3J-02

SMT ソルバを用いた仕様に関する定理の証明とソフトウェア開発への応用

小松　秀生　　岡本　圭史

仙台高等専門学校

１．研究概要

本研究では，文献 Theory of Programs[1]の仕様に関する性質や定理の形式的証明を行う．形式的証明のために，SMT ソルバ[2]の一つ，Z3[3]の API である Z3.py[4]を基にドメイン固有言語(DSL)[5]を開発し，DSL 上で形式的に証明する．さらにその DSL を用いて，単純なプログラムの検証を行う例を示す．

２．研究背景

昨今，高信頼性を保証するソフトウェア開発手法が強く求められている．ソフトウェアの不具合による大きな社会問題が続出している一方で，要求されるソフトウェアの仕様が年々複雑化していることが原因である[6]．

そのようなソフトウェア開発における問題を解決する手段として，形式手法がある．形式手法とは，数学的な基盤を用いて，仕様等を厳密に記述，及び検証するアプローチの総称である．

形式手法の具体例としては，VDM++[7]や Event-B[8]がある．これらの手法では，一階述語論理を用いて仕様を厳密に記述する．特に Event-B では，段階的詳細化により，抽象的な仕様を具体的な仕様に更新していく．しかし，詳細化が適切に行われているかを検証する作業は，一部手動である上，数学的知識が必要になる．

形式手法における段階的詳細化の枠組みが，論文 Theory of Programs[1]で提案されている．この論文では，集合論を基に仕様とプログラムを統一的に表現する方法を提案している．その方法の中で，詳細化関係の形式化も行っている．しかし，論文で導入されている定理は紙上で証明されており，間違いが混入しがちである．そのため，定理が成り立つかどうかを検証することが望ましい．

３．研究目的，研究目標

本研究の目的は[1]の仕様に関する定理を検証し，形式的な証明を与えることである．加えて，証明された[1]の定理を用いて，詳細化関係の自動検証プログラム作成の事例を示す．また本研究で提案する DSL を利用し，形式仕様記述導入の敷居が下がる．

４．研究内容

4.1. 形式的定理証明

本稿では，[1]の定理の一つである，P22 の検証，証明について述べる．以下に，定理 P22 を示す．

$$P22\ p \subseteq (Pre_p : Havoc)$$

ここで，P22 で用いられている記号，記法について説明する．

仕様/プログラム p は，以下の 3 つで構成される．

- 状態集合 $Set_p \subseteq S$ (S は全体集合)
- 事前条件 $Pre_p \subseteq Set_p$
- 事後条件 $post_p : Set_p \Leftrightarrow Set_p$

仕様/プログラムの中でも，Havoc という特別な名前の付いたものがある．Havoc は，以下の 3 つで構成される．

- 状態集合 $Set_{Havoc} = S$
- 事前条件 $Pre_{Havoc} = S$
- 事後条件 $post_{Havoc} = S \times S$

[1]では仕様/プログラムを扱う演算が紹介されている．その中でも制限と呼ばれる演算について説明する．p を仕様/プログラム，C をある状態集合とするとき，制限演算の演算結果となるプログラム「C : p」は，以下の 3 つで構成される．

- 状態集合 $Set_{C:p} = Set_p$
- 事前条件 $Pre_{C:p} = Pre_p \cap C$
- 事後条件 $post_{C:p} = post_p/C$ (関係の制限[6])

仕様/プログラムの詳細化関係について説明する．仕様/プログラム p1 と p2 について，以下の条件を全て満たすとき，p2⊆p1 が成り立ち，p2 は p1 を詳細化しているという．

- 拡張 $Set_{p2} \supseteq Set_{p1}$
- 弱化 $Pre_{p2} \supseteq Pre_{p1}$
- 強化 $post_{p2}/Pre_{p1} \subseteq post_{p1}$

[1]の定理を一階述語論理で表現し，その否定形が充足不能であるとき，定理が正しいと証明できる．一方，充足可能であった場合は，その解が定理の反例となる．そこで我々は，数理論理学における充足可能性判定プログラムである SMT ソルバ[2]の一つ，Z3[3]の Python 言語への拡張である Z3.py[4]を用いる．

また，[1]で導入されている定義をより簡潔に記述できるよう，Z3.py を用いた DSL[5]を定義することで，可読性の高い検証・証明を可能にした．

この DSL を用いることで，P22 を検証するプログラムは，以下リスト 1 のように記述できる．

リスト１．P22 を検証するプログラム

```
# SMT ソルバのインスタンスを生成
s = Solver()
title = "P22 p ⊆ (Pre_p : Havoc)"
# プログラムのインスタンスを生成
p = prog(s, 'p')
# 定理を記述し，検証を実行
conclude(s, p <= Havoc() / p.pre(), title)
```

リスト１の実行結果を，以下の図１に示す．

図１．リスト１の実行結果

```
P22 p ⊆ (Pre_p : Havoc)
Universe = U, has 3 element(s)
Unholds: sat
set of p_1
 else -> False

pre of p_1
 else -> False

post of p_1
 else ->
        else -> False
```

図１の実行結果は，P22 に反例が存在することを示している．

図１に示された反例から，P22 を成り立たせるのに必要な制約を考案する．図１の結果は，p の状態集合が空集合であるとき，P22 が成り立たないことを意味する．ここで詳細化関係の条件の一つである拡張が，この反例上で成り立つかを調べる．

$$Setp \supseteq Set_{Havoc} \Leftrightarrow \Phi \supseteq S \Leftrightarrow False$$

上記の計算から，Setp が空集合のとき拡張が成り立たないことが分かる．

P22 で拡張を成り立たせるには，Setp ⊇ S である必要がある．しかし，全体集合を包含する集合は全体集合のみであることから，P22 を成り立たせるために必要な制約として Setp ＝S が導かれる．

上記の分析から得た，P22 を成り立たせるための新たな制約 Setp ＝S(=Set_{Havoc})を追加することにした．その新たな制約を追加するプログラムを以下のリスト２に示す．

リスト２．Setp＝Set_{Havoc}を追加するプログラム

```
s.add(eq_set(p, Havoc()))
```

リスト２をリスト１の conclude 関数の前に追加して実行すると反例は生成されなかったことから，追加した制約は P22 を成立させるのに十分であることが示された．

4.2. 事例：DSL を用いた詳細化の記述・証明

以上の検証に用いた DSL によって，簡単なプログラム検証の例を示す．これにより，[1]の実用的な場面での利用可能性を確認する．

[1]では，以下の条件を満たすプログラム p2 は，p1 の契約プログラム(contracted program)と呼び，正しいプログラムであるとしている．

- p2 ⊆ p1
- p2 は実現可能(feasible)
  なお，$Pre_p \subseteq \underline{post}_p$ ($post_p$の定義域)
  であるとき，p は実現可能であるという．

本稿では，乗算を行うプログラムが，契約プログラムになっているかを検証する例を示す．以下のリスト３に，そのプログラムを示す．

リスト３．乗算を行う契約プログラムの例

```
title = "double"
p1, p2 = progs(s, "p1 p2")
s.add(+p1, +p2) # p1, p2 は実行可能
s.add(p1.set() == p2.set(),
      p1.pre() == p2.pre())
x, y = consts('x y', U)
s.add(ForAll([x, y], p1.post(x, y) ==
Or([And(x == i, y == j) for i, j in [
(n1,n2),(n2,n4),(n3,n6),(n4,n8),(n5,n10)
]])))
s.add(ForAll([x, y], p1.post(x, y) ==
Or([And(x == i, y == j) for i, j in [
(n1,n2),(n2,n4)]])))
conclude(s, contracts(p2, p1), title)
```

乗算を行う契約プログラムの検証を整数の範囲で実行したところ，実行結果は unknown となった．そこでリスト３では範囲を制限し，自然数 1，2，…10 を n1，n2,… n10 で表し，p1 が入力値を 2 倍したものを出力値とするプログラム，p2 が 1 を入力したら 2，2 を入力したら 4 を出力するプログラムとして記述されている．リスト３を実行すると，反例は出力されなかった．すなわち p2 が p1 の契約プログラムであることが示されたことになる．

また，p2 の間違った入出力関係として，1 を入力，3 を出力するものを含めて実行すると，反例が出力された．しかし，出力される反例は複雑であり，整数全体の範囲では契約プログラムの判定を行うこともできない．今後の改良，発展が必要である．

５．終わりに

本研究では，[1]における定理検証を行い，DSL を用いた詳細化の記述・証明の事例を例示した．しかし，定理検証の 3 割は修正方法を特定しておらず，実用的な応用についても，簡単なプログラムでさえ，誤った部分を自動で発見するには至っていない．[1]の理論の実用化には，更なる分析と改良が必要である．

【参考文献】

[1] Bertrand Meyer：”Theory of Programs”，Dec.2015

[2] 梅村晃広：”SAT ソルバ・SMT ソルバの技術と応用”，特集ソフトウェア工学，pp. 24-35，Aug.2010

[3] Z3Prover，URL:https://github.com/Z3Prover/z3，Jun.2018 アクセス

[4] Z3API in Python，URL:www.cs.tau.ac.il/~msagiv/courses/asv/z3py/guide-examples.htm，Jun.2018 アクセス

[5] Martin Fowler ． “Domain Specific Languages” Addison-Wesley Educational Publishers 2010

[6] 田中譲ら：”ソフトウェア科学基礎”，近代科学社，Sep.2008

[7] John Fitzgerald et al.， “Validated Designs for Object-Oriented Systems” Springer-Verlag 2005

[8] JeanRaymond Abrial，”Modeling in Event-B system and software Engineering” CAMBRIDGE 2010