仕様記述言語 ZZ と実行時仕様

5N－9

鈴木 修†　小野 康一‡　深澤 良彰†

†早稲田大学理工学部　‡日本アイ・ビー・エム（株）東京基礎研究所

1　はじめに

ソフトウェアがどのような性質・機能を求められているかを厳密に定義するために、形式的仕様が利用される。これまでに提案されてきた形式的仕様の中に、ZF(Zermero-Fraenkel) 集合論を基礎とする仕様記述言語 Z がある。一般には、Z で書かれた仕様を直接実行することはできない。実行できるような Z の仕様には、仕様を実現するための情報が含まれてしまう。しかし、仕様は本来どのような機能を持つのかが記述され、複数の実現を持つべきであり、実現のための情報は仕様中に書くべきでない。

一方、仕様の直接実行や仕様からのプログラムの生成などにより、形式的仕様が要求を満たしているかの確認をする必要がある。しかし、仕様の機能確認には実現のための情報が必要である。この問題点を解決するために定義が不完全な部分関数の記述を許す仕様記述言語 ZZ と実現のための情報 — 実行時仕様 — の導入を提案する。本稿では、実行時仕様の概要とその適用を中心に述べる。

2　仕様記述言語 ZZ

ZZ は仕様記述言語 Z の拡張で、Z 同様 ZF 集合論に基づいている。しかし、Z では不完全な定義の部分関数（一意でない部分関数）を用いた仕様は不完全な仕様と見なされてしまう。これに対して、我々の提案する ZZ では、一意であることが証明されない部分関数を用いて書かれた仕様も不完全なものと見なさない。つまり、ZZ では定義が不完全な部分関数があった場合でも、その部分関数が一意であると見なしているのである。

Formal Specification Language ZZ and its Execute Specification
Osamu Suzuki†, Kouichi Ono‡ and Yoshiaki Fukazawa†
† School of Science and Engineering, Waseda University, Tokyo, 169, Japan
‡ IBM Tokyo Research Laboratory

このことにより、仕様記述者が仕様を記述する際に、全ての部分関数に対して定義を記述する必要がなくなる。すなわち、部分関数の存在だけを仕様中で表現し、その定義は実行時仕様として別個に与えるのである。つまり、仕様にはソフトウェアに求められる機能だけが記述されるのである。

3　実行時仕様

これまでにも ZF 記法からのプログラムへの変換がなされてきた。しかし、従来の方法では、仕様が実現のための情報を持ってしまい、仕様が高い抽象度を保つことを妨げているか、暗黙の仮定のもとでプログラム生成されるので、生成されるプログラムに設計者の意図が反映されないという問題を引き起こしていた。この問題の原因は仕様を一種のプログラムと見なしていたことで、仕様だけから生成しようとしたために生まれたものである。この問題を解決するには実行時仕様の導入が効果的であると考えられる。なぜなら、実行時仕様を導入することにより、仕様の記述は高いレベルを保つことができる。また、実現にあたり一つの仕様から複数の実行時仕様を得ることができ、その中から一つの実行時仕様を選ぶことができる。このことは、実現されるプログラムに設計者の意図が反映されることを意味する。

ZZ において、仕様中で不完全なままもちいられた部分関数の定義は外部から定義を与える。これに、仕様中の変数の対象領域に関する情報を与えるものが実行時仕様である。実行時仕様は、仕様に対して複数存在する実現を一つに特定する。実行時仕様の記述には、抽象データ型の仕様記述を採用した。抽象データ型の仕様は等式論理で記述される。実行時仕様によって、未定義な部分関数の定義や変数の対象領域の定義が与えられる。これらの定義を与えることは、仕様に書かれている機能を実現することと等しい。ZZ と実行時仕様の導入したソフ

トウェア開発では仕様として必要な情報と実現に必要な情報とを分離して扱うことができる。実現のための情報を仕様から分離させることにより、仕様記述の際には実現のことを意識せずに仕様を記述でき、実行する必要が生じたときに目的に応じた実現を選択することができる。現実の設計の場合でも、仕様記述の際に実現を考えた仕様を書くわけではないので、我々の考えは妥当なものである。

図1にZZによる仕様の例を示す。この仕様では述語 order の定義がなされてないので、このままでは実現できない。この述語 order は二つの要素に順番をつけるということだけを示していて、このままでは実現できない。従って、順序付けを実行時仕様で与えなければならない。順序付けには様々な方法があるが、ここでは普通の数の大小関係をもって述語 order の定義とする。この場合の実行時仕様を図2に示す。このように、実行時仕様を与えることによって、Order の仕様は実現することが可能になる。

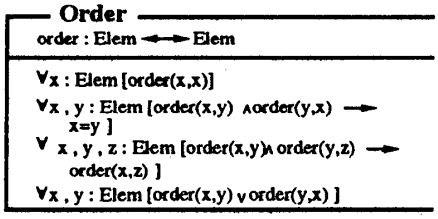

図 1: ZZ による仕様の例

```
executespec Order
using
 Set,Bool
var
 x,y : Elem
equation
 order(x,y) ≡ x ≤ y
```

図 2: 図1に対する実行時仕様

4 評価

ここではZZによる仕様記述を行ない、それに対する実行時仕様を記述することよって、本研究の有効性の評価をする。表1では一つの仕様から書くことのできた、実行時仕様の個数を示している。この表からも分かるように、一つの仕様から複数の実行時仕様を得ることができた。複数存在する実行時仕様のいずれを用いても、仕様に書かれた条件を満たすアルゴリズムを導くことができた。また、ある程度高いレベルの仕様から得られた実行時仕様に対して、不完全な定義の部分関数に対応するプログラム断片を得ることができ、そのプログラムは設計者の意図を反映したものになった。このように、実現のことを気にせずに書いた仕様に対し、複数の実行時仕様が書けて、複数のプログラムを実現できることが確認されたので、本手法の有効性が確認できた。

表 1: 各種問題に対する適用

問題	実行時仕様
スタック	1
キュー	1
列シフト	2
オープンハッシュ	2
順序	3
Fixone ソート	2
クイックソート	2

結局、従来までの方法の問題点は二つとも解消することができたが、問題点もあり、実現のために実行時仕様の記述に制限が加わり、実現を考えた実行時仕様の記述が必要となってしまった。また、ZZによる仕様の記述が抽象的過ぎると、実行時仕様を与えることが容易ではなくなってしまうという問題点も生じてしまった。

5 おわりに

仕様中に書くと実現のための情報となり、仕様としては冗長なものとなってしまう部分関数の定義を、仕様と分離して実行時仕様として記述することを提案した。
一般的に一つの仕様に対する実現は複数存在する。ゆえに、一つの仕様に対して実行時仕様も複数存在する。そこで、複数の実行時仕様により、機能確認や性能評価を行ない、実現の際の決定を支援することができると考えられる。本手法を改良していくことで、実際のソフトウェア開発の際にも有効になると思われる。

参考文献

[1] 福田 剛志、深澤 良彰、門倉 敏夫. 設計上の決定＝プログラム - 仕様. ソフトウェア工学,pp. 119-124, Feb.1991. プログラムの合成.bit,Vol.16,No.6,pp. 723-729.