SMT を活用した Java プログラム解析器の設計

A design of analyzer for Java program using SMT Solver

佐々木 幸広* 岡野 浩三† 楠本 真二‡

あらまし ソフトウェア開発において，プログラムのテストや検証はソフトウェアの品質保障のために重要であるが，コストがかかる作業である．そこで我々は SMT ソルバを用いた, Java プログラム解析を支援する解析器を提案する．このツールを使用することにより，プログラムのパス実行可能性の検査や，実行パスを実行する引数の値の求解，また事前条件や事後条件が成り立つかなどの様々な検査などが可能である．このツールでは，プログラム依存グラフ (PDG) から得られる Java の実行パスの実行条件を SMT ソルバの式へと変換する．これを SMT ソルバで解くことにより，実行パスの実行可能性や，条件を満たす割り当てを求めることができる．また，事前条件や事後条件を入力として与えることでメソッド単位の有界モデル検査も可能である．ツールを複数の Java プログラムに対して，評価実験を行った．その結果，正しい検査結果が得られたことを確認できた．

1 はじめに

ソフトウェアのテストや検証は，ソフトウェア開発や保守において重要な役割を果たす [1]．しかし，プログラムのサイズの増加に伴い，人手により全ての機能のテストや検証を行うことは一般に非常にコストがかかる作業である．文献 [3] [4] では SMT を使った有界モデル検査を実現し、文献 [5] [6] では SMT を使ったプログラム解析を実現している．しかし両方を可能にするツールは著者らの調べた限りでは存在しない. そこで様々な用途に用いれるような，SMT ソルバ [2] を用いた Java プログラム解析器を提案する．このツールを用いることで，利用者の用途に応じて様々な検査が可能である．提案ツールでは，プログラム依存グラフ (PDG) [7] から得られる Java の実行パスの実行条件や各種表明を SMT ソルバの入力に変換する．これを SMT ソルバの一つである Z3 [8] で解くことにより，実行パスの実行可能性や，条件を満たす割り当てを求めることができる．ツールを複数の Java プログラムに対してパス実行可能性等の評価実験を行った．その結果，正しい検査結果が得られたことを確認できた．以降，2 章で研究背景として，本研究で用いるツール等について説明を行う．続いて 3 章で提案するツールの概要について述べ，4 章で主な使用用途について述べる．5 章で部分的な評価実験および考察について述べる．最後に 6 章でまとめる．

2 研究背景

本章では，本研究で用いる概念，ツールについて簡単に述べる．

2.1 PDG

提案手法では，プログラムの実行パスを算出するために PDG を用いる．PDG [7] はプログラム依存グラフの略で，命令文に対応するノードと，制御依存辺，データ依存辺，実行依存辺からなる有向グラフである．制御依存辺は次に実行しうるノード間に構築され，データ依存辺はデータ依存があるノード間に構築される．

*Yukihiro Sasaki, 大阪大学大学院情報科学研究科

†Kozo Okano, 大阪大学大学院情報科学研究科

‡Shinji Kusumoto, 大阪大学大学院情報科学研究科

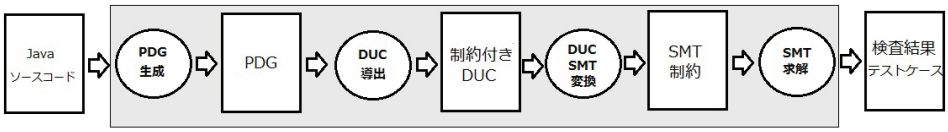

図 1　ツールの概要図

2.2　SMT ソルバ

提案手法では，条件の検査を行うために SMT(Satisfiability Modulo Theories) ソルバを用いる．SMT は，算術演算や配列に関する制約を記述できる理論的体系であり，SMT ソルバは SMT の制約を満たす割り当てを求める解析器である [2]．SMT ソルバに，問題を変数や関数に関する制約式として入力し解くことで，充足可能性の判定と，充足する場合の変数への値の割り当てを求められる．提案手法では SMT ソルバの 1 つである Z3 [8] を用いる．Z3 は SMT-LIB2.0 が入力の標準であり，様々な型や演算を扱うことができ，実行速度や正答率などの性能も高い．

3　ツール概要

本手法ではまず，PDG から得られる Java のパス実行条件を SMT ソルバの文法へと変換する．この情報を SMT ソルバで解くことにより，条件の検査やテストケースの生成など様々なプログラムの解析が可能である．有界モデル検査を行う場合には，各実行パスに対する条件の論理積の充足可能性を判定することにより，メソッド単位での検査を行う．

3.1　提案手法概要

本ツールの概要を図 1 に示す．また，本手法の入力と出力は以下のとおりである．

- 入力：Java プログラムの 1 メソッド，検査したい実行パス (0 個以上)，
  事前条件 (0 個以上)，事後条件 (0 個以上)
- 出力：条件及びパス実行条件の真偽，条件を満たす引数の具体的な値

検査したい実行パスと事前条件，事前条件や事後条件はそれぞれオプションとして与えることができ，パスを指定しない場合はメソッド単位の有界モデル検査，条件を指定しない場合はパス実行可能性の検査となる．各条件は，JML [9] を用いて与える．事前条件はプログラム片の実行前に成り立つべき条件であり，事後条件はプログラム片の実行後で成り立つべき条件とし，それぞれ複数個与えることができる．この際，条件式に JML の構文エラーや，プログラムで使用されてない変数を用いるなど，意味的なエラーがあってはならない．また，検査したい実行パスを入力として数値の系列を用いて与える．ここで各数値はこのシステム生成される PDG の枝に付与される値である．提案手法の手順は以下のとおりである．

1. 対象メソッドの関数内プログラム依存グラフ (Intraprocedural Program Dependence Graph，以降 PDG) [7] を生成する．
2. 対象メソッドにおいて，解析したいパスと検査したい条件を指定する．パスの指定がない場合には，メソッドの有界モデル検査となる．
3. 手順 1 で生成した PDG より手順 2 で指定した対象となる，後述の実行パスの制約付き DUC を取得する．有界モデル検査の場合，PDG の各実行パスの制約付き DUC の論理積を取得する．
4. 手順 3 で生成した制約付き DUC を SMT-LIB2.0 で表現された制約式に変換する．検査したい条件も，同様に変換を行う．
5. 手順 4 で生成した制約式を Z3 を用いて求解を行い，充足可能性と割り当てがあれば変数割り当てを求める．

手順 5 で充足可能であるとき，検査したい条件が成り立つことが分かる．またその解の割り当てを求めることで，その条件を満たし，かつそのパスを実行するための具体的な引数の値 (テストケース) が得られる．この際，条件を追加しなかったときは，パスの実行条件や，そのパスを実行するためのテストケースが求まる．一つのパスに対し，異なる複数のテストケースを生成できる．

実行パスを指定した場合，実行パス単位での検査が可能できる．検査したい実行パスを表す数を与えたうえで，各条件の検査を行う．条件を追加しない場合はパス実行条件の調査となる．

各実行パスに対する条件の論理積の充足可能性を判定することにより，メソッド単位での有界モデル検査 [1] も行える．この際プログラム中のループはボディをユーザが定めた回数だけ実行するように展開し解析を行う．主な提案ツールの利用法は 4 章にて後述する．

3.2 PDG の生成

入力として与えられたソースコードおよび実行パス指定から，PDG [7] を生成する．本ツールでは対象ソースコード全体に対して PDG を生成せず，メソッド内とクラスのフィールドに限定して PDG を生成している．本研究ではこの PDG 生成部の実装には MASU [10] を利用した．

3.3 制約付き DUC の取得

指定したパスの制約付き DUC を PDG から取得する．DUC [11] は，変数使用—定義（DUP）の系列として表される [12].

実行文は PDG のノードに対応するので，DUC は PDG の出口ノード（return 文などのメソッド終了時に実行される可能性のあるノード）からデータ依存辺をたどることで得られる実行文の系列となる [12]．また，DUC を通るパスを実行するための条件も同様に PDG の制御依存辺をたどることで得られ，その条件が追加されたものを制約付き DUC と呼ぶ. 制約付き DUC は，DUC の要素である変数使用—定義（DUP）にその DUP を実行するための条件を付加したものであり，“制約付き DUC を実行できる” とは，“制約付き DUC に含まれるすべての定義条件が成立する” ことを意味する．制約付き DUC を SMT ソルバで解くことにより，DUC の実行可能性を検証することができる．

3.4 制約付き DUC から SMT-LIB2.0 への変換

本節では，制約付き DUC や検査する条件を SMT-LIB2.0 へ変換する規則の概要を示す．制約付き DUC は変数定義を行うときに成立すべき条件をもつため，それを SMT ソルバで解くことで，DUC 系列が実行可能であるときの割り当てを求められる．また，検査したい事前条件や事後条件に対しても同様に変換を行い，SMT ソルバで解くことで，DUC 系列を実行したときに検査したい条件が成り立つかを求めることができる．

なおメソッド呼び出しの変換に関しては，メソッドに表明がついている場合には，その表明が表す条件式を変換する．返り値は事後条件を満たす変数として定義し，変換する．表明がついていない場合はメソッドの返り値は任意の値を取りうる変数として変換する．

この操作により，複数の他のメソッドを呼び出すメソッドに対しても対応可能である．なお，本ツールでは JML の種類を事前条件，事後条件，不変条件のみに限定する．

3.4.1 変数定義変換規則

本小節では変数定義に対する変換規則を示す．基本型は整数として扱う型と実数として扱う型および真偽値として扱う型に分類し，それぞれ対応する SMT-LIB2.0 の型へと変換する．参照型は SMT-LIB2.0 ではレコードとして定義を行うことによ

り，フィールドの表現が可能である．また，配列も対応する SMT-LIB2.0 の型へと変換可能である．同一の異なる変数名に関しては，PDG 生成時に名前が識別可能となる．全てを説明するスペースはないため，詳しくは文献 [13] を参照．

3.4.2 式変換規則

Java 実行パスに現れる式を SMT-LIB2.0 へ変換する [13]．演算子は，基本型に対するシフト演算子，ビット演算子，条件演算子，文字列連結演算子以外に対応している．これらの演算子は SMT ソルバで対応する演算が定義されていなかったため，今回は非対応とした．

SMT-LIB2.0 には代入に相当する演算子はない．そのため，基本型変数への代入は，代入される変数を宣言し，新しい変数と被代入式が等しいという式を用いて表現する．新しい変数名は元の変数名にその変数に新たに値が代入された回数を付け加えて宣言し，区別する．また，参照型変数に対しては，オブジェクトに新たな値やメソッドの実行が行われるたびに，基本型変数の場合と同様に新たなオブジェクトを宣言する．詳しい変換方法は論文 [13] 参照．また検査したい条件は，Java に類似した JML [9] という文法で表されており，それらも同様に変換を行う．

なお，while,for 文などのループは固定回実行するように展開するものとする．ループがある場合に全ての状態を検査するのは非常にコストがかかり，状態爆発を起こす場合があるからである．解析を行う処理系では 2 回展開するものが多く，それらに習い，デフォルトの設定を 2 回とした．

3.5 出力結果の解析

SMT ソルバは入力された制約を満たす変数の割り当てがあるかどうかを検査し，変数割り当てを試行する．割り当てが存在する場合，検査結果が真であることが分かる．変数割り当てを解析することで，変数の値をテストケースとして得られる．

4 使用用途

本解析器の現時点での主な使用法を API として表 1 に示す．

4.1 メソッドの実行パス単位での条件の検査

指定した実行パスにおいて，検査したい事前条件や事後条件をそれぞれ複数 (0 個も可能) 指定し，その条件が成り立つかを検査する．この条件は，指定しないことも可能であり，その場合にはそのパスの実行可能性が求まる．オプションで条件を満たす引数の割り当てを求めることもできる．

4.2 メソッド単位での有界モデル検査

メソッドにおいて，検査したい事前条件や事後条件をそれぞれ複数 (0 個も可能) 指定し，その条件が成り立つかを検査する．メソッド内にある全てのパスで条件が成り立つかや，メソッド内で一つでも条件を満たすパスがあるか，などを判定できる．条件を指定しない場合にはメソッド内のどのパスが実行不可能かを求めることができる．また，オプションで条件を満たす引数の割り当てを求めることもできる．

表 1 ツールの API

メソッド名	機能	引数
boolean checkPath	メソッドの実行パス単位での条件の検査	int[] *path*, String[] *pre*, String[] *post*, int *loop*
boolean checkMethod	メソッド単位での有界モデル検査	String[] *pre*, String[] *post*, int *loop*
boolean checkAllPaths	メソッド内の全てのパスでの条件の検査	String[] *pre*, String[] *post*, int *loop*
boolean checkSomePaths	メソッド内のあるパスでの条件の検査	String[] *pre*, String[] *post*, int *loop*
String[] generateTestCase	テストケースの生成	

path:実行パスを指定する数, pre:事前条件, post:事後条件, loop:ループ展開回数

5　評価と考察

本手法の有用性評価のために行った実験について述べる．なお実験環境は，計算機は HP Z800，OS は Windows 7 Enterprise，CPU は Intel Xeon E5607 2.27GHz × 2，メモリは 32.0GB，Java は JDK 1.6.0_26 を用いた．

5.1　評価の概要

評価実験は本手法の有用性を示すために, 検査にかかる時間や検査結果の正当性を評価する．一つ目の実験として，パスが実行可能性の検査を行う．実行可能な場合，それぞれのパスに対し異なる 20 個のテストケースを解が存在する限り生成している．実験対象は，2 次方程式の解を求めるプログラムなど表 2 に挙げるものを使用した．二つ目の実験として，有界モデル検査の実験を行った．現時点では有界モデル検査の機能は設計段階であるため，式生成を手動で行ったうえで，一部の実験対象のみに実験を行った．

5.2　評価実験の結果

提案手法を実験対象のプログラムに対して適用した．パス実行条件の検査を行った場合，表 2 の結果が得られた．実行時間は PDG を生成してから，SMT ソルバが全実行パスに対して求解を終えたときまでの時間を計測している．また，PDG の情報から実行パス候補数や平均パス長なども求めている．検査結果の正当率は，パス実行可能性を正しく判定でき，そのとき得られたテストケースを用いてプログラムを実行したとき，本来通るべき実行パスを通った割合を表す.

5.2.1　実行時間

提案手法の実行時間は主に PDG の生成部と SMT ソルバの求解部が占めている．特に PDG の生成はコストが高い処理であるため，対象メソッドの実行可能性のあるパスが複雑になるほど時間がかかる．SMT ソルバの求解は求解すべき実行パスが長い場合，制約の数が多くなりそれらを満たす変数割り当てを得るために時間がかかることが考えられる．これらのことから，検査を行う対象メソッドの複雑度や大きさによりテストケースを求める時間が増大することが考えられる．

5.2.2　パス実行条件検査の結果

実行不可能なパスを正しく特定することができた．パス実行可能性判定の例を示す．実験対象 RPS はじゃんけんを表すプログラムであり，グー・チョキ・パーを表す 2 つの引数を与えたとき，じゃんけんの勝敗を返す．このプログラムには返り値が勝ちでも負けでもあいこでもないというパスが存在するが，このパスは実行不可能である．提案ツールで検査を行った結果，このパスは実行不可能であることが分かった．また生成されたテストケースを用いてプログラムを実行した結果，本来通

表 2　パス実行可能性検査の実験結果

プログラム名	行数	実行パス候補数	平均パス長	実行時間 (秒)	検査結果の正当率	テストケース数
Calc	21	2	3.5	28.6	100	40
Telephone	29	7	5.5	23.4	100	26
Flower	28	41	9.5	631.5	100	29
JanCord	26	3	13	161.2	100	44
Operator	21	6	3.7	25.9	100	38
RPS	19	5	5	26.4	100	34
Date	28	9	5.4	29.8	100	160

Calc:2 次方程式の求解
Telephone:電話番号がどの種別を表すかの判定
Flower:窓辺の花 [14] のプログラム
JanCord:8 桁のバーコードチェックディジットが正しいかを判定する
Operator:演算子を表す文字を与えたとき，対応する演算を行う
RPS:じゃんけんの判定
Date:二つの日付のうちどちらが新しいかの判定

るべきパスを通ることが確認された．

5.2.3 有界モデル検査の結果

提案ツールの有界モデル検査を実験対象 RPS に対し適用した．現段階では有界モデル検査の機能は完成していないため，一部の実験対象のみとなった．実験対象 RPS のじゃんけんの結果を返すメソッドに対して事前条件として，片方がグーを出す，事後条件としてじゃんけんの結果が勝ちとなる，として検査を行ったところ，条件を満たし勝ちとなるパスが存在することを確認した．実行時間は 28.2 秒であり，条件を追加する前の実験結果よりわずかに長い時間となった．より大きなメソッドの解析も SMT ソルバの性能の限界に達しない限り可能であると考える．

6 おわりに

本稿では，プログラム解析器を提案した．本ツールは PDG と SMT ソルバ Z3 を用いて実行パス単位の検証やメソッド単位での有界モデル検査が可能となる．本手法を複数の Java プログラムに対して適用し評価を行った結果，正しい検査結果が得られた．今後の課題としては，有界モデル検査の実装と評価実験を行い，有用性を評価することが挙げられる．

謝辞 本研究の一部は科学研究費補助金基盤 C（21500036）の助成による．

参考文献

[1] 橋本祐介, 中島震, “有界モデル検査法を用いた C プログラムのモジュラー検証”, 情報処理学会論文誌, Vol. 52, No. 8, pp. 2422-2430, 2011.
[2] 梅村晃広, “SAT ソルバ・SMT ソルバの技術と応用”, コンピュータソフトウェア, Vol. 27, No. 3, pp. 24–35, 2010.
[3] L. Cordeiro, B. Fischer, and J. Marques-Silva, “SMT-based bounded model checking for embedded ANSI-C software,” *In Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, pp. 137–148, 2009.
[4] L. Cordeiro, B. Fischer, “Verifying Multi-threaded Software using SMT-based Context-Bounded Model Checking,” *In Proceedings of the 2011 IEEE/ACM International Conference on Software Engineering 2011*, pp. 331–340, 2011.
[5] P. Jackson, B. Ellis, K. Sharp, “Using SMT solvers to verify high-integrity programs,” *In Proceedings of the 2nd workshop on Automated Formal Methods 2007*, pp. 60–68, 2007.
[6] P. Francis and S. Angelo and T. Paolo, “Using an SMT solver for interactive requirements prioritization,” *In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, No. 11, pp. 48–58, 2011.
[7] J. Ferrante, K. J. Ottenstein, and J. D. Warren, “The program dependence graph and its use in optimization,” *ACM Trasactions on Programming Languages and Systems*, Vol. 9, No. 3, pp. 319–349, 1983.
[8] L. de Moura and N. Bjørner, “Z3: an efficient smt solver,” *In Proceedings of the 14th international conference on Tools and algorithms for the construction and analysis of systems*, pp. 337–340, 2008.
[9] G. T. Leavens, Albert L. Baker, and Clyde Ruby, “JML:A Notion for Detailed Design,” *Behavioral Specifications of Businesses and Systrems*, pp. 175–188, 1999.
[10] 三宅達也, 肥後芳樹, 楠本真二, 井上克郎, “多言語対応メトリクス計測プラグイン開発基盤 MASU の開発”, 電子情報通信学会論文誌 D, Vol. J92-D, No. 9, pp. 1518–1531, 2009.
[11] N. Gupta, “Generating test data for dynamically discovering likely program invariants,” *In Proceedings of ICSE 2003 Workshop on Dynamic Analysis, WODA 2003*, pp. 21–24, 2003.
[12] 宮本敬三, 堀直哉, 岡野浩三, 楠本真二, “Daikon 生成表明改善のためのテストケース自動生成手法とその評価実験”, コンピュータソフトウェア, Vol. 28, No. 4, pp. 4–306–4–317, 2011.
[13] 佐々木幸広, 小林和貴, 岡野浩三, 楠本真二, “SMT ソルバーと PDG 作成ツールを用いた Java のテストケース自動導出手法の提案”, 電子情報通信学会技術研究報告, Vol. 111, No. 481, pp. 55–60, 2012.
[14] 産業技術総合研究所システム検証研究センター, “4 日で学ぶモデル検査 (初級編) (CVS 教程 (1))”, エヌ・ティー・エス, 2006.