記号実行によるプログラム改造支援技術

Supporting Technology for Modification of Existing Software Using Symbolic Execution

酒井　政裕　　岩政　幹人
■SAKAI Masahiro　■IWAMASA Mikito

大規模なインフラシステム向けソフトウェアは，システムの運用開始後も長期の間，機能の追加や機能の一部変更を行った派生システムが開発されることが多い。このような派生開発では，ソフトウェアの再設計を行うため，以前に設計されたソフトウェアの設計情報が重要であるが，その設計情報が十分存在しないことが，信頼性の高いシステム開発の障害となっていた。既存のソフトウェアを人手で解析し設計情報を作成することも考えられるが，これには多大な労力が必要となる。

そこで，この設計情報をその記述手段の一種である決定表形成，記号実行の技術を用いてソフトウェアから自動的に抽出する技術を開発した。この技術を実現するプロトタイプを作成して実験した結果，ソフトウェアの再利用及び再設計に寄与できるという評価が得られた。

In the development of software for large-scale infrastructure systems, the addition of functions over a long period and modification of functions due to changes in circumstances frequently occur during the operation phase. In such cases, the specification and design information of the target systems are required for redesign or reuse of the software. However, these resources are sometimes missing or have not been updated to the latest version during the process of long-term modification, creating an obstacle to the development of high-reliability systems. Furthermore, enormous efforts are required to re-create specification and design information by manually analyzing existing software.

To overcome these problems, Toshiba has developed a technology to automatically extract decision tables, which are a form of design information, from existing software by employing a symbolic execution technology. Experiments on a prototype using actual source codes have verified that this technology can support the modification and reuse of software.

1　まえがき

現在のソフトウェア開発では，従来製品から他機種やシリーズ機種へソフトウェアを再利用することが多い。ソフトウェア開発プロジェクトの50％以上が，差分開発，派生開発，改修開発，あるいは保守開発のプロジェクトとされている[1]。特に，社会インフラを支えるシステムに関わるソフトウェア開発では，既存ソフトウェアの再利用，すなわちソースコードの流用率が非常に高い。

したがって，差分開発や派生開発で効率的な開発を進めるには，再利用時に，新しい仕様に対する変更内容を洗い出し，既存ソフトウェアをそのまま流用可能な部分と変更が必要な部分とに切り分けることが，最初の重要なステップとなる。しかし，既存ソフトウェアを開発したプロジェクトが他社である場合や，自社のものでも設計書が整備されていなかったり保守されていなかったり，更に当時の設計者が退職などでいない場合も多く，処理内容を熟知していない設計者にとって，分析に多大な工数が掛かることが課題となっている。例えば，ソースコードを人手で解析し設計情報を作成し直すいわゆるリバースエンジニアリングは，多大な労力が必要になる。

東芝はこうした課題を解決するため，プログラムから仕様を逆推定する技術（仕様発掘：specification mining）の研究開発に以前から取り組んでいる[2]。

その取組みの一環として，記号実行[3]と呼ばれるプログラム解析技術を用いて，個々の関数の外部仕様を，プログラムのソースコードから，設計情報の記述手段の一種である決定表形式で，自動的に抽出する技術を開発した[4]。この技術を用いることで，既存プログラムに埋もれている条件判断ロジックを人間にわかりやすい表形式で自動的に出力できる。駅務機器などインフラシステム向けのソフトウェアの一部は，メンテナンスされながら運賃計算処理に代表されるように非常に複雑な条件判断[5]が更新されており，この技術によってソースコード再利用時の労力削減が期待できる。

ここでは，この技術の概要を述べるとともに，C言語向けのプロトタイプの開発とそれを使って実際の製品のソースコードを対象に評価した結果及び今後の取組みについて述べる。

2　決定表とは

決定表は，入力変数で表現された条件式を行とし，成立する条件の組合せとそれに応じた結果やアクションを列として表したもので，仕様書の補助的な記述手段としてよく使われる

表1．仮想施設入場料金の決定表例
Example of decision table to indicate admission fees of virtual facility

組合せ条件と結果		条件成立（Y）及び不成立（N）と得られる結果（X）			
条件	6歳未満	Y	N	N	N
	65歳以上	N	Y	N	N
	それ以外の年齢	N	N	Y	Y
	県内在住	−	−	Y	N
結果	無料	X	X	−	−
	1,000円	−	−	X	−
	2,000円	−	−	−	X

＊−は，真偽を問わない，あるいは結果なし

ものの一つである。

例として，仮想の施設の入場料金を決定表で表現したものを**表1**に示す。表の上半分は条件式の成立及び不成立を表しており，下半分は条件に応じて，どの結果やアクションが選択されるかを表している。各列は，Yが記された条件が成り立ち，Nが記された条件が成り立たないとき，Xが記された結果となることを意味する。“−”は真偽を問わないことを意味する。例えばこの表の右端の列からは，“それ以外の年齢”で県内在住でない場合の料金は2,000円であるという情報を読み取ることができる。

複雑な条件分岐も，決定表として表現すると可読性が向上し，漏れ抜けの判定や修正の影響について分析が容易になることから，要求仕様の表現やテストデータ作成に広く用いられている。

3　記号実行による決定表抽出

プログラムにおける条件判断の情報を決定表の形で出力するために，この技術では記号実行を用いる。

記号実行とは，通常の具体的な値を変数に代入しながらプログラムを実行する代わりに，変数とこれに対する制約条件の対を用いて，制約条件を更新しながらプログラムの実行をシミュレーションする技術である。記号実行により，個別の値ではなく変数がとりうる全ての値の範囲を，制約条件を通じて一気に取り扱うことができる。

3.1　記号状態

通常の実行では，プログラムの状態とはプログラム中の変数やメモリに対して実際の値を割り当てたものである。それに対して，記号実行ではプログラムの入力変数やグローバル変数の値を記号として表現し，プログラムの状態を，変数に割り当てられる値と変数間に成立する条件式の組合せで表現する（**図1**）。この組合せを記号状態と呼ぶ。

また，この条件式はその分岐のパスを通過する際に成り立つ条件を表したもので，条件分岐を実行するごとに条件が追加されていく。そのため，この条件式のことをパス条件と呼ぶ。

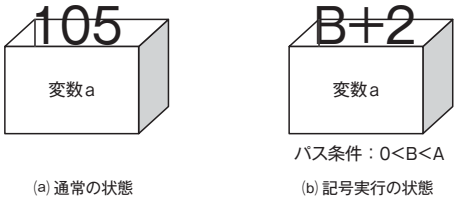

図1．通常の実行における状態と記号実行における状態 — 具体的な値の代わりに変数と条件を状態として用いる。
Concrete and symbolic states

3.2　記号実行

記号実行は，記号状態を書き換えながらプログラムの実行をシミュレートする。具体例として，**図2**に示すプログラム（関数）を用いて考え方を示す。

この関数にはa，bという二つの引数（入力）が存在するため，これらの初期値をそれぞれ変数A，Bによって表し，初期状態はa=A，b=Bという割当てと，恒真を表すtrueというパス条件から成る。

2行目の条件分岐では，条件式a＞bの値を表す式はA＞Bとなり，これが成り立つ場合と成り立たない場合の2通りに実行が分岐する。分岐後はそれぞれ次の状態となる。

割当て: a=A, b=B; パス条件: A>B　　(1)

割当て: a=A, b=B; パス条件: ¬(A>B)　　(2)

ここで，¬は条件式の否定を表している。

更に前者の状態で，c=a−b;という代入文を実行すると，次の状態が得られる。

割当て: a=A, b=B, c=A−B; パス条件: A>B　　(3)

分岐しながら実行が進行するようすを**図3**に示す。ここでは記号状態をノードとする木で実行のようすを表しており，根

```
00: Table f (int a, int b) {
01:   int c;
02:   if (a>b) c=a−b;
03:   else c=b−a;
04:   c+=10;
05:   if (a==b)
06:     return TABLE1;
07:   else if (c<20)
08:     return TABLE2;
09:   else
10:     return TABLE3;
11: }
```

図2．プログラム例 — 条件に応じてテーブルを選択するプログラムである。
Example of program using symbolic execution

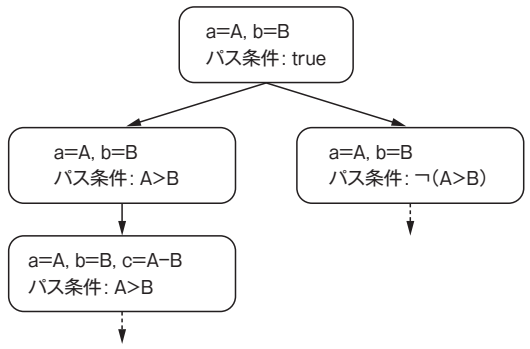

図3. 記号実行による実行木 — 条件分岐によって分岐しながら実行が進む。
Execution tree of symbolic execution

ノードは初期状態を，複数の子を持つノードは実行の分岐を表している。

実行が進むとパス条件に条件が追加されていくが，その際にパス条件が矛盾することがある。そのような状態は矛盾する状態であり，具体的な実行でこれを満たす状態に到達することはありえないので，そのような記号状態の実行は打ち切ってよい。このように条件が成立しなくなる分岐の実行を打ち切ることを枝刈りと呼ぶ。

3.3 決定表の生成

同様にして，プログラムの対象箇所（例えば関数）の最後まで実行を継続することができる。これにより，実行後の状態を表した記号状態が前述の木の末端ノードとして得られる。

この例では以下を得ることができる。ここで，resultは関数の返り値を表す特殊な変数であり，∧は“かつ（and）”を表す記号である。

割当て: a=A, b=B, c=A−B+10, result=TABLE2
パス条件: A>B ∧¬ (A==B) ∧ A+B+10<20 (4)

割当て: a=A, b=B, c=A−B+10, result=TABLE3
パス条件: A>B ∧¬ (A==B) ∧¬ (A+B+10<20) (5)

割当て: a=A, b=B, c=B−A+10, result=TABLE1
パス条件: ¬ (A>B) ∧ (A==B) (6)

割当て: a=A, b=B, c=B−A+10, result=TABLE2
パス条件: ¬ (A>B) ∧¬ (A==B) ∧ B−A+10<20 (7)

割当て: a=A, b=B, c=B−A+10, result=TABLE3
パス条件: ¬ (A>B) ∧¬ (A==B) ∧¬ (B−A+10<20) (8)

これらの最終状態のパス条件には，A>B，A==B，A−B+10<20，B−A+10<20という四つの条件とその否定が含まれている。A，Bは変数a，bの初期値を表しているため，条件式中のA，Bを元の変数に読み替えると，これらはプログラム実行開始時のa>b，a==b，…という条件に等しい。これら条件の真偽によるresultの値の決まり方を表にすることで，**表2**のような決定表を得ることができる。

表2. 生成される決定表の例
Example of extracted decision table

組合せ条件と結果		条件成立（Y）及び不成立（N）と得られる結果（X）				
条件	a>b	Y	Y	N	N	N
	a==b	N	N	Y	N	N
	a−b+10<20	Y	N	−	−	−
	b−a+10<20	−	−	−	Y	N
結果	TABLE1	−	−	X	−	−
	TABLE2	X	−	−	X	−
	TABLE3	−	X	−	−	X

＊ーは，真偽を問わない，あるいは結果なし

4 プロトタイプの開発と評価

考案した技術を実現し評価するために，C言語を対象としてツールのプロトタイプを開発した。C言語を対象としたのは，組込みシステムはC言語で記述される場合が多く，製品寿命の長いインフラシステム向けソフトウェアでその再利用及び再設計が課題となっているためである。

4.1 プロトタイプの構成

開発したプロトタイプは，C言語向け有界モデル検査ツールであるCForge[6], [7]のフロントエンドと内部表現を利用し，その上に記号実行エンジンや決定表の生成機能を実装することで実現した。構成の概要を**図4**に示す。

このプロトタイプは，C言語のソースコードを元に構文解析を行い，プログラムの構造を木で表した抽象構文木を得る。更に，抽象構文木を元に制御の流れを解析し，解析に適した内部表現へと変換する。これらの機能はCForgeと共通のも

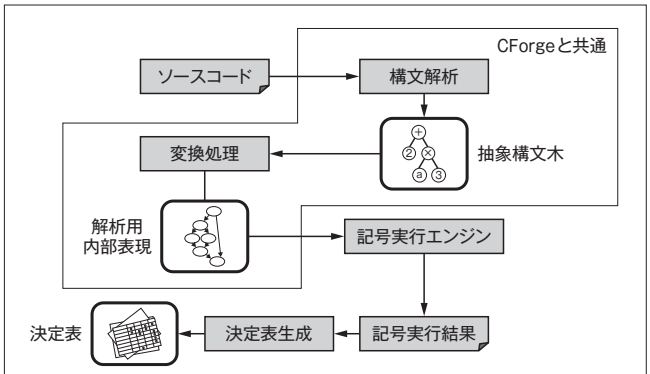

図4. 決定表抽出ツールの構造 — C言語向け有界モデル検査ツールを再利用し，その上に記号実行による決定表抽出のための機能を実装した。
Structure of decision table extractor

一般論文

のを用いた。

その結果を元に記号実行を行う。今回必要な実行パターンごとの条件を取得する機能は，CForgeの記号実行エンジンでは備えていなかったため，それを取得可能な記号実行エンジンを独立に実装した。

この記号実行エンジンは実行の枝刈り処理として，条件式の単純化処理によって偽になる条件をパス条件に含む場合，及びある条件とその否定の両方をパス条件に含む場合，その実行パターンは実際にはあり得ない矛盾したパターンとして除外する処理を実装した。

条件式の単純化処理としては，2+3を5に書き換えるなど，演算の引数が全て定数である場合にその値を計算する処理を行う。また，変数（ここではxとする）が残っている場合にも，0×xを0に書き換えるなど簡単な規則で単純化できるときには単純化するものとした。

また，対象プログラムは一般にループが含まれており，これらの処理は実行回数が事前に決まっていないため，記号実行をどこまで行えばよいかがわからないという課題がある。そこで今回のプロトタイプでは，ループに関して，ループ本体の実行回数があらかじめ指定された回数以下である実行パターンだけを対象として解析を行うことにした。

4.2　評価と今後の取組み

プロトタイプを用いて実験した結果，複雑なポインタ操作などを含まない関数に関しては，人間が作る決定表に近いサイズの決定表を生成できることを確認できた。

例えば，ある製品システムの日付操作関数では，不正な状態に至らない実行パターンに限って手作業で決定表を作成したところ13条件14実行パターンになったが，このプロトタイプは同条件の下，不正な状態に至るかどうかなどを判定するための18条件を除いて，13条件15実行パターンの決定表を生成することができている。実行パターン数の差は，記号実行時の枝刈りの不完全性に起因するものであった。

したがって，今回開発したツールを用いることで人間が作る決定表に近いサイズの決定表を生成でき，既存ソフトウェアの再利用時に掛かっていた多大な工数を削減することが期待できる。特に長期間にわたって保守と変更が行われている社会インフラ向けのソフトウェア開発における効果が期待でき，当社の社会インフラ向けのソフトウェア開発部門からも，この手法は製品開発に役だつ可能性が高いという評価を得ることができている。

その一方で，出力される決定表が大きくなりすぎる場合もあることが明らかになっている。例えば，極端な場合として，実際に既存ソフトウェアの再利用及び再設計をする現場で課題となっているあるロジックでは，624条件3,556実行パターンの決定表が出力されたが，これを人間がそのまま見ても理解するのは難しい。

そこで，このような複雑なロジックに対しては，差分計算を行うことにより，仕様変更に対応するために修正が必要な箇所に限定した表示や，ソースコード変更前後で影響のある部分に限定した表示といった，設計者が着目するポイントに絞った提示を行う改良を進めている。

5　あとがき

ソフトウェアのソースコードから設計情報や仕様を得ようという試みは仕様発掘と呼ばれる活発な研究領域となっている。

当社は，ここで述べた決定表抽出技術に加え，プログラム部品を利用する際に満たさなくてはならない条件を発掘する技術[8]などの研究開発を行っている。これらの技術を洗練化して多製品への適用を図るとともに，更なる仕様発掘技術の開発を推進し，ソフトウェアの再利用及び再設計を支援していく。それにより，低コストで，かつ高信頼なシステムが提供できるよう寄与していきたい。

文　献

(1) 情報処理推進機構. 2011年度「ソフトウェア産業の実態把握に関する調査」調査告書. 2012-08-10, 139p. ＜http://sec.ipa.go.jp/sweipedia/catl001/catm008/cats064/56366/＞,（参照2012-11-13）.

(2) 今井健男 他. “仕様発掘の動向と目指すべき方向性”. ウィンターワークショップ2012・イン・琵琶湖. 彦根, 2012-01, 情報処理学会 ソフトウェア工学研究会. 2012, p.75－76.

(3) King, J. C. Symbolic execution and program testing. Communications of the ACM. **19**, 7, 1976, p.385－394.

(4) 酒井政裕 他. “記号実行によるプログラムからの決定表抽出”. 2011年ソサイエティ大会論文集. 札幌, 2011-09, 電子情報通信学会. 2011, p.196.

(5) 森田隼史 他. 鉄道運賃計算アルゴリズム－Suica/PASMO 利用可能範囲のJR東日本510駅の運賃を対象とした場合－. 日本オペレーションズ・リサーチ学会和文論文誌. **54**, 2011-12, p.1－22.

(6) 酒井政裕 他. “CForge：C言語プログラムのための有界検査ツール”. 第11回プログラミングおよびプログラミング言語ワークショップ（PPL2009）. 高山, 2009-03, 日本ソフトウェア学会 プログラミング論研究会. 2009, p.118－132.

(7) 酒井政裕 他. データ構造に関する仕様を含め検証できるC言語プログラム部品検証ツール CForge. 東芝レビュー. **64**, 8, 2009, p.20－23.

(8) 今井健男 他. “Minimal Unsatisfiable Core列挙によるプログラムの準最弱な事前条件推定”. 第18回ソフトウェア工学の基礎ワークショップ（FOSE2011）. 青森, 2011-11, 日本ソフトウェア科学会 ソフトウェア工学の基礎研究会. 2011, p.187－196.

酒井　政裕　SAKAI Masahiro
研究開発センター システム技術ラボラトリー。
ソフトウェアの設計・検証技術の研究に従事。
System Engineering Lab.

岩政　幹人　IWAMASA Mikito, Ph.D.
研究開発センター システム技術ラボラトリー主任研究員，博士（情報科学）。ソフトウェアの設計・検証技術の研究に従事。情報処理学会，計測自動制御学会，IEEE会員。
System Engineering Lab.