ソースコードからの仕様発掘技術

Specification Mining Technology for Automatically Inferring Software Specifications from Source Code

今井　健男　　酒井　政裕　　岩政　幹人
■IMAI Takeo　■SAKAI Masahiro　■IWAMASA Mikito

ソフトウェアの派生開発を効率的に行うには，仕様とプログラムの対応が維持されていることが重要である。

東芝は，プログラムのソースコードから仕様情報を発掘し，それを仕様へ反映するための技術開発を行っている。仕様に相当する情報をプログラムから機械的に発掘し抽出できれば，今後の製品展開で有用な情報を効率的に洗い出し，設計書を常に最新の状態に保つことができる。今回，その取組みの一環として，プログラムが正しく動作するための前提と仮定している条件（事前条件）を仕様としてプログラムから自動推定するための新たな技術を開発した。この技術を用いてC言語向けに試作したツールを例題プログラムに適用し，この技術が実用的で，人が考えるのと同等以上に汎用的な仕様を得ることができ，この技術の実用性を確認できた。

Correspondence between a program and its specification is essential for efficient redesign and reuse of software.

Toshiba has been developing a technology for mining latent specifications from a given program and reflecting them in the original specification documents. This technology enables developers not only to acquire latent information that is useful for future development and reuse, but also to keep specifications up to date. As part of this work, we have developed a new technology for automatically inferring preconditions from programs; that is, conditions that a program assumes in order to perform its intended function. We have conducted experiments on a prototype tool for C language, and confirmed its ability to obtain generic specifications similarly to or more effectively than by human effort.

1　まえがき

現在のソフトウェア開発では，従来製品のソフトウェアを横展開又はシリーズ展開することが多く，一度書かれたソフトウェアが幾度も拡張され再利用される。しかし，既存のプロジェクトが他社のプロジェクトである場合や，設計者の退職及び移動などで設計書の整備と保守に多大な工数がかかる場合も多く，後継の開発者にとって大きな負担になっている。そのため，仕様からプログラムを開発する従来の工程に加え，相互の対応を維持する工程を効率化することも重要だと考えられる。

そこで東芝は，プログラムのソースコードから仕様情報を発掘し，それを仕様へ反映するための技術開発を行っている。仕様に相当する情報をプログラムから機械的に発掘し抽出できれば，今後の製品展開で有用な情報を効率的に洗い出し，設計書を容易に最新の状態に保つことができる（**図1**）。

ここでは，その取組みのうち，プログラムの事前条件と呼ばれる仕様の発掘技術，C言語を対象としてこの技術を実現したツール，及び今後の展望について述べる。

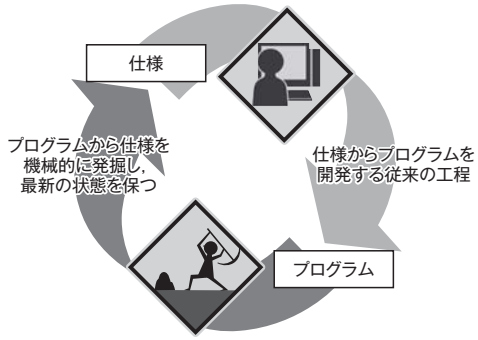

図1．仕様発掘技術による従来の開発工程の補完 ― 仕様からプログラムを開発後，相互の対応の維持に多大な工数がかかっていたが，仕様発掘技術を用いることで仕様の整備と保守を効率的に行うことができる。
Combination of specification mining and conventional software development

2　事前条件とは

ソフトウェアの信頼性を向上させる手段の一つに，プログラムの仕様として，事前条件と事後条件の組から成る“契約”を記述する方法が知られている[1]。

事前条件とは，プログラムが正しく動作するための前提として仮定している条件で，プログラムを呼び出す側が保証する義務を負う。それに対し事後条件とは，それと引換えにそのプログラムが実現することを期待されている条件で，呼び出された側のプログラムが実行後に成り立っていることを保証する義務を負う。

事前条件と事後条件の組を記述することで，プログラマーがプログラムの挙動を把握しやすくなり，また，テストやより数

学的に厳密な検証手法（形式検証）の足がかりとすることができる。しかし，事前条件と事後条件を人手で全て考え記述するのは，往々にして煩雑で，漏れや抜け，まちがいが混入しやすい。

そこで当社は，そのうちの事前条件を自動推定するための新たな技術を開発した。この技術で得られる条件は，従来の技術で得られた条件と比べ，人が容易に理解可能な程度に単純であり，かつ，人が仕様として再利用可能な程度に汎用性があるという特長がある。

3　開発した技術の概要

この章では簡単な例題を用い，開発した技術の概要について述べる。

図2(a)は，絶対値を計算するC言語関数absを実装したプログラムのソースコードである。そして図2(b)は，このプログラムに期待される事後条件を記述したものである。このフォーマットは，当社で開発したソフトウェアの検査ツール[(2), (3)]で採用しているもので，ensuresに続く宣言2行がそれぞれ事後条件を表し，\resultが戻り値を表している。

このプログラムは，引数xがどのような値であっても事後条件を満たすように見えるが，整数に2の補数表現を用いる一般的なC言語処理系を使って動作させると，xが整数型（int型）の最小値であるときに符号が反転せず，事後条件を満たさない。したがって，“xが整数型の最小値より大きい”がこのプログラムに対する事前条件になる。

以下で，このソースコードと事後条件の二つが与えられたときに，この技術がどのようにこの事前条件を求めるかを見ていく。

まず，この技術では事前条件の部品として，プログラムの入力（引数）に関する条件式をシステムが機械的に生成する。ここでは，図2(c)に示す七つの条件式を生成したとする。INT_MINは整数型の最小値を，INT_MAXは最大値を表す。以下，各条件式を図中のA～Gの記号で参照する。

次に，この技術ではこのプログラムを記号実行[(注1)]することで，プログラムの入力及び実行前の状態とプログラムの出力及び実行後の状態の関係を表す論理式Psへと変換する。このプログラムの場合には以下のような論理式(1)となる[(注2)]。

$$(x < 0 \land \backslash result = x \times (-1)) \lor (\neg(x < 0) \land \backslash result = x) \quad (1)$$

また，二つの事後条件の連言（and結合）をとった以下の論理式(2)をZとする。

$$\backslash result >= 0 \land (\backslash result \times \backslash result = x \times x) \quad (2)$$

ここでPsが成り立つと仮定したときに，論理式の集合P={A，B，C，D，E，F，G，¬Z}の要素の組合せのうち，どの組合せが矛盾するかを判定すると，以下の9個のいずれかを含む組合せだけが矛盾することが判明する。言いかえれば，この9個のそれぞれは，矛盾を発生させる極小の組合せである。

$$\{A,\neg Z\}, \{D,\neg Z\}, \{A,E\}, \{A,B,C\}, \{G,\neg Z\}, \{D,E\}, \{B,G\}, \{E,G\}, \{F,G\} \quad (3)$$

このような矛盾した極小の組合せのことを，PのMinimal Unsatisfiable Core（MUC）と呼ぶ。このMUCを列挙し，事前条件の基とするのがこの技術の骨子である。

ただし，これらMUCのうち“¬Z”を含まないものは，引数に関する条件式どうしが矛盾している，すなわち，このコードの実行時にそれらの条件式が同時に成り立つことはないので事前条件に適さないことから，¬Zを含む{A,¬Z}，{D,¬Z}，及び{G,¬Z}の三つを残す。

ここで，{A,¬Z}の場合を例にとると，Aと¬Zが矛盾しており同時に成立しえないということは，つまり，Aが成り立つときには事後条件であるZも必ず成り立つということであり，{A}は事前条件の性質を満たしている。したがって，先ほどの三つの組合せのそれぞれから¬Zを除いた{A}，{D}，及び{G}が事前条件になる。一般に，複数の条件式の連言（and結合）が事前条件となるが，この例では，たまたま全ての事前条件が単一の条件式から成る。

しかし，これら三つの事前条件を見比べると，Gが成り立つ際には常にAも成り立ち，同様に，Aが成り立つ際にはDも必ず成り立つ。すなわち，GよりもAのほうが，AよりもDのほうが，より広い範囲の入力で成り立つ。事前条件としては，できるだけ広い範囲の入力で成り立つ汎用的な条件が望ましいため，この技術では，得られた事前条件の間に包含関係があるかを判定し，他の事前条件に包含される事前条件は推定結果から除外する。

```
int abs(int x) {
  if (x < 0)
    return x * (-1);
  else
    return x;
}
```

(a) プログラム

```
abs(x) {
  // 戻り値が0以上
  ensures \result >= 0;
  // 戻り値の2乗が引数xの
  // 2乗に等しい
  ensures \result * \result
    == x * x;
}
```

(b) 事後条件

```
A: (x >= 0)
B: (x <= 0)
C: (x != 0)
D: (x > INT_MIN)
E: (x = INT_MIN)
F: (x < INT_MAX)
G: (x = INT_MAX)
```

(c) 生成される条件式

図2．絶対値を計算する関数abs ― 開発した技術では，プログラムのソースコードと事後条件を入力とし，また，内部では事前条件の部品となる条件の生成を行う。
"abs" function for calculating absolute value

(注1)　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。

(注2)　ここで，¬は否定を，∧と∨はそれぞれ“かつ（and結合）”と“または（or結合）”を表す記号。

こうして最終的な事前条件として{D}が残る。これは，最初に示した“xが整数型の最小値より大きい”という事前条件そのものである。

4　ツールの試作

3章で示した原理に基づいて，開発した技術を実現するツールを試作した。その構成を**図3**に示す。

4.1　ソフトウェア有界モデル検査ツール

今回試作したツールでは，バックエンドのエンジンとして，当社が開発したソフトウェア有界モデル検査ツール[2), (3)]を用いる。これはC言語向けの検証ツールであり，C言語のソースコードと，JML（Java[注3] Modeling Language）風の仕様記述言語による仕様記述を読み込み，仕様記述に含まれる事前条件及び事後条件をソースコードが満たすかどうか検証する。

このソフトウェア有界モデル検査ツールは，ソースコードを関係論理と呼ばれる論理系[4]の論理式に変換したうえで，これと事前条件及び事後条件の否定が同時に成り立つ場合を探索する。成り立つ場合が存在すれば，それを反例，すなわち仕様を満たさないプログラムの実行例として出力する。反例は，検査対象プログラムのバグの存在を強く示唆するものである。反例が探索で見つからなければ，バグが見つからなかったことになり，検査は成功したとみなす。

試作したツールでは，これを利用して内部で条件式を機械生成し，それらを全て仮の事前条件としてソフトウェア有界モデル検査ツールに与える。事後条件を入力する仕組みは，元のソフトウェア有界モデル検査ツールの機能をそのまま流用する。そして，ソフトウェア有界モデル検査ツールが反例を探索する仕組みそのものを制約ソルバ[注4]として，4.2節で述べるMUCの計算に利用する。

試作したツールは，元のソフトウェア有界モデル検査ツールが扱えるC言語の文法を取り扱うことができ，また，元のソフトウェア有界モデル検査ツールが扱える仕様記述の構文をそのまま事後条件の記述に用いることができる。C言語の言語要素がどのように扱われるか，また，それらに対する仕様記述をどのように与えることができるか，についての詳細は割愛する。

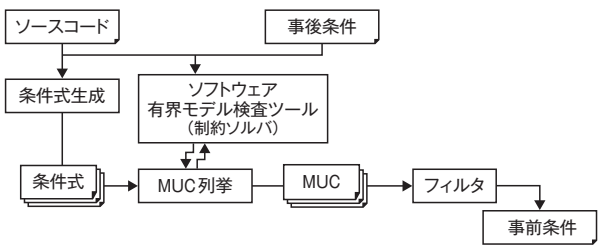

図3．ツールの構成 — 試作したツールは，ソフトウェア有界モデル検査ツールを制約ソルバとして用いてMUCを列挙し，それをフィルタリングすることで事前条件を生成する。
Structure of our tool

（注3）　Javaは，Oracle Corporation及びその子会社，関連会社の米国及びその他の国における登録商標。
（注4）　複数の制約条件（例えば連立不等式）が与えられたときに，全ての制約条件を満たすような値（＝解）を探索するツール。

4.2　MUCの列挙

MUCの列挙には，LiffitonとSakallahのアルゴリズム[5]を採用した。このアルゴリズムは，複数の論理式が与えられたときに，まず，同時に成り立ちうる論理式の組合せのうち最大のもの（それ以上の論理式は同時に成り立ちえないもの）を制約ソルバを用いて全て列挙し，次に，得られた組合せの集合をMUCの集合に変換する。このアルゴリズムを用いると，与えられた論理式の集合に対して全てのMUCを列挙することが可能である。

このアルゴリズムの優れた点は，対応する制約ソルバさえあれば，任意の制約条件に関するMUC列挙アルゴリズムとして応用可能な点である。試作したツールでは当社が開発したソフトウェア有界モデル検査ツールを制約ソルバとして適用した。

4.3　フィルタ処理

複数のMUCが得られた後，それらのMUCのうち，条件式どうしが矛盾するMUCをフィルタリングして，事前条件を構成するものだけを抜き出す。しかし，これだけでは，3章で述べたように他の事前条件に包含される事前条件も残ってしまう。この問題を解決するために，既に求めたMUCの情報を再利用することで，条件式の組合せ間の包含関係を効率的に求める技術を開発した[6]。

5　評価実験

こうして試作したツールを評価するため，整数型又は整数の配列型を引数に持つ，教科書的なアルゴリズムをC言語で実装した関数に対して適用し，実用的な事前条件，すなわち人が容易に理解可能で，かつ再利用可能な事前条件を生成できることを確認した。

5.1　実験詳細

実験対象の関数としては，最大公約数の計算（ユークリッドの互除法），配列の総和の計算，配列の探索（線形探索，二分探索），メモリのコピー（memcpy，strcpy），及び配列の整列（選択ソート，クイックソート）を用いた。

事前条件を表現するために用いる条件式としては，整数値の大小の比較を用いた。比較する整数値としては，関数の整数型引数の値，ポインタ引数の指す先の有効メモリ領域の大きさ，及びゼロを用いた。

事前条件と引き換えにプログラムが保証する事後条件に関しては，“今回用いたソフトウェア有界モデル検査ツールがデフォルトで検査する二つのエラー（メモリの範囲外アクセスとサブ

ルーチンコールにおける事前条件違反）が起こらない”ことだけを考える場合（以下，実験1と呼ぶ）と，それに加えて“ユーザー定義による，ユーザーが望ましいと考えるそれぞれの関数事後条件”も考える場合（以下，実験2と呼ぶ）の2通りを設定し実験を行った。

5.2 実験結果

実験1では，人手による条件と比較し，8個中6個で同等かそれよりも汎用的な条件を発見できた。ただし，memcpyに対しては，正しくない条件も同時に出力された。実験2でも，8個中4個でそのような条件を求めることができた。これらの条件は，より汎用的な条件が存在しないか，又はより汎用的な条件があったとしても，それが複雑で実用的でないものになることを数学的証明などを用いて確認した。

人手によるものより汎用的な条件が発掘できた例として，ユークリッドの互除法の場合を**表1**に示す。このユークリッドの互除法のような，実行回数が固定でないループを含んだプログラムの事前条件を求めることは，補助的な制約条件を人手で与えることなしには今まで非常に困難だったが，この技術を用いれば有効な事前条件を自動で求められることがある。

表1．人手で求めた事前条件と推定した事前条件の比較
Comparison of manually-specified and inferred preconditions

人手で求めた条件	推定された条件
$m \geqq 0 \wedge n \geqq 0 \wedge (m \neq 0 \vee n \neq 0)$	(1) $m > 0 \wedge n \geqq 0$ (2) $m \geqq 0 \wedge n > 0$ (3) $m \geqq 0 \wedge n \geqq 0 \wedge m \neq n$

その一方，論理的には事前条件となるが，人にとっては妥当な入力を許さないほど強い条件が出力されたものや，事前条件が一つも求まらなかったものも存在した。これらは，本来望ましい事前条件が，この実験で与えた条件式の組合せで表現できなかった場合であり，生成する条件式のバリエーションが，推定される事前条件の内容に大きく影響することがわかった。

また，実験1と2を合わせて，16個中4個で今回用いたソフトウェア有界モデル検査ツールが正しい反例を見つけられなかったため，偽の事前条件が出力されていた。これは，この技術そのものの問題というより採用する制約ソルバの問題であるが，この種の問題はソフトウェア有界モデル検査ツールを用いれば一般的に発生するものであり，その対策は今後の課題である。

性能面に関しては，この実験ではクイックソート以外は数秒から3分程度と，全て実用上問題ないと思われる時間で動作した。クイックソートの例でも，パラメータを変えると30分程度の時間で動作した。クイックソートはサブルーチン呼出しと再帰を使っており，ソースコードの行数以上に内包する処理が複雑であることが，計算時間に大きく影響している。

6 あとがき

当社は，MUCの列挙による事前条件の推定技術を開発し，ツールを試作した。試作及び実験によって，開発した技術が原理上，実用的な事前条件を推定できることが確認できた。また，ここで述べた実験では整数型とその配列を引数に持つようなC言語関数だけを扱ったが，今後はリスト構造やツリー構造など，実際のソフトウェアでよく使われる，より複雑なデータ構造を扱えるように拡張していく。

また，ここで述べたような，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（specification mining）と呼ばれる活発な研究領域になっている[7]。当社もここで述べた技術に加え，設計情報の一種である決定表を発掘する技術[8]などの研究開発を行っている。引き続きこれらの技術を洗練化し展開を図るとともに，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や再設計を支援していく。それにより，ローコストでかつ高信頼なシステムを提供していきたい。

文 献

(1) Meyer, B. Object-Oriented Software Construction. USA, Prentice-Hall, Inc., 1988, 534p.
(2) 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.
(3) Sakai, M. et al. "Model-checking C programs against JML-like specification language". Proceedings of the 19th Asia-Pacific Software Engineering Conference (APSEC2012). Hong Kong, 2012-12, IEEE/ACM. p.174－183.
(4) Jackson，D．抽象によるソフトウェア設計 ― Alloyではじめる形式手法．東京，オーム社，2011，368p.
(5) Liffiton, M.; Sakallah, K. Algorithms for Computing Minimal Unsatisfiable Subsets of Constraints. Journal of Automated Reasoning. **40**, 1, 2008, p.1－33.
(6) 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．コンピュータソフトウェア．**40**，1，2013，p.207－226.
(7) 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．p.75－76.
(8) 酒井政裕 他．記号実行によるプログラム改造支援技術．東芝レビュー．**67**，12，2012，p.35－38.

今井 健男 IMAI Takeo
研究開発センター システム技術ラボラトリー研究主務。形式手法，ソフトウェアテスト，プログラム解析などソフトウェア工学の研究に従事。ACM，情報処理学会，日本ソフトウェア科学会会員。
System Engineering Lab.

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.