2K−7

プログラムの移植条件試験法

橋本　辰範　　堀田　博文
NTTソフトウェア研究所

1 はじめに

ソフトウェアの生産性向上、マルチベンダ化への対応のために、プログラムの移植がますます重要になっている。一般に、移植対象プログラムの開発者以外が移植を行う場合には、

①改造を必要とする箇所の判断
②移植後の試験

に大きな工数が費やされる。そこで、本稿では

①を効率化するための**移植条件試験**
②を効率化するための**移植後確認試験**

の方法を提案する。

2 移植作業のモデル

環境Epで正常に動作しているプログラムPpを環境Eqに移植する場合、作業は通常次の手順で行われる(図1)。

(1)Ppの事前調査:改造箇所と改造方法の明確化
(2)Ppを改造しPqを作成(改造不要ならばPq=Pp)
(3)PqをEq上で走行確認

これらの作業の中で、次のことを意識した試験が必要あるいは有効である。

[**移植条件試験**] (1)の事前調査において、改造に関する情報(改造を必要とする箇所とその理由)を得るための試験を実施する。

[**移植後確認試験**] (3)の走行確認において移植によりプログラム動作が変化する可能性の高い部分を重点的に試験する。

本稿では、Ppの移植における改造の必要性は環境Ep、Eq間の差異のみに起因すると仮定する。ここで、Ppと環境とのインタフェースは、コンパイラ及びシステムコールとする。

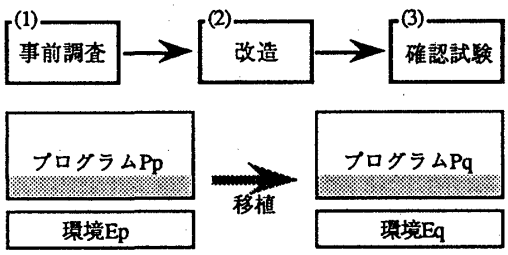


図1　移植作業のモデル

Testing the Conditions of Program Portability
Tatsunori HASHIMOTO, Hirofumi HOTTA
NTT Software Laboratories

3 移植試験方式

3.1 移植条件試験

移植時に改造を必要とする箇所を発見する方法の概要を示す。手順全般にわたって、Ppのフローグラフのノードを彩色する。色の意味は、次に示すものである。

白：移植の観点で、必ず正しい動作をする
黒：移植の観点で、必ず誤った動作をする
灰：移植後の動作が正しいか否か分からない

[step1] Ppのフローグラフ Gf(Pp)を作成する。グラフ上の全ノードを灰色に彩色する。□

[step2] プログラムの静的解析により、移植時に100%安全な地点と100%問題を起こす地点を認識し、Gf(Pp)の彩色を次のように変更したグラフGm(Pp)を作成する。

灰→白：100%安全な地点に対応するノード
灰→黒：100%危険な地点に対応するノード　□

[step3] Ep、Eq間の差異を調査し、両者で動作が同一の地点に対応するGm(Pp)上のノードの色を白に変更したグラフGm(Pp,Ep,Eq)を作成する。□

[step4] PpのEp上での動作を試験データTi (i=1,2,...,n : nは試験データ個数)を用いて試験する際に、Gm(Pp)上の灰色に彩色されたノードに対応する地点に移植性チェック用コードを埋め込む(コンパイラあるいはプログラム変換による)。Tiを用いた実行中のチェックによりTiに対してはどんな環境でも100%安全であることが判明した地点に対するGm(Pp)上のノードの色を白に変更したグラフGmi(Pp,Ti,Ep) を各iに対し作成する。この試験時にEqが既知の場合には、Gm(Pp)の代りにGm(Pp,Ep,Eq)を用いてGmi(Pp,Ti,Ep,Eq)を作成し、[step5]を省略することができる。□

[step5] step4のチェック結果とEqに関する情報を用いて、各iに対し、Gmi(Pp,Ti,Ep) 上の灰色に彩色されたノードを次のように彩色し直したグラフGmi(Pp,Ti,Ep,Eq) (i=1,2,...,n)を作成する。

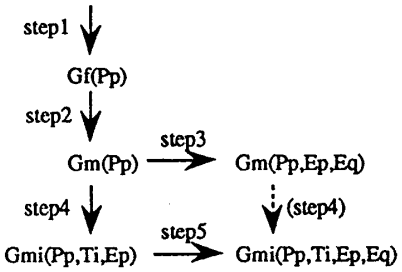
灰→黒：Eq上とEp上の動作が必ず異なる部分に対応するノード
灰→白：Eq上とEp上の動作が必ず同一である部分に対応するノード　□

T=(T1,T2,...,Tn)が試験として完全な集合である場合は、次のように改造箇所を判断できる。

Gmi(Pp,Ti,Ep,Eq) (i=1,2,...,n)内のノードが

(case1) 白の箇所は改造不要。
(case2) 黒の箇所は改造必須。
(case3) 灰色の箇所は改造の必要性を検討要。

本方式の効果は、改造箇所の特定、開発時に作成された試験データの有効な再利用にある。なお、Gm(Pp,Ep,Eq)上



Gf(Pp)	Ppのフローグラフ
Gm(Pp)	Pp の本質的に安全な部分と危険な部分を表現
Gm(Pp,Ep,Eq)	Ep,Eq間でPpの動作が同一の部分と異なる部分を表現
Gmi(Pp,Ti,Eq)	Ti入力時のPpの動作が処理系によりEp上とは異なる部分を表現
Gmi(Pp,Ti,Ep,Eq)	Ti入力時のPpの動作がEp,Eq上で異なる部分を表現

図2　移植性表現グラフの推移

で灰色であるにも関わらずどのGmi(Pp,Ti,Ep,Eq)においても白であるノードがある場合には、Tが完全でない可能性があるため、そのノードがあらゆる場合に問題無いことをTiとは別の手段で確認することが望ましい。このとき、Gmi(Pp,Ti,Ep,Eq)の被覆率を試験進捗尺度として用いることができる。

3.2 移植後確認試験

移植後の試験では、通常次の問題がある。

(1)開発時に使用したすべての試験データを用いて移植後の試験を実施するのは、無駄が多い。

(2)適当な順で試験データを選択すると、問題点検出・改造の後の再試験が多く必要となる。

本節では、これらの問題を解決し、移植後の確認試験を効率よく実施するための試験データの選択方法を示す。

3.1の移植条件試験により改造が必要と判断された場合の改造・移植後の試験手順を優先度の高いものから順に示す。

[優先度1]　移植で改造した部分あるいは新たに作成した部分を狙い撃ちする試験データを新規作成し試験。

[優先度2]　Gmi(Pp,Ti,Ep,Eq)が黒に彩色されたノードを持つ場合、Tiを用いて試験。

[優先度3]　Gmi(Pp,Ti,Ep,Eq)が灰色に彩色されたノードを持つ場合、Tiを用いて試験。

[優先度4]　製品確認試験として必要と思われる全試験を実施。

T=(T1,T2,...,Tn)が試験として完全な集合であり、3.1の手順が正確に実施されている場合は、論理的には[優先度3]までの試験が終了していれば問題無い。しかし、実際には、品質保証のために[優先度4]の試験も実施するのが通常である。

4 Cを対象にしたケーススタディ

4.1 ISO-Cプログラムの検査方法例

```
例1)  int a,b,c;          例2)  signal(KILL,SIG_IGN);
      a = b - c;

例3)  short x;            例4)  size_t size;
      long y;                     void *p;
      x = y;                      p = malloc(size);
```

(1) [step2] 静的に白黒を決める例

[言語仕様]　式の評価順序は処理系依存であるが、例1では式の演算項がすべて変数であるためどの処理系でも同一の動作となり、白である。

[ライブラリ]　シグナルの種類は6種が既定で他は処理系依存である。例2のシグナル名KILLは字面上は処理系依存であるが、既定のシグナルと同一の値を持つようマクロ展開されるか否かを解析することにより白(既定のシグナル)、黒(処理系独自のシグナル)を決めることができる。

(2) [step4] 実行時に白黒を決める例

[言語仕様]　shortは一般にlongよりサイズは小さいが、例3で実行時にオーバフローするか否かは値によるため、実行時に値の記録をとる。

[ライブラリ]　サイズ0をパラメタとするmalloc呼出しの動作は処理系依存であるが、例4がそれに当たるか否かは実行時でないと分からない。そこで、実行時に、malloc呼出し直前に変数sizeの値が0であるか否かをチェックする。

4.2 Cプログラムに対する実施例

簡単な実用Cプログラムに対し本稿で提案した手順を机上でシミュレートした。その内容と結果を以下に示す。

プログラム情報

プログラム内容	：ファイル内文字変換
プログラム規模	：169行
記述言語	：C(+ISO-Cライブラリ)
移植内容	：UNIX→他OS
開発時試験データ数	：13個

ノードの彩色推移

	step1	step2	step3	step4
灰	80	37	28	14
白	0	43	51	65
黒	0	0	1	1

移植後必要な試験データ

開発時のもの流用	：4個
新たに作成	：1個

5 おわりに

本稿では、移植条件試験、移植後確認試験の効率化について述べた。今後、本手法を実際の移植に適用して評価を行う。