ALGOL (JIS 7070) コンパイラの開発

Development of ALGOL (JIS 7070) Compiler

電子計算機言語ALGOLは構文が簡明で機能が大きく、主として科学技術計算に向いた言語である。ALGOLにはJISが定まっておらず、今回その最高の機能レベルであるJIS 7070のALGOLコンパイラを開発した。

本ALGOLは、JIS 7070の機能に複素数データ、記号列データを処理する機能を追加し、豊富なデバッグ用機能を追加してある。また、プログラムの具体的書き方である金物表現を、使用できる金物と書きやすさの度合いに応じて3種類用意した。

一方、機能の大きさのため目的プログラムの実行効率が低下するのを防ぐため、記憶域の割付けパラメータの受渡しの最適化に努め、類似の言語であるFORTRANと比べて実行効率の低下は、平均的プログラムにおいてたかだか40%程度とすることができた。なお、ALGOLは、構造化プログラミングにも適した言語である。

浜田穂積* Hozumi Hamada
 山野紘一* Koichi Yamano
 村上孝也** Takaya Murakami
 高橋 修** Osamu Takahashi

1 緒 言

電子計算機(以下、計算機と略す)言語ALGOL(Algorithmic Language)は主として科学技術計算に適した言語である。Naurら⁽⁵⁾によって1960年に最初の言語仕様が発表され2年後に小改訂⁽⁶⁾がなされたが、これが現在に至るまで、計算機言語に関する理論的な研究の基礎となり、また多くの研究材料を提供してきた。我が国においては、国際標準化機構(ISO)の標準化の動きに応じて、1967年にJIS⁽³⁾⁽⁴⁾が制定された。JIS規格は基準言語(7,000, 6,000, 5,000, 4,000, 3,000)と入出力手続き(70, 60, 50, 40, 30)についてそれぞれ5段階になっていて、7,000と70がそれぞれ最高の機能レベルである。

今回、日立製作所においてこの7,000と70とを組み合わせ、JIS-ALGOL7070に相当するALGOLコンパイラを開発した。

本稿は、その言語仕様と実行方式における特徴及び作成されたコンパイラによって作り出されるオブジェクトプログラムの実行効率の測定結果につき述べる。

2 ALGOL言語仕様

HITAC 8000シリーズALGOLコンパイラは、現在オペレーティングシステムEDOS/EDOS-MSO(Extended Disc Operating System-Multi Stage Option)のもとで動く。言語仕様はほとんどJIS 7070の規格票と同じなので、それからの変更部分についてのみ述べる。

2.1 静的構造

(1) ALGOLプログラムは、主プログラムといくつかの外部手続きの宣言であるコンパイル単位から成る。これはFORTRAN(Formula Translator), PL/1などと同じ機能であって、プログラムのモジュール化をコンパイル単位として行なえるようにするものである。この場合、他のコンパイル単位としての手続きを陽に呼び出すところでは、コンパイラに手続きの型、パラメータの個数と型の情報を与えるために、手

続きの宣言を(本体はEXTERNAL文によって略記する)置かねばならない。

(2) 手続きの宣言においては、名前変え(Call by Name)のパラメータについても必ず規制(Specification)をしなければならない。これは、名前変えのパラメータについても、効率のよいオブジェクトプログラムが生成できるようにするための制限である。

(3) 占有型(Own Type)の配列の宣言においては、上下限ともに、定数で与えねばならない。これは、占有型の配列に対する実行時の記憶域割付けに必要な操作を単純化することによって、実行効率をよくするための制限である。なお、上記(2), (3)については、ほとんどすべてのALGOLコンパイラがこの制限を課している。

(4) プログラムの具体的表現方法として、金物表現(Hardware Representation)を次の3種類用意した。

(a) キーワードと同じ名前も自由に宣言でき、また、使用できる字種が少ない機械でも使える(JIS推奨案)。

(b) (a)において、逃げ文字(Escape Character)であるピリオドが不自然で、プログラムが読みにくいという場合、キーワードを予約語とする英大文字を中心にした表現。

(c) (b)では英字が大文字だけなので読みにくいという場合、もしラインプリンタに小文字の活字があって使用できれば、各種文献で使われている表現とほとんど同様に書けるという表現。

(5) 行末のコメント機能を設けた。

(6) 文の継続、特に文字列定数が次の行にわたるときの表現を、直観に近い分かりやすいものにできるようにした。

(7) 特定の印が行頭にある場合、その行をオプションによって、コメントとみなすか、原プログラムとみなすかを変更できるようにした。この機能はデバッグに利用できる。

(8) FORTRANプログラムと結合することが可能である。この場合、手続きの宣言の本体としてEXTERNAL文を用い、且つFORTRANと指定することによって行なわれる。

* 日立製作所システム開発研究所 ** 日立製作所ソフトウェア工場

2.2 動的構造

(1) コンパイル単位をいくつかまとめたものを単位として、プログラムのオーバーレイができる。データのオーバーレイについては、特に指定しなくても常に行なわれる。

(2) ALGOLの基本的な文法のわく内で、便利なデバッグ機能を豊富に備えた。ALGOLには本来デバッグに便利な機能が含まれていないが、デバッグ機能は、特にバッチ処理向き言語にとって必須であろう。そこで次の機能を設けた。

(a) 外部手続きが呼び出されたとき、パラメータの型と精度が呼出し側と呼び出され側とで矛盾を起ささないかどうかのチェックをする（オブジェクトを、オプションによりコンパイラが作り出す）。

(b) 各種の誤りについて、それが起こったかどうかを検出する標準手続き（宣言なしに呼び出す）として、表1のものを用意した。

次の項目はJISで規定されているが、特徴的であるので特に述べる。

(3) プログラムの回帰的(Recursive)呼出しが可能である。これによって数学公式などの帰納的な関係を、直観に近く自然な形で簡潔に記述できる。また、これと名前変えのパラメータ機構との組合せによって、複雑な操作を簡潔に表現できる場合がある。

2.3 データの型と演算機能

(1) 複素数データを追加した。科学技術計算に向けた言語としては、複素数が扱えないのは不便である。FORTRANでも複素数が扱えるので、ALGOLでも扱えるようにした。このためいくつかの標準関数を追加する必要が生ずるが、それらを含めて、標準関数名のリストを表2に示した。

(2) 算術データの精度を、長短2種類のうちからオプションによって選べる。計算の精度が要求されるならば長精度を、多量のデータを扱う場合、あるいは計算速度が要求される場合には短精度を用いる。

(3) 文字列データを追加した。現在ではALGOLを使って言語プロセッサなどを作ることでもできるようになった。そのためには、ある程度の便利な文字列データを扱う機能が欲しくなり、文字列をデータとして扱えるようにした。文字列データに行なわれる演算は、連結と部分列の取出しができる。

(4) 入出力処理はすべてJISの規格70に従った（サポートが望まれているものも含む）。それらは表3に示すとおりである。簡単な入出力で済ませたいときは、第一種を用いる。データ

表1 デバッグ用標準手続き 異常状態の発生したことを知るために、これら8種の手続きが用意されている。

手 続 き 名	検 出 さ れ る 誤 り
overflow	浮動小数点あふれ
fixedoverflow	固定小数点あふれ
underflow	浮動小数点下位あふれ
zerodivide	ゼロによる除算
size	サイズエラー
subscriptrange	添字不適合
stringsize	文字列長不適合
error	(上記を含めた)各種エラー

表2 標準関数 複素数データを追加したため、第2列後半から第3列までの関数が必要となる。第2列は実数値、第3列は複素数値を返す関数である。

第 1 列	第 2 列	第 3 列	第 4 列
entier	exp	cexp	length
sign	ln	cln	equiv
rem	sqrt	csqrt	
	sin	csin	
abs	cos	ccos	
log	arctan	carctan	
mod	tanh	ctanh	
max <i>n</i>			
min <i>n</i>	Re	cmplx	
(<i>n</i> = 2, ..., 9)	Im	conjugate	
random	arg		

表3 入出力標準手続き JIS70規定にincomplex, outcomplex, instring及びoutstringを追加して、入出力ともに上記19種の手続きが用意されている。

区分	入 力	出 力
第 一 種	inlinefeed	outlinefeed
	ininteger	outinteger
	inreal	outreal
	incomplex	outcomplex
	insymbol	outsymbol
	instring	outstring
第 二 種	inarray	outarray
	input <i>n</i>	output <i>n</i>
	(<i>n</i> = 0, 1, ..., 9)	
第 三 種	inlist	outlist
一 時 記 憶	get	put

の型によってこれらの手続きの中から必要なものを選んで呼び出すと、決まった書式で入出力が行なわれる。書式を指定したいときは第二種を用いる。第三種を用いると、入出力媒体上のレコード、ブロックを意識した処理ができる。その他、一時記憶に向けた入出力手続きもある。

3 実行方式

今回のALGOLコンパイラの開発において、特に力を入れくふうした実行方式について、その主要な部分を述べる。実行方式の良否は、ALGOLコンパイラの性能を著しく左右するものである。ALGOL言語がFORTRAN言語ほど普及していない理由の一つに、性能の良いALGOLコンパイラがないということがある。

3.1 動的記憶域の割当て

2.で述べたように、本ALGOLでは回帰的呼出しが可能

であり、また、配列の大きさのように実行時でないで定まらないものがあるので、記憶域は実行時に割り当てる動的記憶域割当てによって確保される。この記憶域の確保に対しては、ALGOLプログラムの入口において、実行に必要な大きさを計算し、ALGOLプログラムが管理する方式を採用した。

ALGOLにはブロックというものがあるが、これは名前の有効範囲を定めるためのものである。従って、ブロックで必要とする記憶域は、配列などのように動的に大きさが定まるものを除けば、手続きの入口において割り当てられた記憶域の中に、定まった大きさの記憶域を割り当てることができる。また、排他的な関係にあるブロックの記憶域は、同時に参照することがないので、互いにオーバレイさせることにより記憶域を節約できる。しかし、ブロックのネストの仕方次第では記憶域がむだになることが起こるが、それは単純変数に限られ、しかも多くの場合その大きさは小さいものである。このように、手続き単位に動的記憶域の割当てを行なう方式は、Gries⁽¹⁾によって与えられている。この方式は手続きの入口で、一度記憶域の割当てを行なうだけでよいから、実行時間の能率は良いものとなる。以上のことから、動的記憶域の割当ては手続き単位に行なうことにした。

3.2 非局所の変数の参照

手続き単位に割り当てられた記憶域の変数の参照の仕方について述べる。ALGOLでは、占有型変数以外の変数は、前述した記憶域に、動的に割り当てるのが自然である。動的に割り当てるとは、手続きAから手続きBを呼び出すとき、手続きBのための記憶域を手続きAのための記憶域に隣接して次にとる方法を言う。この方法の特徴は、変数割当ての記憶域が完全に二分され、一方は割り当てられた場所、もう一方は割り当てられていない場所となつて、その境界がただ一つだけ存在することになる。手続きBの実行が終了したときに、その記憶域を割り当てられていない場所に属させれば、ちょうどスタックの動きと同じになる。従って、現在動いている手続きのための記憶域は、上述の境界に隣接している部分、すなわち最近に割り当てられたものである。

このような状況の中で、現在動いている手続き中で参照できる変数は、仮パラメータを除くと、必ずしも割り当てられているものすべてではない。すなわち、参照できる変数は次のものである。

- (1) 自分自身の記憶域中に割り当てられた変数
- (2) この手続きを静的に含む手続きにおいて、それぞれの手続きのための記憶域中に割り当てられた変数

ここで、「ある手続きを静的に含む手続き」とは、プログラム上で、「ある手続き宣言の属している手続き」のことをいう。

この分類において、(1)を局所変数、(2)を非局所変数と呼んでいる。ところで、手続きは回帰的に呼び出されることもあり得るから、非局所変数については、2個所以上で割り当てられている場合、それらのうち、どちらを参照すべきかを明確にさせねばならない。これについては、文法に定めてあって、現在動いている手続きを確立させた手続きの変数が参照できることになっている。

非局所変数の識別は上述のように複雑であるから、どの手続きにおいてもそれに制御が移ったとき、参照できる記憶域を(少なくとも直接上のレベルだけでも)確定しておくことが必要となる。この場合、直接上のレベルの記憶域へのポインタをスタティックチェーンといい、スタティックチェーンをたどって得られる参照可能な全部の記憶域へのポイン

タを集めたものをディスプレイと呼ぶ。本ALGOLでは、このディスプレイを、前述した手続き単位ごとに割り当てられる動的記憶域に作成する方式をとっている。この方式は、ディスプレイのための記憶域が手続き単位ごとに必要になるとともに、ディスプレイの作成に時間がかかる。しかしながら、ディスプレイの作成よりも、ディスプレイを参照して、変数の参照を行なう回数の方が多いため、手続きごとに作成すればよいという制御の単純さを考慮して、上のような方式を採用した。

3.3 手続きの制御

3.2に述べたディスプレイは、手続きが実行されるべき環境を示すものであるから、環境情報とも呼ばれる。ところで、手続きが呼び出されるとき次の情報が必要である。

- (1) もどりアドレス
- (2) パラメータリストのアドレス
- (3) レジスタ退避域のアドレス
- (4) 呼び出される手続きの動的記憶域を取るべきアドレス
- (5) 実パラメータが評価されるとき環境情報
- (6) 呼び出される手続きの非局所的な環境情報

非局所的な環境情報とはその手続きを直接に定義した手続きの動的に生きている環境情報のことである。手続き呼出しにおいて、(1)~(5)の情報は簡単に得ることができるが、(6)に関しては、少くもふうを要する。通常のものであって、仮パラメータでない実手続きにおいては、非局所的な環境情報は、呼出しを行なう手続きの環境情報自身であるか、あるいは、それより直ちに得られるものである。ところが、仮手続き呼出しの場合には、同様の取り扱いができない。仮手続き呼出しとは、名前変え仮パラメータとして手続き名を受け渡し、この仮手続きを呼び出すことである。この場合には、この受け渡された手続きのアドレスだけでは不十分で、その手続きが実行されるとき環境情報も必要となる。これらの情報を手続きデータと呼ばれるものの中に作成し、それを受け渡すことによって、非局所的な環境情報を得ることができる。

3.4 パラメータ機構

ALGOLのパラメータ機構には、値とりと名前変えの2種類がある。値とりパラメータ機構は、手続きの本体を包含する仮想的ブロックを考え、そのブロックに局所変数が定義され、その変数に実パラメータの値を必要ならば変換を行ない、代入することを意味する。どのような変換を行なうべきかは、実パラメータの型に依存するため、実パラメータの型に関する情報を必要とする。本ALGOLコンパイラでは、パラメータリストの各要素に、実パラメータの型に関する情報を入れたテンプレートを付属させた。そして、手続きの前置コードとして、テンプレートの情報をもとに、値の変換、代入を行なう処理をするようにして、値とりパラメータ機構を取り扱った。

名前変えパラメータ機構は、仮パラメータを、対応する実パラメータで置き換えることを意味する。このことを実現する処理方式として、実パラメータをアドレス計算手続き(Ingerman⁽⁷⁾に従いサンクと呼ぶ)化し、仮パラメータが評価されるごとに、このサンクを呼び出す方式が考案された⁽⁷⁾。サンクは手続き呼出しの機構に類似しているため、多くの時間がかかり、オブジェクトプログラムの性能を低下させる原因となる。そこで次のような最適化を行ない、オブジェクトプログラムの性能を向上させた。

- (1) サンクは回帰的に呼び出されることがないので、一般の手続きのように専用の動的記憶域を作る必要がなく、サンク

が定義された環境情報のみを用いる。

(2) 実パラメータが定数、あるいは仮パラメータでない変数の名前の場合には、サックは常に同一の実パラメータのアドレスを返すだけなので、パラメータリストの中には、実パラメータのアドレスそのものだけを入れておく。

特に(2)については、ALGOLの実行が遅いという従来の不評を回復するために、力を注いだ部分であり、今回のALGOLの最大のポイントの一つである。

3.5 飛越し文

飛越し文は、行先式を評価して得られる名札にブランチする文である。また、行先式は、手続きのパラメータとしての受け渡し許されている。このようなことから、名札を算術型データなどと同様に扱うことにし、名札データというものを設けた。名札データは名札のアドレスと環境情報の二つから構成することにした。

飛越し文の実行は、次のように行なわれる。行先式を評価し、名札データを得る。次に、同一手続き内へのブランチかどうかを判定する。同一手続き内へのブランチであれば、名札のアドレスを得てブランチする。そうでないならば、名札データの環境情報を利用して、名札のアドレスにブランチする。

しかし、同一手続き内からのみ参照され、且つ、その行先式が条件行先式、スイッチ呼出しでない形で参照される仮パラメータでない名札については、名札データを作成しない。飛越し文の行先式がこのような名札である場合には、コンパイル時に、直接オブジェクトプログラムにブランチアドレスを埋め込むように最適化を行なった。更に、このような場合、8000系の機械では、アドレスを直接指定するためのディスプレイメントが小さいため、それを超えるようなときには、ブランチ命令の前にロード命令などの前置命令を置かなければならない。この前置命令を必要最小限にするため、手続きの入口から4,096バイトごとにページ分けし、同一ページ内のブランチについては最適化し、他のページにまたがるブランチについては、前置命令を置くようにした。

3.6 入出力手続き群の実行方式

ALGOLの入出力手続き文には、表3に示されているものがある。表3において、第一種のクラスの入出力手続き文は、書式として標準書式を用いるものであり、第二種クラスの入出力手続き文は、書式の指定のできるものであり、第三種クラスの入出力手続き文は、書式の書き換えやファイルの行、ページの制御などができるものである。JISの規格⁽⁴⁾によるとこれらの処理は、まず第一種は第二種で、第二種は第三種で定義してあり、第三種が最大の機能を持っていて一般的であるため、第三種の処理が細かい手順を追って規定してある。これをそのままインプリメントすると第三種のものを作成して、あとはつなぎの処理を行なうルーチンを作ればよいということになる。

これでは、それほど処理の複雑さを要しないものを、機能の大きいもので行なうことになり、処理時間を考えると能率の良いものにならない。この点を考慮して、処理機能が小さく、ユーザーの使用度の大きいと思われる第一種の入出力手続き文の実行時間を最も小さくするように実行方式を設計した。第二種の入出力手続き文も、第三種ほど複雑にしないように設計したが、第三種の入出力手続き文は、その機能の大きさより複雑なものにならざるを得なかった。このことによって、入出力手続き文は、その機能とユーザーの使いやすさの度合いに応じて、能率よく行なえるようになっている。

4 評価

本ALGOLコンパイラの作り出すオブジェクトプログラムの実行方式を評価するために、Wichmann⁽⁸⁾によって作成されたプログラムに従って、実行時間を測定した。Wichmannの報告によれば、24個のALGOLコンパイラによって生成されるオブジェクトプログラムの実行時間のデータが示されている。表4は、計算機の性能比較のデータが入手できたIBM 360/65のALGOLコンパイラに対する結果の比較を示すものである。本ALGOLコンパイラはHITAC 8450のもとで稼働させた。本表のデータは生のデータである。計算機の性能は、scient.mixで、IBM360/65が1.8 μ sであり、HITAC 8450が3.4 μ sである。また、FORTRANとの比較のため、同じプログラムをEDOS-MSO FORTRANで記述し、HITAC 8450で実行時間を測定した結果を表4に並記した。

表4において、変数 x , y , z , 配列 e は実数型であり、変数 k , l , m は整数型である。手続きの宣言は次のものである。

```
procedure P0; ;
procedure P1(x); value x; real x; ;
procedure P2(x,y); value x,y; real x,y; ;
procedure P3(x,y,z); value x,y,z; real x,y,z; ;
Loop Time は、for i := 1 step 1 until n do ; の 1
```

表4 実行時間の比較 EDOS-MSO ALGOL, IBM ALGOL, EDOS-MSO FORTRANに対する各ステートメントの実行時間が分かる。

ステートメント	システム	EDOS-MSO	IBM	EDOS-MSO
		ALGOL	ALGOL	FORTRAN
1. $x := 1.0$		3.4	5.6	3.7
2. $x := y$		3.4	4.2	3.9
3. $x := y + z$		7.5	4.4	6.9
4. $x := y / z$		14.5	7.3	15.9
5. $k := 1$		3.3	6.0	3.4
6. $k := 1 \times m$		8.8	22.6	8.8
7. $k := 1 \div m$		16.7	26.5	19.4
8. $x := 1$		21.5	31.6	21.1
9. $l := y$		28.0	31.5	22.0
10. $x := y \uparrow 2$		65.0	49.6	54.5
11. $e[l] := 1$		8.0	6.7	3.8
12. $l := e[l]$		8.4	5.1	3.4
13. begin real a ; end		2.8	327.	—
14. begin array a[1:1];end		90.0	649.	—
15. begin array a[1:1,1:1];end		112.0	685.	—
16. begin goto abcd;abcd:end		2.4	7.4	1.8
17. begin switch ss:=pq; goto ss[l]; pq:end		52.0	344.	—
18. P0		212.	373.	80.0
19. P1(x)		208.	500.	100.
20. P2(x,y)		296.	603.	119.
21. P3(x,y,z)		384.	726.	141.
22. loop time		16.0	22.0	9.0

注：単位 μ s

回の実行時間である。ここで、 n は整数型の適当な定数である。

IBMのALGOLと比較して、本ALGOLはオブジェクトプログラムの実行時間において、全体的に相当優れている。特に、手続き呼出しの実行時間に関する項目(18~21)においては、IBMのALGOLとの直接の比較で、約2倍の効率である。このことは、3.で述べた手続きの制御などの実行方式が十分評価できることを示している。

従来、ALGOL言語は、そのオブジェクトプログラムの実行時間において、FORTRAN言語より格段劣るものとされていた。今回のALGOLコンパイラの開発において、この点を改善するために、FORTRANに比べて、平均50%以内のオブジェクトプログラムの実行効率の低下に抑えることを目標にした。表4のデータにより比較してみると、平均的プログラムにおいて、約40%の実行効率の低下であることが分かり、当初の目標を十分達成することができた。

5 結 言

言語仕様において、JIS 7070の機能に、複素数データ、記号列データ、デバッグ機能を追加し、金物表現を3種類用意することによってユーザーの便宜を図った。また、FORTRANとの結合を可能にした。

オブジェクトプログラムの実行方式においては、動的記憶域の割当て、非局所変数の参照、手続きの制御、パラメータの受渡し、名札へのブランチの方式などにおいて、それぞれ効率の良い実行方式を得た。

入出力手続き群の実行方式を、三つのクラスに分けて、そ

れぞれの機能に応じた処理効率をもつようにした。

作成したALGOLコンパイラによって作り出されたオブジェクトプログラムの実行時間を測定した。同じプログラムをFORTRANに適用し、その実行時間との比較を行なった結果、平均的プログラムにおいて、本ALGOLは約40%程度の実行効率の低下にすぎないことが分かった。

終わりに、本ALGOLコンパイラの開発にあたって、御助言をいただいた立教大学助教授島内剛一氏に対し、深く感謝する次第である。

参考文献

- (1) D. Gries: Compiler Construction for Digital Computers, John Wiley and Sons, Inc. (1971)
- (2) B. Randell and L. T. Russell: ALGOL60 Implementation, Academic Press (1964)
- (3) 日本工業標準調査会: 電子計算機プログラム用言語ALGOL(水準7000), 日本規格協会 (昭和47年3月)
- (4) 日本工業標準調査会: 電子計算機プログラム用言語ALGOLの入出力(水準70), 日本規格協会 (昭和47年3月)
- (5) P. Naur(Ed), "Report on the Algorithmic Language ALGOL60", Comm. A. C. M., 3, 5, 216 (1960)
- (6) P. Naur(Ed), "Revised report on the algorithmic language ALGOL60", Comm. A. C. M., 6, 1, 1 (1963)
- (7) P. Z. Ingerman, "Thanks—Away of Compiling Procedure Statements with Some Comments on Procedure Declarations", Comm. A. C. M., 4, 1, 55 (1961)
- (8) B. A. Wichmann, "Basic Statement Times for ALGOL 60", National Physical Laboratory (1972)

論文抄録

クロスバスイッチの調整障害に関する改善

日立製作所 渡辺順平・笠原 司

電子通信学会論文誌 56A—11, 693 (昭48-11)

公衆通信用に使用されるクロスバ形自動交換機は、日本電信電話公社の仕様により数社で製造される。この交換機は多くの架により構成されているので、架ごとに製造し、電話局に搬入する。電話局では建設工事が行なわれ、試験により交換機の機能、品質を保証したのち、サービスに入る。建設工事期間は数ヶ月ないし1年程度にわたる場合が多い。ゆえに建設工事段階の障害率は、製造段階の不良率又は稼動初期の障害率との間に相関があると考えられる。そこで製造段階と稼動段階を対象として行なわれてきた交換機の品質の分析に対して、建設工事段階を加えて、分析を行なった。

著者らが所属する会社の製品は、1966年以前に実施した部品の取替えを要する障害の改善に伴い、交換機の通話路を構成するクロスバスイッチ(以下、XBSWと略す)の機械的な調整が変化することによる障害(以下、調整障害と呼ぶ)の占有率が増加し、

1966年ごろより建設工事段階が目立つ問題となった。そこでこの障害について、調査、分析を行なった。

分析の結果、製造、建設工事、稼動初期の3段階の不良率又は障害率の相互間について、次の量的関係を得た。

- (1) 製造段階の不良率と建設工事段階の障害率との関係
- (2) 建設工事段階の障害率と稼動初期段階の障害率との関係
- (3) 製造段階の不良率と稼動初期段階の障害率との関係

障害を調査した結果、障害は主としてフィングの調整の変化により生ずることを確認した。

以上の障害の分析、調査に基づき、製造段階の不良率を低減して、建設工事又は稼動初期段階の障害を減少させることとし、フィングの調整を対象に改善を実施した。

改善にあたっては、製造段階の不良率と

建設段階の障害率の量的関係に基づいて、建設工事段階の障害率を基準として、製造段階の目標不良率を定めた。更に、フィングの調整変化は、製造段階から建設工事段階までの架の移動によるものであることを確認した。そこで、この変化分を考慮して、製造上の規格値を変更した。

上記の改善により、製造段階の不良率、建設工事段階の障害率はいずれも目標値を達成し、成果を収め、稼動初期段階の障害率も減少した。

すなわち、交換機の素子の障害について、XBSWを事例として、製造、建設工事、稼動初期の3段階の量的関係を求め、この関係から建設工事段階の障害に基づいた目標値を定め、製造上の改善を行なうことが、製造、建設工事、稼動初期の3段階の品質向上に有効であることを明らかとした。