ソフトウェア紹介

仕様記述言語Zと証明環境Isabelle/HOL-Z

来間 啓伸　Burkhart Wolff　David Basin　中島 震

1 はじめに

Z 記法 (以下 Z) は形式手法の代表例として常に紹介される形式仕様言語である．ソフトウェアの仕様を数学的な意味を持つ記法で厳密に表現することを目的として，1980 年代に Oxford 大学が中心となって開発された．その名称からわかる通り，当初はシステム仕様の記法として用いられた．代表的な事例として，Oxford 大学と IBM 社が行った IBM CICS の仕様記述がある．この成功によって，1992 年に英国の Queen's Award for Technological Achievement を受賞した．その後，VDM-SL と同様に ISO 標準となった [15].

Z は仕様記述に主眼が置かれていたが，Z の意味論を高階論理 (HOL) によって表現することが可能である．その結果，Z で書かれた仕様記述を解析するツールを作成することができる．たとえば，ProofPower [20] ならびに本稿で紹介する Isabelle/HOL-Z [4] などがある．このようなツールを用いることで，仕様の性質やリファインメント関係の検証を行うことが可能になった．

以下，第 2 節で Z の概要を紹介する．第 3 節で有名な誕生日帳の問題 [10] を用いて，Z の記述とリファインメントの例を紹介する．第 4 節で Isabelle/HOL-Z 3.0(以下 HOL-Z) [17] を紹介し，第 5 節で Z 関連の話題を整理する．

2 Z の概要

Z はモデル規範型の形式仕様言語であり，対象システムを適切な抽象度で表現するための言語要素を提供する．公理的な集合論 (Zermelo 集合) と一階述語論理に型概念を導入した体系に基礎をおく．Z を用いる場合，対象システムを状態遷移システムとして表現することが多い．すなわち，操作あるいは手続きを状態の変化を表す数学的な関係として表現する．また，記述を容易にするためにいくつかのイディオムを提供している．特に,「スキーマ」と呼ぶ考え方によって関連する述語をまとめて表現することが特徴である．状態空間を定義するスキーマ，初期状態を定義するスキーマ，状態の変化を表す操作スキーマなどに分類できる．

Z は，当初，B メソッドで著名な Jean-Raymond Abrial 氏らによる「集合論を仕様記述の道具に使う」という考え方 [1] に影響を受けたという．特に，公理的集合の要素に対応する記法を整理したものであり，そのために，仕様記述言語といわず,「Z 記法」と呼んだ．J.M.Spivey による意味論 [9] はこの考え方に基づくものである．

その後，Z の意味論ならびに推論規則に関する研究が進み，現在では，形式仕様記述言語として確立したといえる．厳密な意味論は ISO の標準勧告文書 [15] にある．また，J. Woodcock の教科書 [13] にはリファインメントに基づく形式証明技法が整理されている．

3 Z の記述例

本節では，Spivey による誕生日帳 (birthday book)

The Z Specification Language and the Proof Environment Isabelle/HOL-Z.

Hironobu Kuruma, 株式会社日立製作所, Hitachi, Ltd.
David Basin, Burkhart Wolff, ETH Zurich.
Shin Nakajima, 国立情報学研究所, National Institute of Informatics.

コンピュータソフトウェア, Vol.24, No.2 (2007), pp.21–26.
[ソフトウェア紹介] 2007 年 1 月 26 日受付.

の例題[10]を用いて，Z の基本的な使い方を解説する．

3.1 誕生日帳

誕生日帳には，名前と誕生日が関係付けられて格納される．そこで，はじめに名前の集合と日付の集合[†1]を宣言する．

$$[NAME, DATE]$$

先に述べたように，誕生日帳システムを遷移システムとして形式化する．次のスキーマ *BirthdayBook* は，システムの状態空間を宣言する．

BirthdayBook

$$known : \mathbb{P}\, NAME$$
$$birthday : NAME \pfun DATE$$
$$known = \mathrm{dom}\, birthday$$

このスキーマは，システムに登録されている名前を表す状態変数 *known* と，名前を誕生日に関係付ける部分関数 *birthday* を宣言する．スキーマの述語部は，これらの変数の間の不変条件を定める．ここでは，「システムに登録されている名前の集合と *birthday* の定義域は一致すること」が不変条件である．

遷移システムを定義するため，はじめに初期状態を定める．

$$InitBirthdayBook == [BirthdayBook \mid known = \emptyset]$$

スキーマ *InitBirthdayBook* は，*BirthdayBook* の *known* 変数を初期値の空集合に束縛する．不変条件から，*birthday* の初期値も空集合となる．

次に，状態変化を操作スキーマに記述することで，遷移関係を定義する．例えば名前と誕生日をシステムに追加する *AddBirtyday* は，次のようになる．

AddBirthday

$$\Delta BirthdayBook$$
$$name? : NAME;\ date? : DATE$$
$$name? \notin known$$
$$birthday' = birthday \cup \{name? \mapsto date?\}$$

ここで，「Δ」は既定義のスキーマを取り込むことを示し，*BirthdayBook* の遷移前 (事前状態) と遷移後 (事後状態) の状態を表す二つのコピーを作る．事後状態の状態変数は，「′」を付けることで区別する．記号「?」を後置した変数は入力変数を表す．したがってこのスキーマは，「誕生日帳の事前状態と入力 *name*? と *date*? が与えられたとき，入力 *name*? が *known* に登録されていないならば，事後状態では部分関数 *birthday* は *name*? と *date*? の関係付けを追加したものに更新されること」を表す．

3.2 リファインメント

リファインメントは，系統的なプログラム開発方法の一つである．ここでは，誕生日帳の例を使ってデータリファインメントを説明する．データリファインメントでは，はじめに抽象的なデータを操作する抽象的なシステムを定義しておき，いくつかの詳細化段階を通して，より具体的なデータ表現に作用する操作に設計しなおす．このため，どのようなデータ表現を使って情報を表現するかの決定を設計の上流工程では行わず，下流のアルゴリズム設計の段階まで遅らせることができる．

このような具体的なデータ表現の例を，誕生日帳を使って示す．部分関数 *birthday* を，具体化したシステムでは *names* と *dates* の二つの配列を使って実現する．配列は自然数から値への関数として表現し，変数 *hwm* を導入して既に使用済みの配列領域の最大値を管理する．

*BirthdayBook*1

$$names : \mathbb{N} \to NAME$$
$$dates : \mathbb{N} \to DATE$$
$$hwm : \mathbb{N}$$
$$\forall i, j : 1..hwm \bullet i \neq j \Rightarrow names(i) \neq names(j)$$

さらに，初期化スキーマを次のように定義する．

$$InitBirthdayBook1 == [BirthdayBook1 \mid hwm = 0]$$

次の *AddBirthday*1 は，*AddBirthday* をより具体化したものである．

†1 これらの集合は，型検査の際の基本型となる．

*AddBirthday*1

$\Delta BirthdayBook1$

$name? : NAME;\ date? : DATE$

$\forall i : 1..hwm \bullet name? \neq names(i)$

$hwm' = hwm + 1$

$names' = names \oplus \{hwm' \mapsto name?\}$

$dates' = dates \oplus \{hwm' \mapsto date?\}$

直感的には，hwm はシステムに格納された名前と誕生日の対の数を表し，このような対は配列 $names$ と $dates$ のインデックス $i \in \{1..hwm\}$ に格納されている．リファインメントの検証の目的は，具体化されたシステムを表すスキーマが，この関係を満たすことを証明することである．

データリファインメントは，Hoare によって提案された[7]．その後の詳細な研究を経て (例えばサーベイ[8] を参照)，Z や B メソッドのような開発手法の重要な要素となっている．データリファインメントの本質は，一方のデータ構造が他方を模倣することにある．いろいろな模倣の方法が文献で紹介されており (上記サーベイを参照)，それらは Z でも使うことができる．以下の例では前方 (下方とも呼ばれる) シミュレーションを使い，文献[10] にしたがって説明する．

$Astate$ と $Cstate$ を，それぞれ抽象的なデータ表現と具体化したデータ表現のデータ空間を表すものとする．各データ表現は，それぞれの状態空間上に定義された遷移システムに対応する．したがって，Z ではデータ表現は以下のように定義される．

- 正しい初期状態を定義するスキーマ．以下では，それぞれ $Ainit$ と $Cinit$ と呼ぶ．
- 操作スキーマ．それぞれの状態空間上の関係を定義するもので，Aop と Cop のように呼ぶ．

前方シミュレーションの証明は，具体的なデータ表現での遷移が抽象的なデータ表現での遷移によって模倣されることを示す．このような証明のためには，具体化した状態と抽象的な状態の間の関係を与える抽象化スキーマ (模倣関係とも呼ばれる)Abs を定義しておく必要がある．誕生日帳の場合には，以下のようになる．

Abs

$BirthdayBook$

$BirthdayBook1$

$known = \{i : 1..hwm \bullet names(i)\}$

$\forall i : 1..hwm \bullet$

$\quad birthday(names(i)) = dates(i)$

これは，先に述べた直感的な対応関係を形式化したものである．

3.3 リファインメントの検証

模倣を証明するためには，二つのシステムの初期状態を Abs のもとで関連付け，さらにあらゆる遷移の結果も Abs のもとで関連付ける必要がある．はじめに一般的な証明責務を示して抽象的に説明し，その後で具体的な誕生日帳の例を示す．

初期状態を関連付けるためには，具体的なデータ表現の初期状態が，抽象的なデータ表現の初期状態のどれかを (Abs のもとで) 表現することを示さなければならない．これは，次の式で表される．

$$\forall Cstate \bullet Cinit \Rightarrow \exists Astate \bullet Ainit \wedge Abs \quad (1)$$

システムの遷移を関連付けるためには，各操作が次の二つの条件を満足することを示す必要がある．一つは，Aop の事前条件を満たすあらゆる状態に対して，Abs のもとで対応する状態に関して Cop が定義されていることである．これは，次のように形式化される†2．

$$\forall Astate;\ Cstate \bullet pre\ Aop \wedge Abs \Rightarrow pre\ Cop \quad (2)$$

ここで，「pre」はスキーマで表現された操作の事前条件を返すものとする．二つめの条件は，具体的な操作の結果得られる状態が，対応する抽象的な操作の結果得られる状態のいずれかと対応していることである．

$$\forall Astate;\ Cstate;\ Cstate' \bullet$$
$$pre\ Aop \wedge Abs \wedge Cop \Rightarrow\ (\exists Astate' \bullet Abs' \wedge Aop) \quad (3)$$

Abs が与える対応関係が，具体的な状態から抽象的な状態への関数 (一つの具体的な状態に一つの抽象的な状態が対応付けられる) である場合には，条件 (3)

†2 簡単のため，入出力はないものとした．

は次のように簡単化できる．

$$\forall Astate;\ Astate';\ Cstate;\ Cstate' \bullet \\ pre\ Aop \land Abs \land Cop \land Abs' \Rightarrow Aop \quad (4)$$

このようなリファインメントは，関数的リファインメントと呼ばれる．誕生日帳の場合も，関数的リファインメントである．

次に，上で述べた証明責務を証明するのに必要な推論の例を挙げる．例えば，*AddBirthday* について条件 (4) を証明することは，次の式 (これ一つではないが) を証明することを意味する (*AddBirthday* の述語部より)．

$$birthday' = birthday \cup \{name? \mapsto date?\} \quad (5)$$

前提と，集合に関する基本的な性質から，次のように変形できる．

$$\begin{aligned} & \mathrm{dom}\ birthday' \\ = \ & known' \\ = \ & \{i : 1..hwm' \bullet names'(i)\} \\ = \ & \{i : 1..hwm \bullet names'(i)\} \cup \{names'(hwm')\} \\ = \ & \{i : 1..hwm \bullet names(i)\} \cup \{name?\} \\ = \ & known \cup \{name?\} \\ = \ & \mathrm{dom}\ birthday \cup \{name?\} \end{aligned}$$

配列 *names* は添え字 $i \in 1..hwm$ については値が変化しないことから，

$$birthday'(names'(i)) = birthday(names(i))$$

を得る．また，新たに追加した *name?* について，

$$\begin{aligned} birthday'(name?) &= birthday'(names'(hwm')) \\ &= dates'(hwm') \\ &= date? \end{aligned}$$

である．したがって，(5) が成り立つことが示せた．

4 HOL-Z を使った検証

4.1 HOL-Z

HOL-Z[17] は，Z で書かれた仕様のための証明環境であり，汎用の定理証明系 Isabelle[18] を基礎とする．論理的には，HOL-Z は Isabelle/HOL への Z の埋め込み (embedding) である．Z に特化した証明手続きは，この埋め込みによって Isabelle フレームワークに統合され，ISAR コマンド言語[12] を通じて使うことができる．

図 1 に示すように，HOL-Z では仕様記述過程を二つの段階に分けている．モデル化段階では，設計者は対象システムのモデルを Z で記述する．記述したモデル (LaTeX ドキュメント) は，ZETA ツール[21] によって型検査された後，検証段階に渡される．検証段階では，証明技術者が ISAR コマンドを使って証明責務を証明する．定理証明過程は証明ドキュメントに記録される．HOL-Z は，「文芸的仕様記述 (literate specifications)」をサポートしている．すなわち，モデル化段階や検証段階では形式仕様や証明に非形式的な説明を付けることができ，HOL-Z は証明をチェックした後に非形式的な説明を含む最終ドキュメントを生成する．もちろん，モデル化を行う設計者と，証明を行う証明技術者は，同一であっても良い．両者に求められる能力が異なるので，各々が自らの作業に専念できるよう段階を分けているのが，HOL-Z の特徴の一つである．

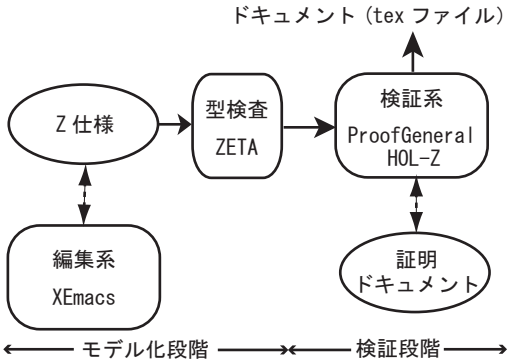

図 1 HOL-Z を使った仕様記述／検証過程

4.2 HOL-Z を使った検証の概要

HOL-Z は，証明ドキュメントを対話モードでもバッチモードでも処理することができる．対話モードでは，ProofGeneral[19] と呼ばれる Emacs の特定のモードを使って，証明ドキュメントを一行一行たどって証明を進めることが可能である．以下では，誕生日帳の検証の主なステップについて述べ，検証過程の様子を示す (完全な証明は[17] を参照)．

まず，ZETA を使って型検査したモデルをロードする．

```
load_holz "BBSpec"
```

この結果，全ての定義と，定義から自動的に生成された簡単化規則が利用可能になる．

次のコマンド

```
set_abs "Abs" [functional]
```

は，HOL-Z に *Abs* スキーマが抽象化スキーマであることを知らせる．ここでは，`functional` オプションを使って関数的リファインメントであることも知らせている．このことで，関数的リファインメントの証明責務を生成するモードに設定される．生成する証明責務の中には，関数的であることの証明も含まれる．

証明責務の生成は，次のコマンドで行う．

```
refine_op AddBirthday Addbirthday1
```

これは，*AddBirthday* について二つの証明責務 (2) と (4) を生成し，HOL-Z のデータベースに格納する．データベースの内容は，コマンドによって表示することができる．例えば生成された証明責務 (4) の具体的な内容は，

```
BBSpec_functional.fwRefinementOp_AddBirthday_2
```

という名前で参照できる．

次のコマンド

```
po BBSpec_functional.fwRefinementOp_AddBirthday_2
```

は，指定した証明責務の証明過程に入ることを指示する．HOL-Z は，下記の内容を出力して，証明状態を初期化する．

$$\forall\, BirthdayBook \bullet (\forall\, BirthdayBook' \bullet (\forall\, BirthdayBook1 \bullet (\forall\, BirthdayBook1' \bullet (\forall\, date?; name? \bullet preAddBirthday \land Abs \land AddBirthday1 \land Abs' \Rightarrow AddBirthday))))$$

HOL-Z を使った証明とは，Isabelle/HOL と HOL-Z の証明コマンド (詳細は [12] を参照) を使って証明状態を真にすることである．証明コマンドでは，簡単な規則の適用だけでなく，自動化された証明手続きの適用も指示することができる．

証明が終了したら，

```
discharged
```

コマンドによってツールに指示し，証明責務をデータベースから削除する．

4.3 議論

Z は HOL と同等の表現力を持つ言語であり，仕様を簡潔に表現することができる．その反面，モデル検査の場合と違って，決定手続き (decision procedure) が存在せず，仕様の解析は基本的に人の介入を要する定理証明に依らざるを得ない．

しかし，HOL-Z を使った証明は教科書的な人手による証明とは異なる様相を持つ．例えば，証明ドキュメントは証明コマンドを使って証明状態をどのように変換するかの記述であり，証明状態が人間によってどのように解釈されるかに重点を置いたものではない．さらに，教科書的証明では通常省略される詳細を取り扱わなければならない．例えば，3.3 節の証明では $birthday \cup \{name? \mapsto date?\}$ が部分関数であることが暗黙の前提として使われているが，HOL-Z を使った証明ではこの条件を含め全ての暗黙の前提をたどる必要がある．それにもかかわらず，3.3 節の議論に対応する証明ドキュメントの長さは，同じ (ともに 21 行) であった．

Z で記述した仕様を証明環境を使って解析することは，どのくらいの効果があるものだろうか．電子署名のクライアントサーバアーキテクチャの設計検証 [3] では，経験ある証明技術者が行ったところ，検証にかかる時間や労力がモデル検査に比べて極端に大きくなることはなかった．その一方で，モデルや検証すべき性質が適切に設定されているかどうかに関して，モデル検査よりも多くの情報が，証明過程で示される反例を通して得られた．

5 Z の周辺

Z は，主として仕様記述言語と，解析の基盤の二つの方向で使われている．仕様記述言語の点からは，プログラム開発への第一歩として，要求や設計を正確に文書化し，コミュニケーションの手段を提供する．例えば CICS プロジェクトでは，Z で書き表すことによって自然言語を用いた仕様書の問題点を顕在化させることができ，改版時の設計にフィードバックできたという [6]．さらに，長期間にわたる保守用ドキュメントとしても有用であったと報告されている．他にも，広く使われているオペレーティングシステムの一部を形式的に記述した例 [5] が挙げられる．

Z は，曖昧性なく解釈できることが要求される標

準勧告文書でも使われてきた．ITU-T から発行された ODP トレーダ [16] や ANSI の Role-Based Access Control [14] において，基本的な操作の機能仕様を表現するために Z が用いられている．ところが，このような標準化文書の策定作業に関わる形式手法の専門化が少ないことが原因であろうか，Z の記述部分に誤りが散見されることは，公式の標準勧告文書としては残念なことである．逆に，Z に精通していれば容易に誤りが分かるので，逆説的であるが Z を用いた効果とも言える．

Z は，形式的な検証の基盤としても使われている．前述の電子署名アーキテクチャの設計検証は，この一例である．さらに，リファインメントに基づくプログラム開発への Z の適用も，4 節に示したように適切なツール支援があれば可能であろう．

6 おわりに

Z は 25 年以上の歴史を持ち，研究者にも産業界の技術者にも共に使われる，しっかりした基礎を持つ仕様記述言語である．ZETA や ZTC のような型チェッカ，Jaza や ZAP のような Z のサブセットに対して仕様実行を行うアニメータ，HOL-Z のような高階論理への埋め込みに基づく定理証明系など，Z のためのツールは数多くある ([22] 参照)．

本稿では，簡潔で直感的な仕様記述，仕様上でのシステムの性質の解析，リファインメントを使ったプログラムの開発，の点から Z の多様な使い方の可能性を紹介した．システムの仕様を形式仕様記述言語を使って書くことの効果は大きいが，形式仕様記述言語に基づいて仕様を解析することのメリットについても強調しておきたい．なお，Z については単行本 [10] [13] などが良い入門書である．日本語でも解説 [2] [11] があるので，参考にされたい．

参 考 文 献

[1] Abrial, J.-R., Schuman, S.A. and Meyer, B.: Specification Language, *On the Construction of Programs*, McKeag, R. M. and McNaghten, A. M. (eds.), Cambridge University Press, 1980.

[2] 荒木啓二郎, 張漢明 : プログラム仕様記述論, IT Text, オーム社, 2002.

[3] Basin, D., Kuruma, H., Miyazaki, K., Takaragi, K. and Wolff, B.: Verifying a Signature Architecture: A Comparative Case Study, *Formal Aspects of Computing* (to appear).

[4] Brucker, A. D., Rittinger, F. and Wolff, B.: HOL-Z 2.0: A Proof Environment for Z-Specifications, *Journal of Universal Computer Science*, Vol. 9, No. 2 (2003), pp. 152–172.

[5] Brucker, A. D. and Wolff, B.: A Verification Approach to Applied System Security, *International Journal on Software Tools for Technology Transfer*, Vol. 7, No. 3 (2005), pp. 233–247.

[6] Hayes, I. (ed.): *Specification Case Studies*, Prentice Hall, 1993.

[7] Hoare, C. A. R.: Proof of correctness of data representation, *Acta Inf.*, Vol. 1 (1972), pp. 271–281.

[8] de Roever, W.-P. and Engelhardt, K.: *Data Refinement Model-Oriented Proof Methods and their Comparison*, Cambridge University Press, 1998.

[9] Spivey, J.M.: *Understanding Z*, Cambridge University Press, 1988.

[10] Spivey, J.M.: *The Z Notation: A Reference Manual* (2ed.), Prentice Hall, 1992.

[11] 玉井哲雄: ソフトウェア工学の基礎, 岩波書店, 2004.

[12] Wenzel, M.: The Isabell/Isar Reference Manual, TU München, 2005.

[13] Woodcock, J. and Davies, J.: *Using Z: Specification, Refinement, and Proof*, Prentice Hall, 1996.

[14] Role-Based Access Control, American National Standards Institute, Inc., 2004.

[15] ISO/IES 13568, Information Technology – Z Formal Specification Notation –Syntax, Type System and Semantics, 2002.

[16] ITU-T Rec. X.950-1, Information Technology – Open Distributed Processing – Trading Function – Part 1: Specification, 1997.

[17] The HOL-Z home page, `http://www.brucker.ch/projects/hol-z`.

[18] Isabelle, `http://isabelle.in.tum.de`.

[19] Proof General, `http://proofgeneral.inf.ed.ac.uk`.

[20] The ProofPower Web Pages, `http://www.lemma-one.com/ProofPower/index`.

[21] The ZETA System, `http://uebb.cs.tu-berlin.de/zeta`.

[22] The World Wide Web Virtual Library: The Z Notation, `http://www.zuser.org/z`, (September 2005).