展　望

交換プログラム用仕様記述法と高水準言語†

丸山勝己††　吉田靖之††

1. まえがき

交換分野のソフトウェア技術のうち，仕様記述法，高水準プログラミング言語について，現状を紹介し，今後の動向を展望する．

2 では蓄積プログラム制御交換機および交換プログラムの構成と特徴について述べ，3 で，交換プログラムの作成過程と仕様の規定法について考察する．4 および 5 では仕様記述法および高水準プログラミング言語について，現状を紹介する．6 では C.C.I.T.T.（国際電信電話諮問委員会）の標準高水準プログラミング言語（C.C.I.T.T. High Level Programming Language：CHILL）の設計概念を言語仕様を中心に解説し，具体例を示す．

2. 電子交換機

2.1 蓄積プログラム制御交換機

交換機の最も基本的な要素は，通話路の接続・切替えを行う通話スイッチ回路網と，それを制御する制御部である．この制御部としていわば万能形の電子計算機を用いて，プログラムの指示に従って交換動作を行う方式の交換機を蓄積プログラム制御（Stored Program Control：以下 SPC と略す）交換機と呼ぶ．これは，一般には電子交換機とも呼ばれているが，正確には交換機の電子化には通話スイッチ回路網の電子回路化と，制御部の電子回路化とがあり，両方電子化したものを全電子交換機，後者のみ電子化したものを半電子交換機と呼ぶこともある．

SPC 交換機は，概念的には図-1 に示す構成をしている．蓄積プログラム制御方式の利点は，異なる交換動作が必要となっても，ハードウェアの変更なしにプログラムを変更するだけで対処できるので，融通性・可変性に富むこと，ハードウェアの規格統一が可能なこと，プログラムによる障害の自動識別や故障個所診断により保守性の向上を計れること，新サービスやデータ交換・蓄積交換のような新しい高度な機能の導入が容易なことなどにある．

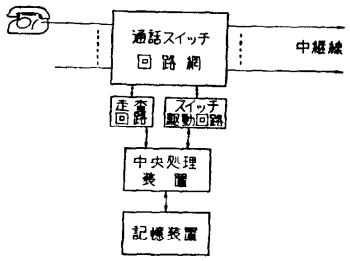


図-1 SPC 交換機の構成

世界で最初に商用化された SPC 交換機は，アメリカの No. 1 ESS であり，1965 年に導入されている．日本においては，1964 年から本格的研究が開始され，1972 年の D10 形交換機の商用化導入以来，すでに約 350 局が稼動している．

2.2 交換プログラムの構成と特徴

交換プログラムシステムは，一般に次のものから構成されている．

①実行管理プログラム：各プログラムの実行のスケジューリングを行う．
②呼処理プログラム：電話呼を検出し，通話路の接続・切換えを行う．
③運転管理プログラム：保守者の指示に従って，交換機の運転を管理する．
④障害処理プログラム：障害発生時に障害装置と予備装置を切換えて，自動的に交換機の再立上げを行う．
⑤診断プログラム：障害個所の診断を行う．

また，通話スイッチ回路網の規模，接続されている回線などは，各交換局ごとに異なるので，このように局ごとに異なりうるものはデータテーブル化して局データ部と呼び，各局で共通に使用されるジェネリックプログラム部とは分離されている．加入者増などにより交換機に通話スイッチ回路網等を増設する場合にも，プログラム部分は変更せずに，局データ部の変更

† Specification Methods and Implimentation Languages for Switching System Programs by Katsumi MARUYAMA and Yasuyuki YOSHIDA (NTT Musashino Electrical Communication Laboratory).

†† 日本電信電話公社武蔵野電気通信研究所

のみで実現される.

D10形交換機の場合，ジェネリックプログラム部は約400Kステップ(アセンブラ)，局データ部は最大約100K語の規模をもっている.

交換プログラムの特色は，次の通りである.

①非常に多くの端末機器（例えば数万個の電話機）を相手にする.

②数千個にのぼる通話を，同時に実時間で処理する.（この高多重度のために，D10形交換機の例では，多数の仮想プロセスがワークエリアを共用して走るような構造を採っている.）

③障害が生じても，波及範囲を最小にしてシステムダウンを避けなければならない.(交換機の許容不稼動時間は20年間に1時間)

④全部の通話で共通リソースを共用する.

⑤同一プログラムが，多数局において長年にわたり使用される.

3. 交換プログラムの製造[1]

電子交換機の適用分野の拡大および具備機能の複雑化に伴い，ソフトウェア開発規模も増大し，生産性・保守性を向上させる技術が非常に重要になってきている．このため各国とも情報処理分野での諸技術の応用，あるいは交換処理の特殊性を生かした新技術の開発に力を入れている.

交換プログラムの一般的な開発手順と仕様の構成例を図-2に示す.

3.1 プログラム作成過程

ライフサイクルの区分方法については，情報処理分野との差異はないと考えられる.

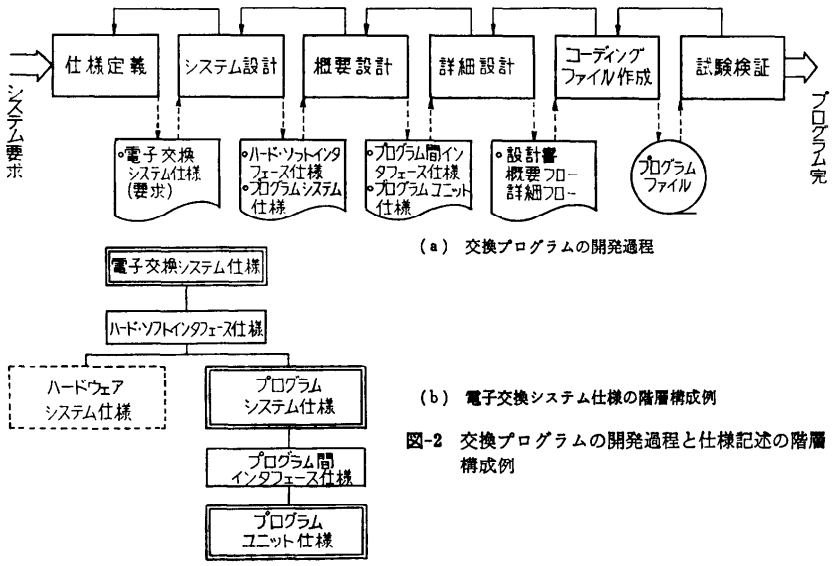
形式言語として実現されているものは，設計段階の各種プログラミング言語がほとんどである．要求仕様を形式言語で定義しようということが積極的に取り組まれ始めている．また，要求仕様定義を機械処理しようとする動きもある．しかし，ISDOS, SREP, SADT[2]等の交換プログラムへの適用を試行している段階であり，多くは自然言語に依っているのが実状である.

交換プログラムを用途別に分類すると，2.2で述べたように実行管理・呼処理・運転管理・障害処理・診断の5種類になるが，これらは主に設計およびドキュメンテーション上の分け方であり，個々のプログラムを磁気テープにファイルする際は，取扱い上の利便からジェネリックプログラム部・局データ部・加入者データ部の3種類に分けている.

ジェネリックプログラム・局データ・加入者データは電話交換局とは別に設けられたソフトウェアセンタで作成・検証後，磁気テープにファイルされ，電話交換局へ供給される．サービス開始後の加入者データの変更などは各局ごとにタイプライタを介して入手で行われるのが普通である.

3.2 ソフトウェア仕様[3),4)]の種類

交換ソフトウェア仕様の詳細を表-1に示す．図-2



(a) 交換プログラムの開発過程

(b) 電子交換システム仕様の階層構成例

図-2 交換プログラムの開発過程と仕様記述の階層構成例

表-1 交換ソフトウェア仕様の例

仕様		定義	記述内容	表現レベル	記述形式
電子交換システム仕様		一般的な機能条件，保守・運用条件等システム全般にわたる外部条件を規定する	・適用範囲　・サービス条件 ・網構成　・保守・運用条件 ・番号構成	方式にインディペンデント	・文章 ・図表 (機能状態遷移図等)
ハード・ソフトインタフェース仕様		ハードウェアとソフトウェアの機能分担をハードウェアの機能をソフトウェア側から規定することにより記述する	・システム構成 ・ハードウェア構成条件 ・ハード・ソフトインタフェース条件	ソフトウェアにインディペンデント	・構成図 ・トランク動作図 ・カルノーマップ
プログラムシステム仕様	プログラム機能仕様	外部条件・ハードウェア条件を満足し，プログラムシステムが実現する機能を規定する	・サービス機能 ・保守機能 ・運用機能	プログラム処理，プログラム構成にインディペンデント	・状態遷移図 呼処理，トランク，装置 ・トリー（規制・再開）
	対外部インタフェース仕様	システムの保守・運用上およびシステム管理上，外部に対して明確にすべき条件を規定する	・マンマシンインタフェース　・局データ作成システムインタフェース ・増設条件　・料金局システムインタフェース ・ファイル入替条件	同上	・図表
	プログラム処理仕様	プログラムシステムに要求される機能を実現するための共通的，基本的な処理方式を規定する	・基本概念　・システム管理処理 ・設計基準　・呼処理 ・共通処理　・運転管理処理 ・プログラム制御処理	プログラム構成にインディペンデント	・図表 ・フローチャート ・動作図
	プログラム構成仕様	プログラム構成条件，プログラムシステム全体の構造をプログラムユニット単位(管理単位)の論理的なつながりとして規定する	・プログラム構成概念　・メモリ属性と割付条件 ・プログラム分割基準　・ファイル仕様 ・システムジェネレーション条件　・プログラム構成 ・品名表	プログラム間インタフェースプログラムユニットにインディペンデント	・プログラム構成図
プログラム間インタフェース仕様		プログラム相互間，プログラム・メモリ間の各種インタフェースを規定する	・プログラム・ブロック間インタフェース ・共通メモリブロック間インタフェース	プログラムユニットにインディペンデント	・マトリックス
プログラムユニット仕様		プログラムユニットの機能，処理方法を規定する	・機能　・外部情報 ・ユニットの位置づけ　・処理法	機械語にインディペンデント	・図表 ・フローチャート

(b)で示したように電子交換システム仕様，プログラムシステム仕様，プログラムユニット仕様の3レベルで構成される．各レベル間の補助仕様として2種のインタフェース仕様がある．このインタフェース仕様を守る限りその配下の各仕様は独立に仕様の変更をして機能向上や経済化を図ることが可能である．

(1) 電子交換システム仕様　システムの目的，要求条件を明確にするために主管庁・運用会社等システム開発依頼側が規定する．主として自然言語と図表により表-1 のような項目が規定されている．なお現状では通信網の中の交換点ごとに仕様が規定されているが，将来，通信網の発展に応じて，更に網の仕様と交換点の仕様といった多層化が必要になると思われる．

(2) ハード・ソフトインタフェース仕様　要求された電子交換システム仕様を実現するために，ハードウェアとソフトウェアとの機能分担，ソフトウェア設計に必要なハードウェアとソフトウェアのインタフェースを規定する．記述法は図と表が主体であり，各種の接続構成，冗長構成，運転形式等システム構成に関するもの，各種ハードウェアごとのハード・ソフトインタフェース（例えば制御装置の命令構成，論理仕様など）が記述される．

(3) プログラムシステム仕様　電子交換システム仕様で要求された機能を，プログラムでどのように実現しているかを明確にするため製造会社等開発側が次のような事項を規定する．

①プログラム機能仕様(動的特性)：機能対応に処理のアルゴリズム，入出力の情報的・時間的因果関係を図と表により記述し，機能の実現法を明確にする．②対外部インタフェース仕様：保守運用上やシステム管理上の条件を外部に対して明確にするため，図・表のほか，必要に応じて自然言語で記述している．前述のジェネリックプログラム部と局データ部のインタフェース条件，局データの構造条件等は本仕様で規定される．③プログラム処理仕様：プログラムの設計が多数の設計者により分担される場合，共通的な処理を標準化するための手法や設計基準を明確にする．図を中心に，応用例などを交えて設計用マニュアルとして記述される．④プログラム構成仕様(静的特性)：プログラム構成概要，プログラム分割基準，ファイル管理・フ

ァイル生成条件，メモリの属性と割り付け，ファイルの仕様・品名表などで構成される．

（4） プログラム間インタフェース仕様　プログラムシステム仕様からスムーズにプログラムユニット仕様や各種インタフェース仕様を決定し，チェックするため，プログラムブロック間の情報の授受の関係を明確にする．記述形式としては，マトリックス形式の組合せごとに詳細な説明およびフォーマットなどが表および文章で規定される．

（5） プログラムユニット仕様　プログラム管理上の最小単位であるプログラムユニットごとに，インプリメント上の仕様例えば，プログラムユニットへのエントリ名，エントリする条件，実行の優先順位，実行周期などを規定する．

4. 仕様記述法

仕様の記述については世界各国とも研究を重ねているが先にも述べたとおり，部分的な技法はともかく，完全にまとまったものは見当らない．ここでは代表的な2，3の例について概略を説明することとする．

4.1 C.C.I.T.T. ・SDL[5),6)]

C.C.I.T.T. の第XI研究委員会第3作業部会を中心に SPC 電話交換機の機能仕様および内部論理の記述法として機能仕様記述言語 (SDL*) が検討されている．SDL は，我が国の D10 形交換機で呼処理サービス仕様の規定に使用している状態遷移図記法が検討のベースになり発展したもので，その基本的な考え方は，「交換動作は，信号待ちの状態(例えば被呼加入者呼出中）と信号入力（例えば被呼者応答）に反応して何らかの動作を行って次の状態（例えば通話中）に移る遷移との組合せから成る」ということである．表現形態として，状態遷移図（フローチャート）表現の SDL/GR** と形式言語表現の SDL/PR*** の2形式が検討されている．現状では，機能仕様を記述するための構文要素として，signal input state, output, task transition, dicision 等の概念および図表現のためのシンボルとそれらの接続規則が定義されている．当面は言語表現 (SDL/PR) を機械入力することにより直感的で理解し易い図式ドキュメント (SDL/GR) の自動作成を目標としている．SDL の基本概念を図-3 に示す．

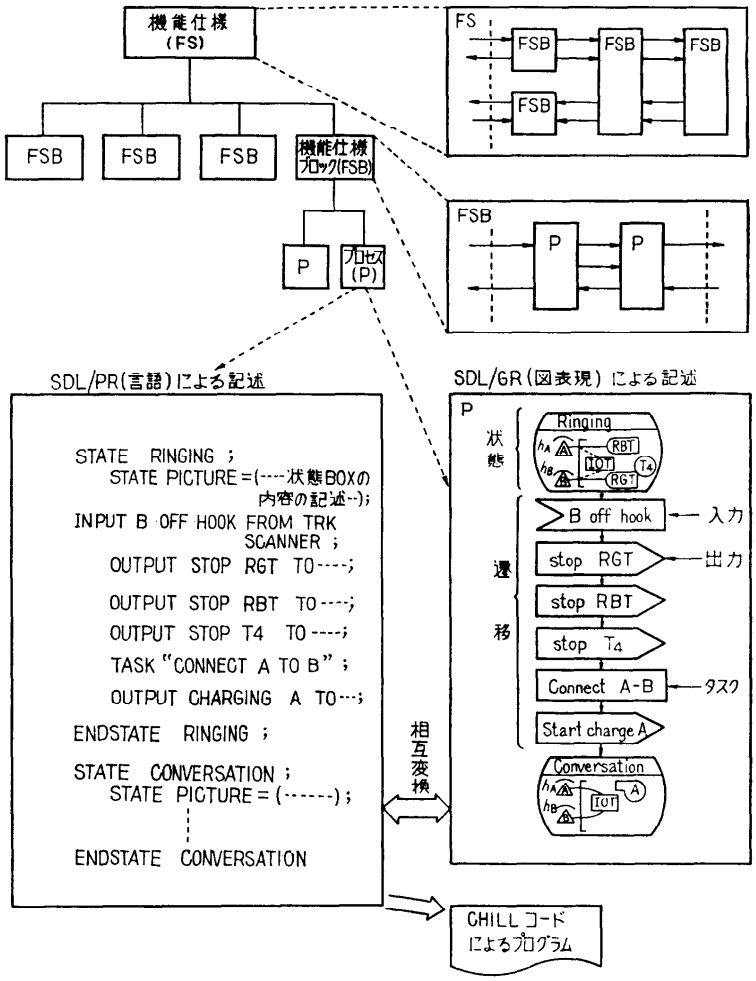


図-3 SDL の基本概念

4.2 D10 状態遷移図[7)]

我が国の D10 形交換機では，状態遷移図を文章表現と併用することにより，システムの記述を一義的か

* Functional Specification and Description Language
** Graphical form
*** Program form

つ容易にしている．表現法としては，図-3 の SDL/GR (図表現) において遷移部分を省略した形式をとっている (ただし入力は記述している)．SDL/GR との基本的な相違は，SDL/GR が **3.2** で述べた 3 レベル対応に機能の仕様・記述のレベルを設定しようとしているのに対し，D10 状態遷移図ではハードウェアの状態とソフトウェアの状態とを混合して記述し 3 レベルを一つの共通な表現にまとめていることである．D10 では，実際のソフトウェア開発において，新サービスなどの追加変更が生じた場合には，まず状態遷移図を作成してシステム設計を行い，十分チェックを行った上で，ソフトウェアのフローチャートを書くことにしている．これにより，システムの把握が十分にでき，かつ容易にシステム分析ができるのでミスも少なく，効果的である．

4.3 SX1 自動プログラミングシステム[8] (UKPO エセックス大)

フローチャート的な状態遷移図の作成および特定のプログラミング言語 (CORAL) により書かれたソースプログラムの生成を目的とする仕様入力言語をイギリス郵電省 (UKPO) およびエセックス大学で開発している．作図に関しては上述の SDL/PR から SDL/GR への変換にほぼ対応している．プログラムコードの生成では入力・判断・タスク等の要素をそれぞれ REQUEST, IF THEN ELSE, 手続き呼び出し等のステートメントへ変換している程度である．

4.3 SPLEX[9),10)] (スウェーデン・エリクソン社)

SPLEX はエリクソン社の電子交換システム AXE-10 で使用されている仕様記述レベルの言語である．

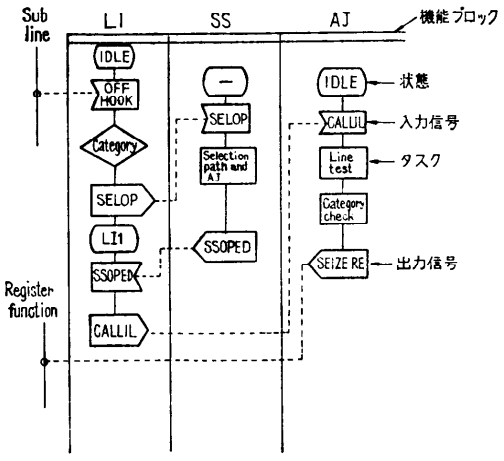


図-4 AXE-10 の機能フローチャート例

この言語はシステム機能の静的特性を主に記述・ドキュメント化しシステムジェネレーション等へ利用しているようである．なお各機能ブロックの機能の記述 (動的特性) には図-4 に示す機能フローチャートを使用している．

4.4 仕様記述言語の展望

電子計算機の技術が通信 (交換) の分野に導入され，他方電子計算機がデータ通信やコンピュータ間ネットワークのように通信と結合することになり二つの技術の合流が進むと将来的には二つのソフトウェアは区別ができなくなるものと思われる．すでに，電子交換方式の応用分野がデータ交換を始め種々多様化していることを考えると，交換ソフトウェアの特殊性を考慮することはもちろん重要であるが，むしろ通信システムの制御プロセス用ソフトウェアといった広い立場で仕様の規定法，仕様記述言語を考える必要があると思われる．その意味では，仕様記述言語の開発アプローチとしては，汎用 EDP 分野における ISDOS を構成する，PSL, SREP を構成する RSL, ALPHARD 等が指向しているような仕様の記述，仕様の早期チェックに重点を置いた言語の研究が交換分野においても盛んになるものと思われる．他方交換プログラムのジェネリック化と結びついた局データ仕様記述言語，特定分野でのシステム仕様から，自動プログラミングをねらったシステム設計用言語 (問題向き言語) 等へ発展するものと思われる．

5. 交換プログラム作成用言語

5.1 高水準言語の有用性

交換プログラムは，同一のものが数百局において長年にわたって使用されるため，最少のメモリ量で，最大の処理能力を出すことが要求される．また，交換プログラムは頻繁にハードウェアを制御せねばならない．これらの理由により，今までの交換プログラムの作成には，アセンブリ言語が用いられてきた．

しかしながら，交換プログラムは規模が大きく，かつ長年にわたって保守をする必要があるので，その生産性，保守性，信頼性 (潜在バグの少なさ) の向上が強く望まれている．

この対策としては，高水準言語の適用がまず考えられるところであり，最近の言語設計技術，コンパイラ技術の進展およびハードウェアコストの低下により，高水準言語使用にともなう欠点が軽減されてきたこともあって，最近は各国で交換プログラム作成用の高水

準言語の開発が行われており，さらにはC.C.I.T.T.でも標準言語を制定するようにまでなった．

5.2 交換プログラム作成用高水準言語の具備すべき条件

交換プログラムも，いわゆるシステムプログラムの一種であるので，本題に入る前に，まず既存のシステム記述用言語を簡単に見渡しておく．

システム記述用言語は，一般に次の目標で設計されている．

- 各種のデータ構造（配列，構造体，リスト構造等）を意図通り記述できること．
- ビット単位のデータを記述できること．
- 性能の良い（所要メモリ量が少なく処理速度の速い）オブジェクトコードに展開できるように，性能を低下させる要因が除去されていること．

今までに発表されてきたシステム記述言語には，表-2に示すように対象プログラムや設計思想の違いによって各種のものがあるが，大きな流れとしては次のものがある．

- PL/I サブセット型[19),22)]：PL/I は機能的にはシステムプログラム記述に必要なものを大部分備えているので，これからシステム記述には不用な機能や性能低下の要因を削除し，かつ不足している機能を追加したもの．数多くある．
- Pascal 系[27)~32)]：Pascal は簡明でありながら，一応システム記述用機能も備えているので，これをベースとした言語も多く発表されている．

表-2 システム記述に向いた言語の例

言 語	設 計	ベース言語	備 考
PL 360	N. Wirth	Algol 60 +アセンブラ	アセンブラレベルに Algol 風の構造を与えたもの
BLISS	CMU 大学	Algol	タイプなし言語
PL/S	IBM	PL/I	
SYSL	電電公社	PL/I	DIPS で使用
SPL, PLUS, PL/M etc.	各 社	PL/I	PL/I のサブセット的なシステム記述言語は数多い
BCPL	ケンブリッジ大学	CPL	タイプなしの言語
C言語	Bell 研	BCPL	シンプルさに特色がある
Pascal	N. Wirth	Algol 60	(汎用) シンプルさと，設計妥協のうまさに特色がある
Euclid	B. Lampson 他	Pascal	(汎用) Pascal の改良と検証機能をねらう
Concurrent Pascal	P. Hansen	Pascal	並列処理機能を持つ
MODULA	N. Wirth	Pascal	〃
MARY	Trondheim 大学 (ノルウェー)	Algol 68	Algol 68 の欠点をうけついで複雑
ADA	DoD (米)	(Pascal)	(国防省のもとに設計中)

表-3 各国の交換用言語 (C.C.I.T.T.-HLL の設計開始以前)

言 語	設 計	ベース言語	適用交換機	備考 (完成年)
DPL	電電公社	PL/I	研究用	(1975)
ESPL-1	ITT (米)	PL/I	Metaconta	(1972)
PAPE	CNET (仏)	PL/I	E11, E12	(1974)
TPL II	UKPO (英)	Algol 68		設計のみ (？)
Plex	Ericsson (ノルウェー)	混 合	AXE	(1974)
EPL	Bell 研 (米)	?	No. 4 ESS	内容未公表 (1972)
COMSYL II	〃	PL/I		実用化されず？
TSPL	GTE (米)	PL/I	TSPS	(1972)

- PL/360 系[17)]：アセンブリ言語レベルの機能を構造上 Algol 風に書けるようにしたもの．
- Algol 68 系[24)~26)]：Algol 68 も強力な言語であるが，複雑すぎて，あまり実用化はされていない．
- その他：BCPL[20)]，BLISS[18)]，C[23)] など．

このような既存のシステム記述言語を用いて交換プログラムを記述することも考えられるが，交換プログラム作成の立場からはいずれも完全な機能を有しているとは言い難く，下記項目を強化した交換分野に密着した交換プログラム作成用言語が望まれた．

- 大規模システム向きの効果的なモジュール化機能を有すること．
- オペレーティングシステム経由でなく，直接裸マシン上で走れ，かつハードウェアを制御できるオブジェクトコードを発生できること．
- リアルタイムシステムに必要な多くの部分プログラム（プロセス）を協調させながら並列に動かす処理を記述できること．
- ぼう大な共通データの記述に向くこと．

交換用として今までに開発されてきた高水準言語を表-3に示す．このうち，電電公社で開発したDPL[13),14)] (DEX Programming Language) は，共通データ記述，データ構造記述等を交換向けに機能強化した PL/I サブセット系言語であり，1975年にコンパイラを完成している．

5.3 C.C.I.T.T. における交換用言語 CHILL の制定の経緯

交換用高水準言語が各国で開発されるのにともない，C.C.I.T.T. でその標準化の検討が開始された．これは，SPC 交換機の動的な機能はソフトウェアの中に集約されているので，ソフトウェアを仕様レベルで標準化しておけば，交換機の発注，入札，保守などで多大の利点を期待できるからである．

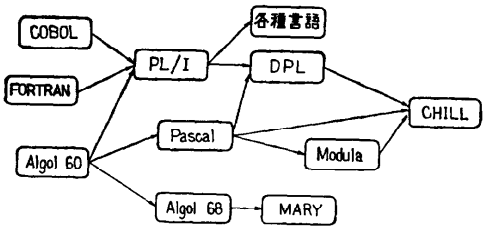


図-5　言語の流れにおける CHILL の位置

当初は既存言語をベースとして交換向きに機能拡張をする予定で，目的にそう7候補言語 (DPL[13),14)], PAPE[16)], ESPL-1[15)], Pascal[27)], MARY[26)], TPL II, Plex) を選択したが，この中から一言語に絞ることが出来ず，新規言語を開発することに決定された．こうして，1975年に専門家チーム（構成員は電電公社を含む8組織，その後 20 組織に増加）が結成され，強力に設計が進められてきた．本言語は CHILL (C. C. I. T. T. High Level Language) と呼ばれ，現在最後の推敲を行っており，1980年の第7次総会で勧告される予定である．CHILL と他言語の影響関係を，図-5に示す．CHILL はシンタックス的には PL/I から，セマンティクス的には Pascal から比較的大きな影響を受けている．

6. CHILL の設計概念と概要

6.1 CHILL の基本的な構成

CHILL で書かれたプログラムは，基本的には次の3要素から成っている．

・処理対象（変数，モード等）の定義
・処理実行手順の記述
・プログラム構造の規定

したがって，以下ではこれらの観点から CHILL の説明を行う．なお，並列処理記述機能は便宜上分けて説明する．

6.2 処理対象データの扱い

（1） タイプ（モード）付き言語

システム記述用言語では，データを整数，論理数というようにタイプ分けしたタイプ付言語 (例: Pascal) とする流儀と，データをビットパターンとして扱かい演算子，文脈でその意味付けをするタイプなし言語 (例: BLISS) とする流儀がある．CHILL では，コンパイル時チェックの強化，解読性向上の可能なタイプ (CHILL ではモードと呼んでいる) 付きとし，かつモードチェックはコンパイル時に行える設計になっている．

また，サブレンジ形データやシンボリック型データも解読性向上と省メモリ割付に有効である．

[例]
```
DCL I INT, J RANGE (0 : 1023);
    I=J+512;
DCL DAY SET (SUN, MON, TUE, WED,
             THU, FRI, SAT),
    WEEKDAY RANGE (MON : FRI);
DAY=SUCC (WEEKDAY);
```

CHILL のモード一覧を表-4 に示す．

（2） 配列，構造体モード

これらの各要素の配置はビット単位で詳細に指定することも可能である．また，構造体ではタグ付の選択フィールド（タグの値によって，対応フィールドが選

表-4　CHILL で扱かえるデータの分類（モード）

分類		表記例
スカラーデータ	整数	INT, BIN……フルレンジ RANGE (m : n), BIN (n)……サブレンジ
	シンボリック	SET (α, …, ε)……フルレンジ RANGE (β : δ)……サブレンジ
	ブーリアン値	BOOL
	ビット列	BIT (n)
	文字列	CHAR (n)
	パワーセット	POWERSET 〈要素モード〉
	ポインタ：モード自由	PTR
	ポインタ：モード固定	REF 〈モード〉
	ポインタ：可変長配列用	ROW 〈配列モード〉
	プロシージャ	PROC (……)

分類		表記例
集合体データ	配列	[ARRAY] (m : n) 〈モード〉
	構造体	1, 2 A INT, 2 B BOOL または STRUCT (A INT, B BOOL) ・CASE フィールドも可能 ・フィールド配置も細かく指定できる
並列処理用	同期	EVENT
	データ転送	BUFFER (〈長さ〉) 〈モード〉
	プロセス実体	INSTANCE

択される）も定義できる．なお，可変長配列を実現するために ROW ポインタが用意されている．

（3） ポインタモード

ポインタは交換プログラムの中では頻繁に使用される．コンパイル時のモードチェック強化の立場からは，Pascal 的なモード固定ポインタが有利であるが，現実の交換プログラムでは，同一のポインタで色々のモードのデータにアクセスせざるを得ない場合が多い．この理由から CHILL ではモード固定ポインタ (REF 〈モード〉) とモード非固定ポインタ (PTR) の両方を備えている．被ポイント変数の参照は，PL/I 風に記述する (Algol 68 のような強制変換は行わない)．

（4） モード変換

交換プログラムを書く上では，PL/I や Algol 68 のような自動モード変換はむしろプログラムを解りにくくするので，一際行われない．しかしながら，ポインタ値の加算といったモードに反した演算を必要とする場合も偶々有る．これに備えて，次例のような強制モード変換（オブジェクトには何も展開されない）を行えるようにしている．

[例] DCL P, Q REF AMODE;
INT(P)=INT(Q)+128;

6.3 処理実行手順の記述

（1）文 言 語

プログラムを“文”の集まりから構成されるとする文言語 (Pascal 等) と，“式”の組み合わせから構成されるとする式言語 (Algol 68 等) のいずれの方式を採るかは，議論の分れる所である．式言語は言語学的な

表-5 CHILL の主要ステートメント一覧

定義・宣言文			実行文		
項目		記述例	項目		記述例
モジュール構造の定義		モジュール名: MODULE SEIZE 文 GRANT 文 〈モジュール内容 END;	代入文		X=Y * (Z−U);
				式の演算子	+, −, *, /, AND, OR, XOR, NOT, //, >, >, =, =, ……etc.
	SEIZE 文	SEIZE 名前, ……;	CALL 文		CALL SUB (X, Y);
	GRANT 文	GRANT 名前, ……;	フロー制御文	IF 文	IF X>Y THEN X=I+J; Y=0 ; ELSE X=0; Y=J; FI;
プロシージャの定義		入口名: PROC (形式パラメータ); 〈プロシージャ内容 END;		CASE 文	CASE X OF 0 : 7; (0, 1): I=X; J=Y; (2, 5): I=I+1; J=J−1; (ELSE): I=0; ESAC;
プロセスの定義		プロセス名: PROCESS (形式パラメータ); 〈プロセス内容 END;		DO 文	DO FOR EVER; ……OD; DO FOR I=L TO M; ……OD; DO WHILE I<J; ……OD;
				EXIT 文	EXIT L;
				RETURN 文	RETURN X+Y;
				GO TO 文	GO TO L;
変数宣言		DCL X, Y INT; etc.	並列処理	START 文	START AA (I);
				STOP 文	STOP;
モード名定義		SYNMODE M=モード; NEWMODE N=モード; etc.		DELAY 文	DELAY E;
				CONTINUE 文	CONTINUE E;
				SEND 文	SEND BUF (I);
				RECEIVE 文	X=RECEIVE BUF;
定数名定義		SYN TMAX= 1023; etc.		DELAY CASE, RECEIVE CASE 文	DELAY CASE (E1): 実行文の並び (E2): 実行文の並び ESAC;
			例外発生		CAUSE OVERFLOW;

面白みがあるが，技巧的プログラミングを助長しがちであるので，CHILL は単純明解な文言語としている．CHILL の文一覧を表-5 に示す．

（2） フロー制御文

自然に構造の良いプログラムを書けるようなフロー制御文を備えること，複合文は必ず閉じ括弧をもつこと (IF……FI；等)，最適化コンパイラによるフロー解析が容易なことなどに留意して設計されている．

（3） 例外時処理記述機能

実行時にメモリ確保を出来なかった場合など，例外時の処理を構造的に書く機能を有している．サブルーチン内で生じた例外条件を，呼出し側で処理する例を次に示す．

```
[例] A : PROC
          ⋮
          CALL B(I+J) ON(Z) : "例外時処置"
                                    END;
          ⋮
          END;
       B : PROC (X INT IN) EXCEPTIONS(Z);
          ⋮
          CAUSE Z; /*例外条件を発生させる*/
          ⋮
          END;
```

6.4 プログラム構造

（1） 名前の有効範囲とブロック構造

交換プログラムのような大規模プログラムを作成するためには，名前の有効範囲を効果的に規制できる機構が必須である．その一つとして，CHILL ではプロシージャ，プロセス，ビギンブロックから成るブロック構造を採っている．

しかしながら，システム記述用言語においては，ブロック構造だけでは十分な機能と言えず，さらに名前の有効範囲をユーザが明示規制できるような機構が望まれる．これに応えるのが，次のモジュール構造である．

（2） モジュール構造

モジュールは，名前の有効範囲を GRANT 文と SEIZE 文とにより明示規制できるようにした，キイワード MODULE と END に囲まれた部分プログラムである．そのモジュール内で定義された名前のうち GRANT 文で指示されたもののみがモジュール外から参照可能となり，逆にあるモジュールからその外側で見える名前を参照可能にするためには，SEIZE 文で指示する必要がある（図-6 参照）．本機能は，プログラムの分割作成，モジュール間インタフェースの明確化，情報隠蔽，一種のデータ抽象化などの目的に有効である．

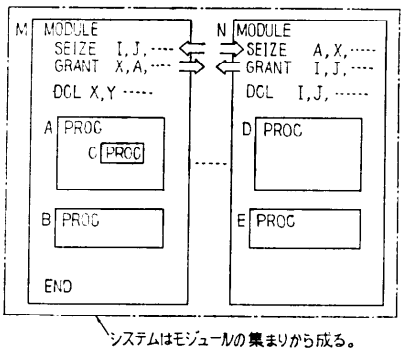


図-6 プログラムの枠組み

（3） プロシージャ

PL/I に準ずるが，パラメータ引継ぎは call by value，call by result，call by reference を備えている．（プロシージャモードパラメータの使用により，call by name の効果も実現できる．）

6.5 並列処理記述機能[33)~45)]

交換プログラムのような実時間多重処理やマルチプロセッサシステムでは，多くの部分プログラム（プロセス）を協調させながら並行して動かす必要がある．一般の計算機システムでは，並列処理の同期等の扱いの大部分を汎用オペレーティングシステム (OS) にゆだねており，プログラミング言語自体は OS マクロのインタフェースのみを用意しているのが通常である．しかしながら，交換プログラムのように専用の実行管理プログラムと，その下で通信しながら並列に走るプロセスから成るマルチプログラミングシステムを高水準言語で記述するためには，言語仕様自体に並列処理を明示的に記述できる能力が必要となる．

この機能を並列処理記述機能と呼び，CHILL の場合，次の内容を行う機能を備えている．

- プロセスの定義とその生成・消滅
- プロセス間の同期
- プロセス間のデータ転送
- 共用リソースへの排他的アクセス

（1） プロセスの定義

並列的に実行できる部分プログラムの定義をプロセス定義と呼び，次のように書く．

```
AA : PROCESS (形式パラメータの並び);
        変数宣言，プロシージ定義等の並び
        実行文の並び
        END;
```

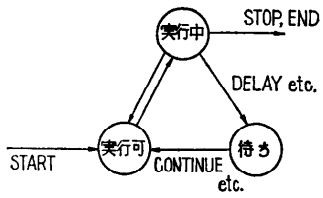


図-7 プロセスの状態遷移

これをコンパイルするとリエントラントコードになる．つまり一つのプロセス定義を使って複数の処理を並列的に走らせることができる．このような実行の処理単位をプロセス実体 (Process instance：以後プロセス) と呼ぶ．プロセスは START 文によって生成されスタートする．

START AA (実パラメータの並び)；

プロセスは，**図-7** に示す状態遷移をする．

(2) プロセス間の同期

プロセス間の同期を行う機構としては，EVENT モードと DELAY 文，DELAY CASE 文，CONTINUE 文を用意している．EVENT 変数はイベント待ち行列であり，DELAY 文を実行したプロセスがつながれる (待ち状態)．CONTINUE 文は，待ち行列の先頭のプロセスを実行可状態に移す (但し，待ち行列が空のときは無視される)．

```
[例 1]  DCL EV EVENT;
        AA: PROCESS ( );
              ⋮
            DELAY EV;
              ⋮
            END;
        BB: PROCESS ( );
              ⋮
            CONTINUE EV;
              ⋮
            END;
[例 2]  DCL E 1, E2 EVENT;
        CC: PROCESS ( );
              ⋮
            DELAY CASE
          (E1): "E1 発生時の処理"
          (E2): "E2 発生時の処理"   } 先に発生したイベントに対応するものが実行される.
            ESAC;
              ⋮
            END;
```

[例 2] は，複数イベントのうちいずれかが発生するまで待つ場合の例である．

この機構は基本的なものであるので，後述のクリティカルリージョンと組み合わせて，Semaphore 機構や IBM 系 OS の EVENT 機構なども実現できる．

(3) プロセス間のデータ転送

バッファを介したプロセス間データ転送は，EVENT 機構と後述のクリティカルリージョンを用いて容易に実現できるが，頻繁に用いられることから BUFFER 機構として言語内に組み込まれている．

```
[例]  DCL B1, B2 BUFFER INT;
      AA: PROCESS ( );
            ⋮
          SEND B1 (I);
            ⋮
          END;
      BB: PROCESS ( );
          DCL X, Y INT;
            ⋮
          RECEIVE CASE
        (B1 IN X): "B1 から受けた場合の処理"
        (B2 IN Y): "B2 から受けた場合の処理"
          ESAC;
            ⋮
          END;
```

(4) 共通リソースへの排他的アクセス

マルチプロセッサシステム等で注意を要する問題の一つに，共用リソースへの排他的アクセスの保証がある．この実現法としては，マシンレベルでは Test & Set 命令，言語レベルでは Semaphore 機構，ENQ/DEQ 機構などを使う手法があるが，機能や解り易さの点で十分とは言い難い．これに対し，CHILL では，Hoare の提案[42] であるモニタ機構をモジュール機能と融合させた "クリティカルリージョン" (以下リージョンと略す) を用意している．リージョンは変数等の宣言とプロシージャ定義とから成る特殊なモジュールであり，あるプロセスがリージョン内プロシージャを実行している間は，他のプロセスがそのリージョン内のプロシージャを呼んでも，前者がリージョンを解放するまで待たされるように働く．ここに，リージョンの解放とは，リージョン内プロシージャから制御が戻されるか，もしくはリージョン内で待ち状態に入った時に起る．つまり，DELAY 文の実行によりリージョンは解放されるので，いわゆる条件付クリティカルセクションも容易に実現できる．

6.6 CHILL の現状

CHILL の言語仕様は，1980年の第Ⅶ次 C. C. I. T. T. 総会における勧告に向けて，現在最終的な推敲が行われている．本仕様書は Algol 60 形式の仕様書[12] と Denotational Semantics 記法による仕様書[56] とから成っている．

本言語は国際間で設計されたものだけに，当初のシンプル化の目標に反して，かなり大規模となったが，

[例]

```
MM: REGION; GRANT PUT, GET;
    DCL BUF CHAR(32), STATE SET(EMP, FULL) INIT=
        EMP, EVFULL, EVEMP EVENT;
    PUT: PROC(X CHAR(32) LOC);
        IF STATE=FULL THEN DELAY(EVEMP); FI;
        BUF = X; STATE = FULL; CONTINUE(EVFULL);
        END;
    GET: PPOC( X CHAR(32)LOC);
        IF STATE=EMP THEN DELAY(EVFULL); FI;
        X = BUF; STATE = EMP; CONTINUE(EVEMP);
        END;
    END MM;
AA: PROCESS ( );
    :
    CALL PUT(Z);
    :
    END;
BB: PROCESS ( );
    :
    CALL GET(U);
    :
    END;
```

フルセットのインプリメントは強用していない．現在CHILL のコンパイラは，電電公社，フィリップス(蘭)，ジーメンス（西独)，ITT（米)，北欧主管庁連合等で作成されている．

電電公社の CHILL コンパイラは，1979 年に完成し，新交換プログラムの作成に本格使用されている．このコンパイラは，データフロー解析に基づいた広域レジスタ割付など，最適化処理を特に重視した設計を採っている．

CHILL は，C.C.I.T.T. という国際機関によってSPC 交換機用言語として勧告されるものだけに，交換分野では広く使われることになろうが，交換専用言語ではなく，一般のシステム記述言語としても充分に使用できる機能をもっている．

なお，米国防総省 (DoD) で設計の進められてきた標準言語 ADA[46)~49)] の最終案が最近発表されたが，機能的に CHILL と類似点の多いことに興味がもたれる．ADA は必要以上に複雑化しているきらいがあるが，(Dijkstra の批評[48)] にも関わらず) DoD の影響力の大きさからみて，今後大いに注目されよう．

6.7 交換プログラム作成用言語の展望

この分野での今後の研究の話題になるかと思われる 2,3 の項目をあげてみる．

- 抽象データ操作機能[51),52)]：これはデータとそのアクセスルーチンとを，外部からはより高レベルの論理データ構造として見えるようにカプセル化する機能であり，研究も進んでいる．
- 分散プロセス[44),45)]：多数のマイクロプロセッサが通信線経由で通信しあう処理方式の場合には，それに適した並列処理記述機能が必要となってこよう．
- 特定問題向き言語：適用分野を特定問題に限定してその中でかなり高レベルな問題向言語を実現することが考えられる．

7. あとがき

交換プログラムの作成に使用される記述手段のうち，仕様記述言語と高水準プログラミング言語について解説してきた．これらは，わが国でも比較的早くから研究されており，C.C.I.T.T. で研究テーマとして採り上げられてからは積極的に寄与してきている．

ここで述べられたものは，交換プログラム作成工程の一部の改善に役立つものであり，今後に残されている問題は，まだまだ多いと言える．

参考文献

1) 村田：交換技術とソフトウェア，信学誌，58, 11 p. 1168 (昭 50-11).
2) 例えば，宮本，東谷：ソフトウェア要求定義技術，bit, 9, 6, p. 616.
3) 電子交換ソフトウェア分科会中間報告，付属資料 その 3，電気通信技術委員会交換部会 (昭 50-01).
4) 武田：ソフトウェアの階層構造，信学誌，58, 11, p. 1228 (昭 50-11).
5) C.C.I.T.T.: REPORT (GENEVA, 19-23 JUNE 1978), C.C.I.T.T. SG XI/3-4 (June 1978).
6) C.C.I.T.T.: REPORT ON THE MEETING HELD IN GENEVA FROM 19 TO 23 JUNE 1978, C.C.I.T.T. SG XI/3-1A (June 1978).
7) Kawashima, H. et al.: Functional Specification of Call Processing by State Transition Diagram, IEEE Trans., COM-19, 5, p. 581 (1971).
8) Hill, A.J. et al.: The SXI automatic Programing Project, Third International Conference on S.E.T.S.S., 27-29 (June 1978).

9) AXE 10 Software Structure and Feature, Ericsson Review, 53, No. 2, p. 90 (1976).
10) Hemdal, G. et al.: SPLEX-A Specification Toll for AXE 10, ISS '76, 25-29 (Oct. 1976).
11) C. C. I. T. T., PROPOSAL FOR A RECOMMENDATION FOR A C. C. I. T. T. HIGH LEVEL PROGRAMING LANGUAGE, C. C. I. T. T. SG XI (May 1977).
12) C. C. I. T. T., Z. 200 CHILL: A Recommendation for a C. C. I. T. T. High Level Language, C. C. I. T. T. SG XI (1980 予定).
13) 工藤他: 交換プログラム作成用言語 DPL/I, 研実報, 24, 11 (1975).
14) Kakuma 他: DPL—A High Level Programming Language for Electronic Switching Systems, ISS '76 (Oct. 1976).
15) ITT 社: ESPL-1 REFERENCE MANUAL ITT.
16) Rozmaryn, C. et al.: PROGRAMING LANGUAGES LP1 AND LP2, ISS '76 (Oct. 1976).
17) Wirth, N.: PL 360, A Programming Language for the 360 Computers, JACM, 15 (Jan. 1968).
18) Wulf, W. A. et al.: BLISS: A Language for systems programming, CACM, 14 (Dec. 1971).
19) IBM マニュアル: GC 28-6794-0, Guide to PL/S II.
20) Richards, M.: BCPL: A Tool for Compiler Writing and Systems Programming, Proc. AFIPS SJCC, 34 (1969).
21) McKeeman, W. M. et al.: A Compiler Generator, Prentice Hall (1970).
22) 寺島信義他: システム製造用言語 SYSL-2 の設計, 情報処理, Vol. 16, No. 8 (1975).
23) Ritchie, D. M. et al.: The C Programming Language, BSTJ, 57, No. 6 (1978).
24) Wijngaarden, V. et al.: Revised Report on the Algorithmic Language Algol 68, Acta Informatica 5 (1975).
25) Holdworth, D.: System Implementation in Algol 68-R, Software-Practice & Experience, 7 (1977).
26) Rain, M.: MARY Programmers Reference Manual, RUNIT, Trondhein (1973).
27) Wirth, N.: The programming Language Pascal, Acta Informatica, 1 (1971).
28) Haberman, A. N.: Critical Comments on the programming Language Pascal, Acta Informatica, 3 (1973).
29) Hoare, C. A. R. et al.: An Axiomatic Definition of the programming Language Pascal, Acta Informatica, 2 (1973).
30) Welsh, J. et al.: Ambiguities and Insecurities in Pascal, Software-Practice & Experience, 7 (1977).
31) Lampson, B. W. et al.: Report on the programming Language Euclid, ACM Sigplan Notice, 12, 2 (1977).
32) Jensen, K. et al.: Pascal-User Manual and Report, Lecture Notes in Computer Science, 18, Springer Verlag (1974).
33) Hansen, P. B.: The programming Language Concurrent Pascal, IEEE Tr., SE-1, No. 2 (1975).
34) Hansen, P. B.: CONCURRENT PASCAL REPORT, Information Science, California Institute of Technology (June 1975).
35) Hansen, P. B.: Operating System Principles, Prentice Hall (1973).
36) Hansen, P. B.: Concurrent Programming Concepts, Computing Serveys, 5, No. 4 (1973).
37) Dijkstra, E. W.: Co-operating Sequential Processes. In F. Gennys Programming Languages, Academic Press (1968).
38) Dijkstra, E. W.: The structure of the THE-Multiprogramming System, CACM, 17 (May 1968).
39) Wirth, N.: Toward a Discipline of Real-time Programming, CACM, 20 (Aug. 1977).
40) Wirth, N.: Modula: A Language for Modular multiprogramming, Software-Practice & Experience, 7, No. 1 (1977).
41) Wirth, N.: Design and Implementation of Modula, Software-Practice & Experience, 7, No. 1 (1977).
42) Hoare, C. A. R.: Monitors: An operating system structuring concept, CACM, 17, No. 10 (1974).
43) Staunstrup, J. et al.: Platon: A High Level Language for Systems Programming.
44) Hansen, P. B.: Distributed Processes: A Concurrent Programming Concept, CACM, 21, (Nov. 1978).
45) Hoare, C. A. R.: Communicating Sequential Processes, CACM, 21 (Aug. 1978).
46) DoD: PRELIMINARY ADA REFERENCE MANUAL Sigplan Notice, 14 (June 1979).
47) DoD: Requirements for High Order Computer Programming Languages Revised IRONMAN, Sigplan Notice, 12 (Dec. 1977).
48) Dijkstra, E. W.: DoD-1: The Summing Up, Sigplan Notice, 13 (July 1978).
49) Shaw, N.: TARTAN Language Design for the Ironman Requirement: Reference Manual, SIGPLAN Notice, 13 (Sept. 1978).
50) Wulf, W. et al.: Global variable Considered Hormful, SIGPLAN Notice (Feb. 1973).
51) Liskov, B.: Abstruction Mechanism in CLU, CACM, No. 8 (Aug. 1977).
52) Dahl, O. J. et al.: SIMULA 67 Common Base Language, Publication No. S-2, Norweigian

Computing Center, Os 10 (1968).
53) Backus, J.: Can Programming Be Liverated from von Neumann Style? A function style and It's Algebra of Programs, CACM, 21 (Aug. 1978).
54) Zahn, C. T.: A control statement for natural top-down structured programming, Symp. on Prog. Lang., Paris (1974).
55) Goodenough, J. B.: Exception Handling: Issues and a Proposal Notation, CACM, 18 (Dec. 1975).
56) C.C.I.T.T.: A FORMAL DENOTATIONAL SEMANTICS DEFINITION OF CHILL, C. C. I. T. T. SG XI (1980 予定).
57) C. C. I. T. T.: INTRODUCTION TO CHILL: The C. C. I. T. T. high level programming Language, C. C. I. T. T. SG XI (1980 予定).

(昭和54年11月1日受付)