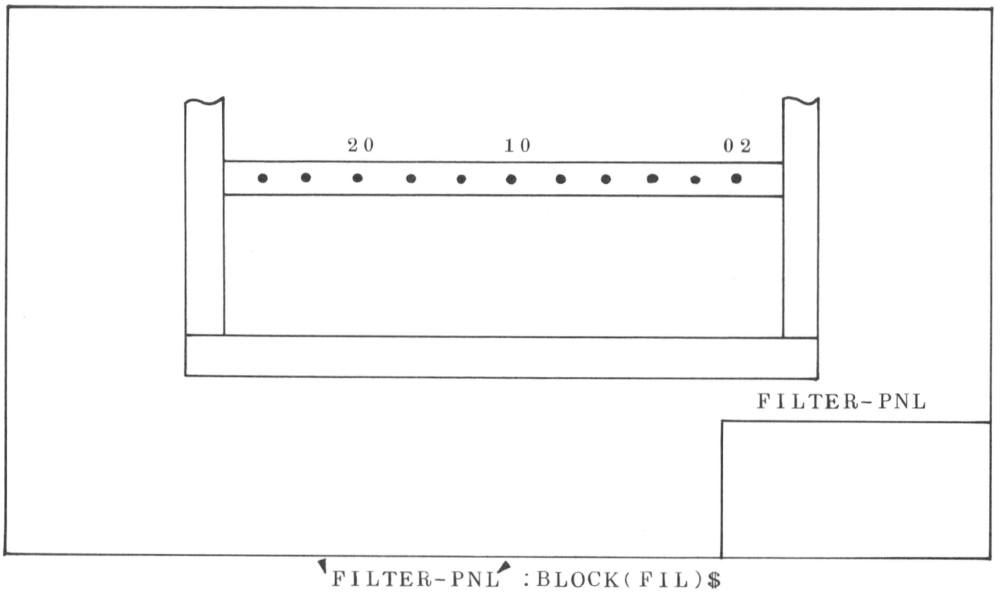

図2　BLOCKステートメントの例

BLOCK-STATEMENTで指定された骨組に取付られる部品とその配置は，取付状態-STATEMENTで記述される．取付状態-STATEMENTで記述された部品に保守，操作上必要な表示文字を所定の位置に表示するのは，文字構成-STATEMENTである．

図3　回路名称の例

図3に示すように，回路図上の部品はR1，C1，等の回路名称を有している．この名称は，回路設計者が自由に決定し付与するものである．これに対し，部品の注文のとき用いるのが品名である．同一品名のものを例えば図4のように続けて並べることが多い．

図4(a)，(b)のような配列をそれぞれ単位配列と言う．図3(a)，(b)の単位配列は

(a)　(R1；R2；R3)%RD10Ω

(b)　(C1；C2)%MD10F

で表現される．；と，はそれぞれ縦方向，横方向の配列オペレータである．

図4の場合の取付状態を

F1(ORG)=(R1；R2；R3；)%R10Ω，(C1，C2)%MD10F$

で表現する．F1は取付状態－STATEMENTを表示するBNF Linkage symbolである．これに関しては後で詳しく述べる．

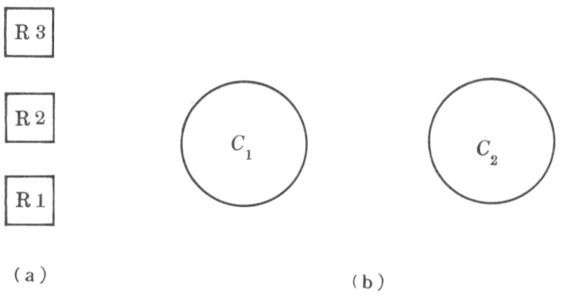

図 4　部品配置例

あるシリーズの装置に関して約４００種の画素ライブラリーとカード３０００枚の画素基準データベースを準備すれば，上述のX言語は，一枚の実装図を２０行以内で記述する能力を有することを確認した．画素の数はほぼ使用部品点種に等しい．

3. 文法のBNFによる記述

実際に，前述の言語の文法をBNFで整理して見ると，収束の遅い工数のかかる仕事となった．BNFの整理を進めるうちに，シンタックス・グラフ上では共通の構造をもつ部分から，出力図形に，２種以上の図を出力する必要があることが問題になった．例えばA1継電器に火花防止ダイオードを付加したものは

(A1(%*SRD*)%*RELY*

で表現される．また100V 15Aの電源スイッチの文字構成は，

(*POWER*(100*V* 15*A*))%*DC*

で表現される．この２つの表現の差をシンタックス・グラフで示すと図５になる．図５の共通部に示す部分は上記２例につき同一である．上記２例の出力図として，それぞれ図６の(a),(b)が要求されている．これからわかるようにシンタックス・グラフ上の同一部分に対して，別々の図面を書くオブジェクトコードYを発生する必要がある．この区別は，Kの下位にある個別部によって判別できるが，下位のノードによって上位のノードのセマンテックの割当を制御する方式は，制御が複雑になるので，X言語のようにかたまってない言語には適してない．

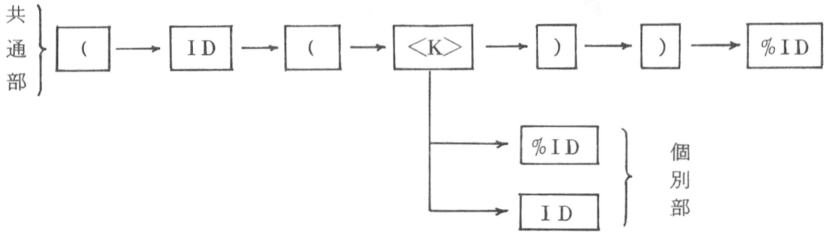

図５　シンタックス・グラフ

(a)　　(b)

図6　出力図例

上述の問題をさけるには，図を書くオブジェクトが変るごとに，BNFも対応して変る言語を作るか，BNF の共通部を2度定義するかの2通りの方法が考えられる．第一の方法は言語としての表現法が繁雑になるので使用者のミスが多くなり好ましくない．第二の方法は，多少の重複が生じるためその分だけBNFは冗長になるが，個々のBNFは短くなりBNFの整理がしやすくなる．この利点に比べれば，重複損は無視できる．そこで，BNFを5セグメントに分割した．図7から図10にBNFから2までの一部を説明の都合上簡単化して示した．

＜block＞：：=＜head＞END$

＜head＞：：=＜block－statement＞|＜head＞
（KEY WORD END以外）
＜statement＞

＜block－statement$^{\nabla}$＞：：=ID：BLOCK（＜parameter－list＞）$

＜statement$^{\nabla}$＞：：=＜assignment－statement＞|＜図面番号－statement＞|
＜control－statement＞

＜assignment－statement＞：：=F1$^{\Rightarrow}$|F2$^{\Rightarrow}$|F3$^{\Rightarrow}$|F4$^{\Rightarrow}$

＜parameter－list＞：：=ID|＜parameterlist＞／or；ID

図7　BNF（0）

B ⊨ ⇐

＜F　1＞：：=（ID）=＜配　列＞

＜配　列＞：：=＜R－配列＞|＜配列＞；or，＜R－配列＞

＜R－配列＞：：=＜単位配列＞|＜単位配列＞↑ or →NUM

＜単位配列＞：：=（＜回路名称配列＞）＜品　名＞

＜回路名称配列＞：：=＜回路名称＞|＜回路名称配列＞；or，＜回路名称＞

＜回路名称＞：：=ID|ID＜品　名＞

＜品　名＞：：=%ID

図8　BNF（1）

B |⇐⇐

＜F　2＞：：＝（ID：ID）＝（＜T－表示文字配列＞）＜品　名＞

＜T－表示文字配列＞：：＝＜T－表示文字＞|＜T－表示文字配列＞；or ，＜T－表示文字＞

＜T－表示文字＞：：＝ID|ID（＜U－表示文字配列＞）|♭

＜U－表示文字配列＞：：＝＜U－表示文字＞|＜U－表示配列＞，＜U－表示文字＞

＜U－表示文字＞：：＝ID|ID（＜L－表示文字配列＞）

＜L－表示文字配列＞：：＝＜L－表示文字＞|＜L－表示文字配列＞，＜L－表示文字＞

＜L－表示文字＞：：＝ID|♭

図９　ＢＮＦ（２）

＜配列－operator＞：：＝；|，

＜repeate－operator＞：：＝→|↑

＜separator＞：：＝／|＝|：

＜bracket＞：：＝（|）|▼

＜品名マーク＞：：＝％

＜blank＞：：＝♭

＜statement－termindtor＞：：＝$

図１０　代表的Terminal Symbol

4. PASERの概要

言語Xのコンパイラーは，Lexical ，Parsing ，オブジェクト発生の３部分からなる．今回，新しい手法を用いたのはParsing部であるので，以降ではその部分について述べる．

ソース言語はLexicalモジュールで．一語づつ，Wに読み出される．Wの語は，Parserによって解読される．解読された語は，解読されるたびに語単位にParserよりParsing Treeバッファ（P.T. BUF）に出力される． 解読が進むにつれ，P.T. BUFには，Wから読み込まれた語とBNFから挿入されたnon－terminalが解読された順に積み上げられる．

１ステートメントの解読が完了した時，P.T. BUFの内容はオブジェクトコード発生部へ送出する．

5. BNFセグメントのLinking命令

分割されたBNFを，一つの言語を解釈するBNFとしてLinkするために，次のLinkage命令をBNFに付加した．

$S\overset{\Rightarrow}{T}$	Terminal Symbol STを認識したらスタートシンボルSTで始まるBNF へ次の語を読込んでからJUMPせよ．このSTをBNF－linkage symbolと呼ぶ．
$<ST\overset{\Leftarrow}{>}$	Non－Terminal Symbol STを認識したら親のBNFへReturnせよ．
$<n^{\nabla}>$	Non－Terminal nを認識したらPT BUFの内容をオブジェクトコードジェネレータに出力せよ．
$\overset{\vdash}{<}n>$	non－terminal nよりSG上下位の構文解析中一つ以上のTerminal を認識した後にSGの

終端に達しないまま解析できないTerminal Symbolにそうぐうし，BNFをnまでもどったら，エラーメッセージをプリントして（$<^{B}\Leftarrow ST>$によるreturnを含む）Stackをnon-terminal nをはき出すまでpop-upし，S.Gの次のSCan開始点をnon-terminal nのSucessorにし，処理を続けよ．

$<^{B}\Leftarrow ST>$ もし，$ST^{\Leftarrow}$ で起動されたBNFでスタートシンボルを，認識できなかったら，エラーメッセージをプリントして親のBNFへReturnせよ．

6. コンパイラーの作成

上記のLinkage-命令を付加したBNFによって，図11に示すように2つのParser（PARS1，PARS2）を持つコンパイラーを構成した．

次に，

```
01      FILTER-PNL  :BLOCK(FIL)$
02              F1(ORG)=(C1)%MD|OF$
03              END$
```

の例について動作を説明する．

02行の第一語であるF1をPARS1が認識したとき，図7のBNFに

$$<\text{Assignment-statement}> ::= F1^{\Rightarrow} \mid F2^{\Rightarrow} \mid F3^{\Rightarrow} \mid F4^{\Rightarrow}$$

があるので，次の語を読込んでStart Symbol F1で始まる，図8のBNFへJUMPするためコントロールはメインコントローラ（CNTL）に渡る．CNTLは次の語“(”をWに読込んだ後，図8のBNFをCallする．“(”から“$”までが認識された後に，CNTLにPARS2からコントロールがReturnしたとき，CNTLは．ENDをWに読込んで再び，図8のBNFをCallする．このとき，図8のBNFは，Start Symbol <F1>を認識する．このとき，コントロールは，WにあるENDを認識せぬままにR PARS1に移る．図7のF1にはsucessorがないので，PARS1はBNFをStart Symbolへともどる．statementまでもどったとき以前にTarminal F1を認識している故にstatementが認識される．statementのsucessorは，statementであるので，PARS1は，statementのBNFを順次SCanするがENDは認識できない．そこで．PARS1は再びStart Symbolへ向ってもどって行く．non-terminal <statement>までもどったとき，今度は認識できたTerminalが一つもないので，

$$<\vdash \text{statement}>$$

によってErrorとならずに，PARS1はnon-terminal <head>までもどる．PARS1はこうして<head>のsucessor ENDを認識する．以下PARS1によって<block>が認識されるまで解読は進められる．

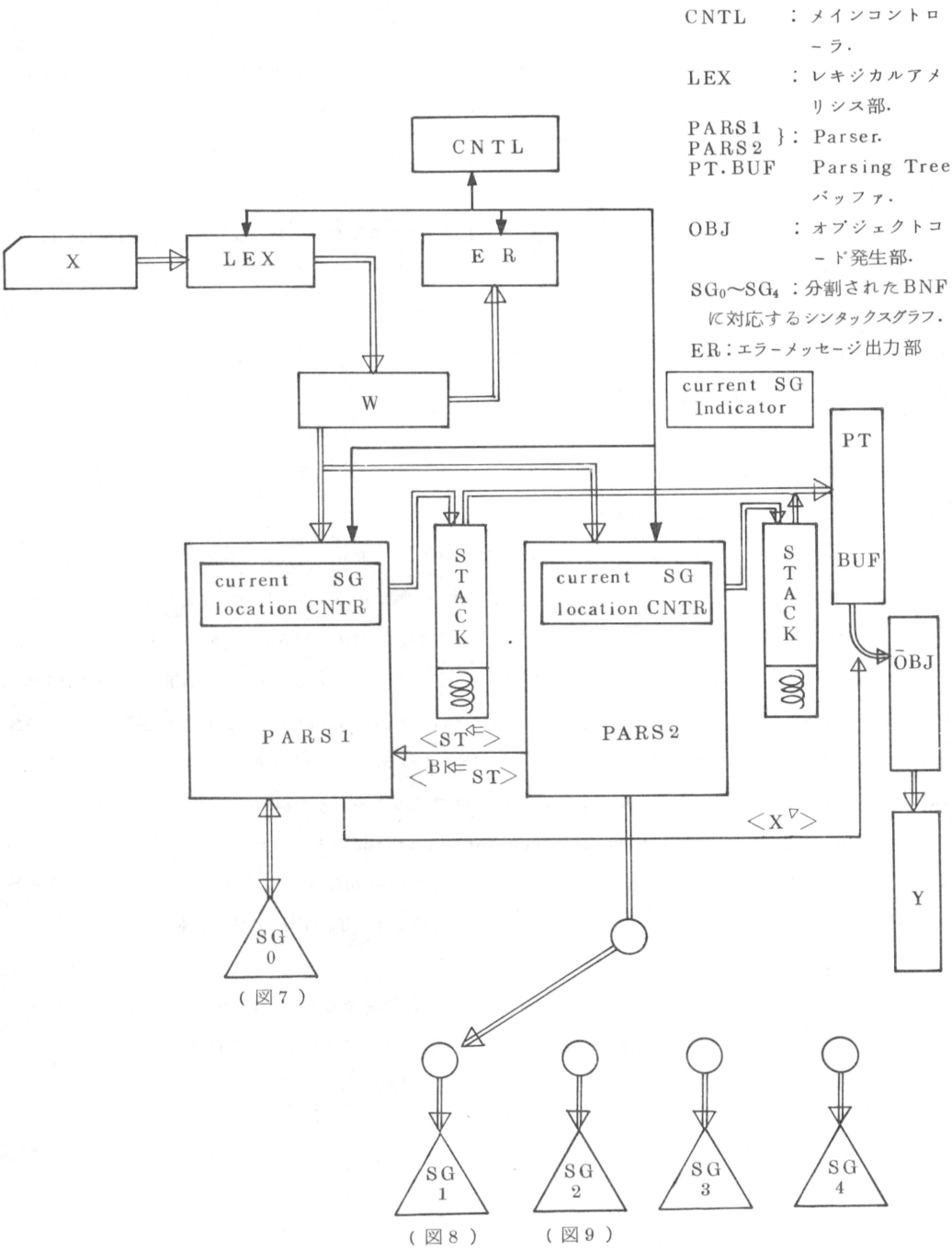

図１１　ＢＮＦ分割法による×言語コンパイラー

7. おわりに

上述の手法は×言語に次の制限を要求する.

(1) $<\vdash_n>$命令は言語×をdeterministic acceptorで認識できる部分 d の存在を必要とする. すなわち nondeterministic acceptorを必要とする部分 nd を d より分けること.

(2) d と nd をLinkするためBNF linkage symbolを導入する.

第一の条件は,×言語の中に少くとも一つ以上の d 部分を必要とするが,そのように言語×を定義するのは容易である. 第2の条件は,使用者にごくわずかの手数を加えるのみで,実用上障害となるほどではない.

本手法は,PL/1のような既存言語には不向きである. しかし上記の制約の中で問題向き言語を定義して用いる場合には有効な手法である. 本手法の利点は,

(1) 言語定義がしやすい.

BNFを分割することによって問題向き言語の定義が容易になる.

(2) 記述能力の高い言語を作れる.

nondeterministic acceptorで認識する nd 部は高い記述能力を言語に与える.

(3) コンパイラーをコンパイラー・コンパイラー手法を用いて作れる.

(4) $<\dashv_n>$命令によってステートメント単位でコンパイルエラーメッセージを出せる.

手書きのコンパイラーと比べれば,エラーメッセージとして不十分ではあるが,実用性は十分ある.

(5) nondeterministic acceptorが参照するBNFを短くできるので, コンパイル速度の確保ができる. BNFが分割されているので,alternateの数も少なくなりSG上の無効な試行トレースを少なくできる.

等である.

本手法の制約の中で定義した問題向き言語×をまだ実用していない. その意味で,本手法の実用性はまだ評価できない面が残る. しかし,上記のように本手法は,問題向き言語用として実用上多くの利点を持っている.

本手法で定義した問題向き言語の主な目的は,今まで文字と数字のら列で入力していたapplication system への入力データをapplicationに適した問題向き言語での入力に置きかえることによって, 入力データを大巾にさく減することである. 入力データ量を大巾に減すには,前述のような問題向きデータベースを必要とする. データベースの準備には相当の時間を要する. しかし,その効果は問題向き言語によって大巾に増巾される.

終りに,本手法の研究を指導して下さったコンピュータサイエンス研究部の藤野喜一研究マネージャー,本手法を具体的に検討しまとめて下さった電子交換事業部員の人々に感謝します.

参考文献

(1) 藤野喜一,下村建之「コンパイラ自動作成の一実験」
第12回プログラミング・シンポジウム(1971) B3

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