ソフトウェア工学 40－7
（１９８５．２．７）

ソフトウェア移植用 ＦＯＲＴＲＡＮ翻訳ツールの半自動生成

西岡健自，平田陽一郎
（横河北辰電機株式会社）

１．はじめに

ソフトウェアがハードウェアの付随物から一個の独立した商品としての地位を固め始め，高度で多様なハードーウェアが安価に入手できるようになるにつれ，ソフトウェアの移植性が注目され始めた。ソフトウェアの移植によりユーザは特定の機種上でしかできなかった仕事を手持ちの，あるいは，自分の好みのコンピュータ上で同様に行うことができ，メーカは別々の機種上で同様のソフトウェアを新規に開発するのに較べて，膨大な開発工数を節約することができる。

こうして，ユーザとメーカの利害関係一致のもとに移植性の高いソフトウェアが追究されるに及んでいるが，研究の場では移植可能なソフトウェアを作り出す研究に較べ，既存ソフトウェアの移植の研究は少なく，いくつかの事例が紹介されているに過ぎない。

ＦＯＲＴＲＡＮのような高級言語ですら処理系による相違は大きく，その移植には多大の工数を要するにもかかわらず，このように取り組みが十分と言えないのは次の理由によると思われる。

1) ハード，ＯＳ，言語仕様(方言)，内部表現等の相違を越えて既存ソフトの移植を完全に自動化することがむずかしい。
2) 自動化ツールを開発できたとしても，そのツールは処理系や言語に依存するため，多様な移植の要求に合わせてそれぞれのツールを用意する必要がある。

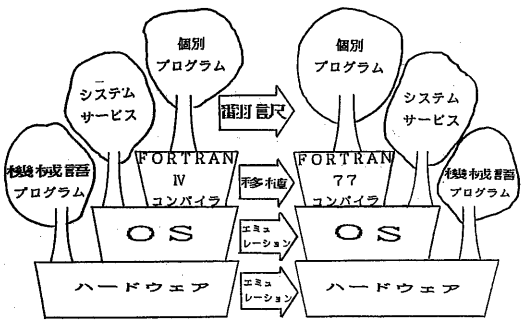
しかし，移植への配慮なしに蓄積された過去の莫大なソフトウェアを新しい環境のもとで再利用したいという要求は強く，我々は次の方針でこの既存ソフトウェア移植の問題に取り組んだ。

1) 可能な限りの移植の自動化を図り，その支援ツールを開発する。
2) ツール生成系を用意して移植ツールの開発を効率化し，多様な移植の要求に対応できるようにする。

この結果，約17万ステップのＦＯＲＴＲＡＮプログラムを約７人月で移植できる自動移植システムを構築した。本稿では特にツール生成系に重点を置いて自動移植の概要を報告する。

２．自動移植システムの概要

既存ソフトウェアの移植にはソース側(移植対象)の処理環境をターゲット側(移植先)にそっくりエミュレートする方法から，システム・サービスのエミュレーション，コンパイラの移植，個々のソース・ファイルの処理系に依存する方言部分のターゲット側言語仕様への翻訳など，(図１)のように多くの方式レベルが考えられる。しかし，処理系のエミュレーションでは言語によらずソース側で開発された全プログラムを移植できる代り，移植されたプログラムのパフォーマンス低下が予想され，エミュレータ自身の開発工数も膨大となる。また，コンパイラの移植ではその言語で記述されたプログラムの移植性は高まるが，コンパイラ自身の移植は一般に容易ではない。そこで，我々は個々のプログラムの翻訳と，システム・サービスのエミュレーションを組合せる方式をとった。具体的移植対象としては十万ステップ以上の大規模ソフ

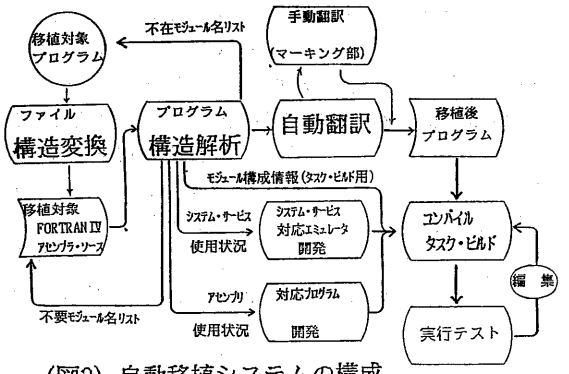


(図1) 既存ソフトウェア移植方式の階層

トウェア，言語としては拡張されたＦＯＲＴＲＡＮⅣをＦＯＲＴＲＡＮ77の拡張版へ移すケースを設定した。

この規模の移植ではソース側処理系のプログラム・ファイルをターゲット側のファイル構造へ自動変換する機能が必要となり，完備したドキュメントが実際問題として期待できないことから，ソース・テキストからタスクのモジュール構成を解析したり，エミュレータを開発すべきソース側システム・サービスの使用状況を抽出するプログラム構造解析ツールも必要となる。また，入力テキストの数％程度の翻訳所要箇所を見つけだすのに全テキストを走査する必要があるため，人手による翻訳は現実的にほとんど不可能となるのだが，この意味で，自動翻訳しきれない部分についてもそのテキスト上の位置を示し，かつ，自動翻訳後のオペレーション・サービスとして，その部分の手動翻訳の便宜も与えておく必要がある。

このような検討のもとに構築したのが(図２)の自動移植システムであり，ユーザはメニュー方式のマンマシン・インタフェースにより標準的流れに沿って移植作業を進めることができる。



(図2) 自動移植システムの構成

この移植システムの構成要素のうちマンマシン・インタフェースと，自動翻訳ツールに対して生成系を用意した。マンマシン・インタフェースのメニュー，及び，各メニュー対応処理の会話部分などはメニュー・マネジャと呼ばれるツールによって会話的に構築される。翻訳ツールはトランスレータ・ジェネレータと呼ばれる生成ツールによって半自動生成されるが，その詳細を次節に述べる。

３．翻訳ツールの自動生成

3.1 移植における翻訳とは

(図３)はソース側の拡張ＦＯＲＴＲＡＮⅣをターゲット側の拡張ＦＯＲＴＲＡＮ77へ翻訳する例である。このように，同じＦＯＲＴＲＡＮでありながら２つの言語には処理系による差異があり，本例の場合は次のように差異は36項目に及ぶ。

(ｉ) 表現の拡張による相違点(18点)
(図３－１，４)
(ii) 表現は同じで意味の異なるもの
(５点)(図３－２，３)
(iii) 表現が異なり意味の同じもの(６点)
(iv) 内部表現の相違に基くもの(１点)
(v) ソース側固有の偽命令(４点)
(vi) その他(２点)

このうち(図３－４)はソース側でASSIGNされたシンボルをサブルーチンの引数として使用できるが，ターゲット側では使用できないことによる相違である。この場合の翻訳は一律に決められないため，翻訳ツールは位置のマーキングのみ行う。このように翻訳ツールは自動翻訳が不可能か，あるいは，困難な場合には処理を手動翻訳に委ねるという柔軟な姿勢をとる。ただし，本例のマーキング対象となるのは36の差異のうち３項目のみで，実際のテキスト中に出現する頻度も十分小さい。

上記の差異の分類とは別に，翻訳処理の観点から次のような分類も可能である。

(ｉ) 文脈に依存しないもの(20点)
(ii) テキスト・レベルで文脈に依存するもの(11点)(図３－１，４)
(iii) 実行時に文脈に依存するもの(５点)
(図３－２，３)

文脈に依存するものは(図３)のように，ステートメントの前後の状況によって翻訳を変える必要のあるものである。特に(iii)タイプに属す(図３－２，３)の例では実行時にとる変数の値により意味が変わるため，テキスト上の翻訳

ではこの変化に追随できない。この場合には図の例のような冗長な翻訳によって対処している。

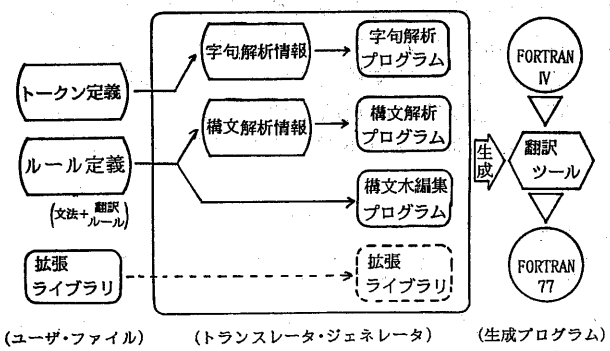
（ソース・テキスト）（ターゲット・テキスト）

	拡張 FORTRAN Ⅳ	拡張 FORTRAN 77

1	IF(A.EQ.B)100,200
	条件 { A,B:変数 / A:配列}
		IF(A.EQ.B)GOTO 100
		GOTO 200
		IF(A(1).EQ.B)GOTO 100
		GOTO 200

2	DO 100 I=L,M 条件 { L,M:整数 / L:整 M:実}	DO 100 I=L,MAX(L,M) DO 100 I=L,MAX(L,INT(M)) ◆ ◆
3	GOTO(10,20,30),N	GOTO(10,20,30) MIN(MAX(1,N),3)
4	CALL SUB(L) 条件 { L:ASSIGN変数 / L:通常変数}	CALL SUB(L <-- MARK<10> !*) CALL SUB(L)

(図3) 翻訳例



(図4) トランスレータ・ジェネレータの構成

3.2 トランスレータ・ジェネレータの概要

トランスレータ・ジェネレータから生成される翻訳ツールは構文解析，構文木編集方式をとっている。これは翻訳の対象となる言語間の差異が多様でマクロプロセッサなどの単純なパターン・マッチングでは処理しきれず，yacc，lexを用いる構文解析による階層的なパターン検出が必要なこと，及び，ステートメント単位の構文解析の結果により翻訳の形式が左右されることがあり，構文木として構文解析情報を保存しその後に翻訳処理を行う必要のあること，などによっている。この翻訳ツールを生成するトランスレータ・ジェネレータの概要を(図4)に掲げる。この図のように，トランスレータ・ジェネレータは入力されるトークン定義，ルール定義(文法＋翻訳ルール)，及び，必要に応じてユーザの用意する拡張ライブラリから翻訳ツールを生成する。

3.3 ルール定義言語の特徴

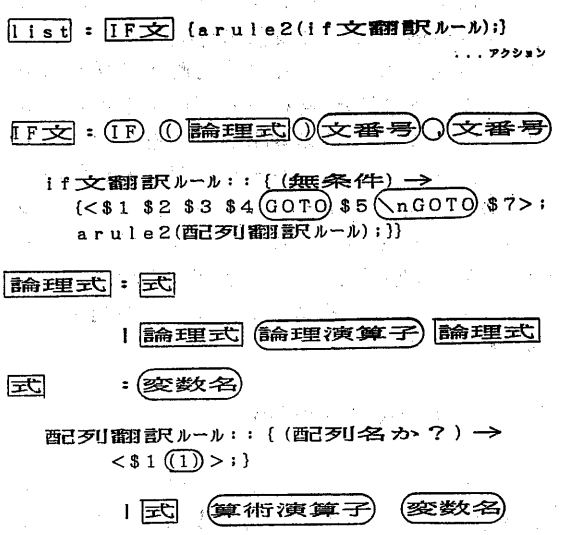
トランスレータ・ジェネレータにはトークン定義言語とルール定義言語が提供される。ここでは，ルール定義言語に関する特徴をとりあげる。((図5)にルール定義の記述様式例，(図6)に実際の記述例をあげる。)

(ｉ)文法と翻訳ルールの対応の視覚化

文法と翻訳ルールは必ずしも一意に決まらない。相互の関連に注目しながら全体として最もシンプルな形にするのが理想である。このようなプログラミングを可能とするには，まず，文法と翻訳ルールを並記でき，かつ，その対応が取り易い言語が必要になる。ルール定義言語では次のように非終端シンボルと翻訳ルールを対応づけることにより，文法と翻訳ルール対応の視覚化を実現した。

```
IF文 : (IF) ( 論理式 ) (文番号) , (文番号)
                                  ・・・ 文法
    < $1 $2 $3 $4 (GOTO) $5 (\nGOTO) $7 >
                                  ・・・ 翻訳ルール
```

これは(図3)のＩＦ文の翻訳に対応するルール定義の一部だが，if文翻訳ルールは非終端シンボル ＩＦ文 のシンボル並びにトークン (GOTO) を挿入し，かつ，(,) を (\nGOTO) に置き換えることを表している。また，"＄n"は文法のｎ番目のシンボルを表わしている。



(図5) ＩＦ文のルール定義構成

(ii)意味処理

文脈に依存する翻訳では条件によって翻訳の形式が変化する。この変化に柔軟に対応するため，次のように翻訳ルール中に条件を記述でき，この条件が真のときのみ翻訳処理が実行される。

(配列名か？) ーー> < 変数名 (1) >;

これも(図３)のＩＦ文の翻訳に対応するルール定義の一部だが，変数名が配列名に一致した時のみ，変数名のあとに"(１)"を付加することを意味する。

なお，この条件式にはＣ言語の条件式が記述できる。

(iii)翻訳関数の導入

実際の翻訳では入出力テキストの文法はほとんど同等なので上記の翻訳ルール記述方式でおおよそ間に合うが，非終端シンボルの構成シンボルの挿入，削除，置換のみでは表わしきれない場合もある。これに対応するために(図６)のようにＣ言語で記述された関数を使用することも許されており，特殊な翻訳にも対応できるようになっている。

(iv)翻訳ルールの活性化方式

翻訳ルールは構文解析の途中，文法に付随するアクション部の活性化処理によって登録され，ステートメント単位の構文解析終了後起動される。この方式により，属するステートメントによって翻訳形態の異なる非終端シンボルに対応して，複数の翻訳ルールを定義しておき，ステートメント種類の決定した段階で所望の翻訳ルールを活性化するといった翻訳処理が実現できる。

(図５)は(図３)のＩＦ文の翻訳に対応するルール定義の完全な形だが，スタート・シンボル"list"のアクション部に記述されたarule2関数によってif文翻訳ルールが活性化され，実際の翻訳は構文解析終了後，ステートメント単位の構文木上の"ＩＦ文"非終端シンボルの構成

文	ルール定義
GOTO	sotos [GOTO '(' ars ')' ',' arithexpr '\n']{arule2(sotoconv);}. sotoconv ::{(TRUE) -->{<$1 $2 $3 $4 $5 "MIN(MAX(1," $6 "),"argcnt($3,number) ")\n">;ulos(UGTO);}}. (< >内の翻訳関数 argcnt はアーギュメントの数をカウントする。図３－３参照)
ASSIGN	assigns [ASSIGN number IDENTIFIER identifier '\n']{arule1(recsym);}. recsym ::{(TRUE) --> spush($4,_ASSI);}. (翻訳ルール recsym はパス１で起動され、翻訳関数 spush は ASSIGN変数を収集する。図３－４)
	calls [CALL identifier '(' ars ')' '\n']{arule2(assichk1,assichk2);}. ars [arithexpr]. assichk1 ::{(exprchk($1,_ASSI)) --> {<$1>; mark(); ulos(UASS);}}. [ars ',' arithexpr]. assichk2 ::{(exprchk($3,_ASSI)) --> {<$1 $2 $3>; mark(); ulos(UASS);}}. (翻訳関数 mark はマーキング処理を行う。図３－４参照)

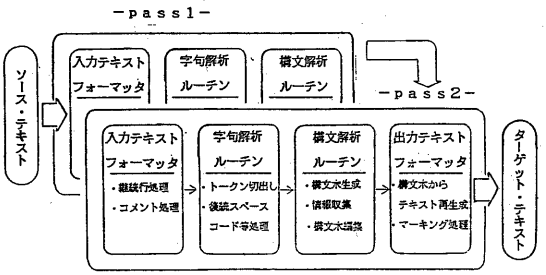
(図6) 実際のルール定義例

要素を編集するという形で実行される。

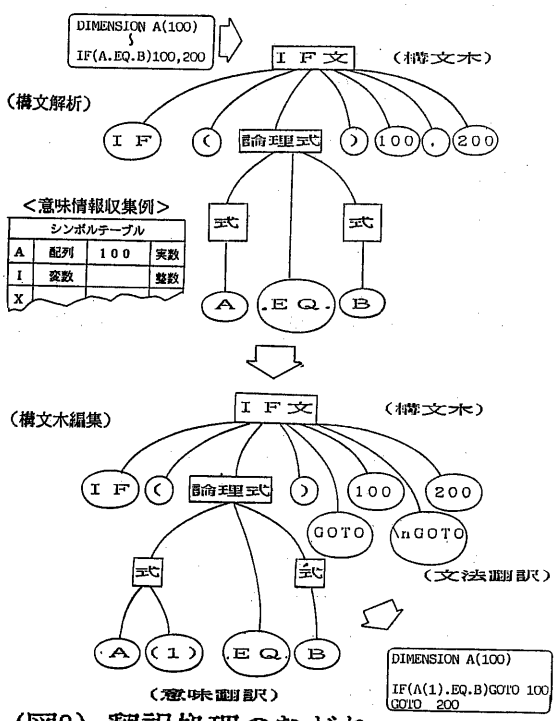
ここで注目されたいのは，翻訳ルール中の翻訳関数としても翻訳ルールの活性化処理を記述できることで，これによって，文法と関連する柔軟な翻訳処理の記述が可能となっている。((図3) if文翻訳ルール参照。)

3.4 生成翻訳ツールの特徴

トランスレータ・ジェネレータによって生成される翻訳ツールは(図7)のような構成をとる。また，(図3)のＩＦ文の翻訳処理は(図8)



(図7) 翻訳ツールの構成



(図8) 翻訳処理のながれ

のように進行する。以下，この翻訳ツールの特徴について述べる。

(ｉ)readabilityの保存

プログラム言語は固有の継続行，コメントなどの表現法を持っているが，これらは字句解析・構文解析で扱いにくい。そこで，(図7)のように生成される翻訳ツールを階層化し，字句解析の前処理として入力テキスト整形フェーズを導入した。ここでは，継続行，コメントなどのテキストのreadability上の形式を除去して上位フェーズに情報を送出し，各解析処理を単純化する。また，除去された情報は出力テキスト・フォーマットにより，構文木から翻訳済テキストを生成する際に再び付加される。これによって，翻訳後のテキストでも入力テキストのreadabilityは保存されることになる。なお，これら整形ルーチンは言語に依存するため，トランスレータ生成の際ユーザが用意する必要がある。(標準ルーチンは用意されている。)

(ii)エスケープ・コード処理

ＦＯＲＴＲＡＮにおけるスペース・コードなど整形用の区切記号は構文解析を煩雑にするので，字句解析フェーズで直前のトークンの一部として取り込み，構文解析側に現われないようにする。このような整形用区切記号をエスケープ・コードと呼び，翻訳ツール生成時トークン定義として次のように記述しておく。

```
%ESC  " (スペース) " | " (tab) " .
   ... スペース,タブをエスケープ・コードとするトークン定義例
```

(iii)構文木の表示とログ

翻訳ツールは次のようなオペレーションで作動するＣ言語プログラムである。

```
XX  [-tn] (入力file名) (出力file名)
```

ここで，ＸＸは翻訳ツール名を表わし，-tnはオプション・スイッチでデバック用に構文木を表示する時に使用する。また,翻訳を受けたテキスト上の位置などを記録するログ・ファイルも生成され，翻訳の統計処理の便宜も図られている。

4．実績と評価

トランスレータ・ジェネレータによって生成された翻訳ツールによる移植作業の実績を(図9)にまとめる。手動介入を受けた100個のモジュールはコンパイル段階，あるいは，実行段階でエラーを発生したものだがその要因は次のようなものである。

（i） オーバレイ形式のプログラムを仮想メモリ上で実行するため全体リンクしたことに伴う不具合。(78%)
（ii） ソース，ターゲットでのハード上の相違(15%)
（iii） ソース側の虫。(7%)

次に，トランスレータ・ジェネレータについては，これを使用しない場合に較べ1/3程度の工数で翻訳ツールを開発することができ，所期の効果を得ることができた。

項　目	データ	備考
移植モジュール数	1230ケ	
ステップ総数	17万行	
自動翻訳時間	3h以下	
手動介入モジュール	100ケ	エディタを用いた手動のプログラム修正
平均処理時間	3秒/100行	
平均翻訳率	4．4%	変更を受ける文の割合
マーキング個所	0	自動翻訳不能個所

(図9) 移植実績データ

5．トランスレータ・ジェネータの応用

移植用の翻訳ツールを生成するツールとしてトランスレータ・ジェネレータを考えてきたが，テキスト・レベルの小規模の変換を行うトランスレータを生成するツールとして捕えると，トランスレータ・ジェネレータの応用範囲は広いものになる。

特に，ソフトウェア開発ツールにはプリティ・プリンタ，構造化ＦＯＲＴＲＡＮプリプロセッサ，プログラム構造解析ツール，ドキュメント生成ツールなどテキスト変換を主体とするものが多く，これらはトランスレータ・ジェネレータによって容易に生成できると考えられる。

この意味で，トランスレータ・ジェネレータはソフトウェアの統合化開発環境を構築する上で強力な武器となる可能性があり，今後，この方面での検討を進める予定である。

6．あとがき

移植性を配慮されることなく開発された過去のソフトウェアの膨大な蓄積を再利用すべく，既存ソフトウェアの自動移植システムを構築し，実際に17万ステップのプログラムの移植に成功した。移植方式は個々のソース・プログラムをターゲット側処理系にテキスト・レベルで適合するよう翻訳する手法をとったが，言語や処理系に依存する翻訳ツールを移植要求に合せて効率的に開発できるよう，その生成系であるトランスレータ・ジェネレータを開発し，所期の成果を得た。

更に，トランスレータ・ジェネレータをソフトウェア開発用ツール開発のための支援ツールとして応用すべく検討を進めている。

[参考文献]

[1] S.C. Johnson "Yacc: Yet Another Compiler-Compiler" (ＵＮＩＸマニアル)
[2] M.E.Lesk,E.Schmidt "Lex - A Lexical Analyzer Generator" (ＵＮＩＸマニアル)
[3] 藤田，葛山，川原 "プログラムの移植について" 情報処理 Vol.21 No.11 P1128~1135 (1980)
[4] 平田，西岡 "プログラム移植ツールと標準化" 第４回ソフトウェア生産における品質管理シンポジウム予稿集 P67~74 (1984)
[5] 西岡，平田 "ソフトウェア・ツールを自動生成するトランスレータ・ジェネレータの開発" 情報処理学会第30回全国大会講演論文集 (1985) <予定>
[6] 平田，西岡 "トランスレータ・ジェネレータによるＦＯＲＴＲＡＮプログラム移植ツールの試作" (同上) <予定>