福山大学工学部紀要
第27巻 2003年12月

形式仕様記述に基づく検証の自動化

石川 洋*

On an Automatic Verification based on Formal Specifications

Hiroshi ISHIKAWA*

ABSTRACT

We presents an approach to reason about specifications written in a formal specification language automatically. Our reasoning approach is based as follows: (1)Specifications are written in Z, one of formal specification languages. (2)Reasoning system is based on $\mathcal{W}$, a logic for the Z notation. (3)Verification system is written using CafeOBJ. The logic $\mathcal{W}$ is similar to a sequent calculus in the style of Gentzen. CafeOBJ is an executable algebraic specification language. Therefore our approach attempts to transform the specifications into sequents. This paper gives an example how to translate the Z notations into sequents and how to prove using them.

キーワード：形式仕様，Z，代数仕様言語，検証

Keywords: Formal Specification, Z, Algebraic Specification Language, Verification

1. はじめに

ソフトウェアシステム開発の上流工程では，要求仕様を厳密に記述することが期待されている [6]．厳密な仕様を記述する方法の一つに形式仕様言語の利用があげられる．ここでいう厳密性とは，数学的概念や記法を利用し，曖昧性を排除することを想定している．このとき，厳密な仕様に基づく検証作業は，数学的な定義と論理に基づいた推論作業と考えることができる．形式仕様言語は，一般的に，仕様を記述することに重点が置かれており，記述された仕様を実行する処理系を必ずしも備えてはいない．したがって，検証作業は手作業で行わなければならず，効率的とはいえない．

本稿では，形式仕様言語によって記述された仕様に基づく検証作業を自動化するための一方法を提案する [8] [9]．本稿で対象とする形式仕様言語は，集合論と一階述語論理を基礎とする Z とする．

形式仕様言語 Z [17] は高い表現力を持っているが，それによって記述された仕様を検証するためには，前述の通り，人間が地道に演繹規則に従って計算しなければならない．ソフトウェアの規模が増大化し，構造が複雑化すると，もはや人手で処理することは困難である．この問題を解決するために，Z で記述された仕様 (以降，Z 仕様と呼ぶ) を実行可能な処理系で動作可能な記述に変換し，コンピュータ支援による検証を行なうというアプローチがいくつか提案されている [1] [2] [10] [12] [14] [18] [19]．しかしこれらのアプローチはまだ試作段階であり，まだ実用化には至っていない．

本稿では Z 仕様を推論するための論理体系の利用を想定し，Z 仕様の検証作業を実行可能な処理系を持つ代数仕様言語 CafeOBJ [3] [16] を利用して自動化する方法について提案する．このアプローチ [19] はすでに提案されているが，Z 仕様から代数仕様への変換作業は手動で行っていた．この場合，大規模な仕様を扱う際，変換作業に時間を要する，変換ミスを誘発するといった問題が生じる．その問題を解決するための一案として，Z 仕様から代数仕様への変換部分の自動化を提案する．

本稿は次のような構成となっている．2 節では本研究で扱う諸概念の概要を述べる．3 節では，Z 仕様を代数仕様に変換する方法について述べる．4 節では変換され

* 情報処理工学科

た仕様に基づいた検証例を紹介する．5節で関連研究を紹介し，最後にまとめと今後の課題を述べる．

2. 諸概念の概要

本節では，形式仕様言語 Z，Z 仕様を推論するための論理体系 $\mathcal{W}$，および代数仕様言語 CafeOBJ の諸概念について概略を述べる．

2.1 形式仕様言語 Z

Z [17] は集合論と一階述語論理に基づいた，スキーマと呼ばれる記述を単位とする表現能力の高い形式仕様言語である．Z は仕様を記述することに重点を置いており，実行可能な処理系は備えていない．

Z では仕様化する対象を，とりうる状態の集まり，初期状態，および状態に適用する操作の 3 種類で表現しており，それぞれを，状態スキーマ，初期状態スキーマ，および操作スキーマとして記述する．スキーマはスキーマ名，宣言部，述語部から構成される．例として誕生日帳の仕様 [17] を Z で記述する．誕生日帳は，人名と日付の組で構成される．誕生日帳に新たな情報を追加する場合は，人名が既に登録されているかどうかを確認し，登録されていなかった場合にのみ誕生日帳に登録する．以下の記述は Z 仕様記述用のスタイルファイル [11] を用いている．

$[NAME, DATE]$

BirthdayBook

$known : \mathbb{P}\, NAME$

$birthday : NAME \pfun DATE$

$known = \mathrm{dom}\, birthday$

AddBirthday

$\Delta BirthdayBook$

$name? : NAME$

$date? : DATE$

$name? \notin known$

$birthday' = birthday \cup \{name? \mapsto date?\}$

上記の記述は上から順に，基本型の宣言，誕生日帳の状態宣言，誕生日帳にデータを登録する操作の宣言である．

基本型の宣言では，誕生日帳で使用するオブジェクト (データ) の集合名を与えている．これは，通常のプログラミングで例えると型宣言にあたる．

誕生日帳の状態宣言では，宣言部において，名前の集合 *known* と名前と日付の組の集合 *birthday* を，述語部では，それらの集合の関係を宣言している．

誕生日帳にデータを登録する操作の宣言では，宣言部において，データを登録する前の誕生日帳の状態 *BirthdayBook* と登録後の状態 *BirthdayBook′* を同時に表現する $\Delta BirthdayBook$，登録しようとする名前 *name?* と日付 *date?* を与えている．述語部では，名前が登録済みでないことを確認したのちに名前と日付の組を誕生日帳に登録するという操作が定義されている．

2.2 Z の論理体系 $\mathcal{W}$

Z の記述に基づいた検証を行うための枠組が幾つか提案されているが，そのなかの一つに，Z の論理体系 $\mathcal{W}$ [13] がある．Z の論理体系はシーケントと推論規則で構成される．シーケントとは前件と後件の組 (前件⊢後件) であり，前件は宣言と述語のリストの組 (宣言,···, 宣言 | 述語,···, 述語)，後件は述語のリスト (述語,···, 述語) で構成される．

推論規則は，前提から結論を導く規則で，前提はシーケントのリスト，結論はシーケントとして与えられる．

$$\frac{\text{前提}}{\text{結論}}\ \text{[規則名][適用条件]}$$

適用条件は，前提や結論に現れる自由変数や属性名についての決定可能な条件である．恒真 (valid) な前提から valid な結論が導かれるとき，推論規則は健全 (sound) であるという．前提が空である健全な推論規則を定理と呼ぶ．

Z の論理体系には，推論規則を簡略化するためのメタ定理がある．

定理 (リフティング) [13]

推論規則

$$\frac{E, D \mid \Psi \vdash \Phi}{E, D' \mid \Psi' \vdash \Phi'}$$

が健全ならば，推論規則

$$\frac{F, E, D \mid P, \Psi \vdash Q, \Phi}{F, E, D' \mid P, \Psi' \vdash Q, \Phi'}$$

もまた健全である．ただし，

$$(\alpha D \cup \alpha D') \cap (\phi P \cup \phi Q) = \varnothing.$$

ここで，ϕ は与えられた引数に現れる自由変数の集合を返す関数であり，α は与えられた引数に現れる属性名の集合を返す関数である．

定義終

このメタ定理により，推論規則の適用によって変化しない宣言や述語を推論規則の記述から省くことが許される．例えば，以下の推論規則

$$\frac{D' \mid P' \vdash Q'}{D, D' \mid P, P' \vdash Q, Q'}\ [thin]$$

$$\frac{D \mid P, R \vdash Q \quad D \mid P \vdash Q, R}{D \mid P \vdash Q}\ [cut]$$

はリフティングを用いるとつぎのように記述できる．

$$\frac{\vdash}{D \mid P \vdash Q}\ [thin]$$

$$\frac{R \vdash \quad \vdash R}{\vdash}\ [cut].$$

ここで，*thin* は不必要な宣言部や述語部を取り除く推論規則，*cut* は証明に補題を導入する推論規則を意味する．

2.3 代数仕様言語 CafeOBJ

実行可能な処理系を持つ代数仕様言語 CafeOBJ [3], [16] は代表的な代数仕様言語 OBJ [4] に等式とは異なる，左辺から右辺への書き換え規則 [15] や，カプセル化されたオブジェクトの状態を扱うことが可能な隠蔽代数 [5] を導入し，拡張した言語である．したがって，CafeOBJ は OBJ が備えている機能であるパラメータ付きモジュールの利用，ソート間の順序の定義，名前を変更したモジュールの参照，および，自由度の高い項の記法といった特徴はすべて継承している．また，新たに導入，拡張された機能や概念により，等式論理の範囲で記述することが困難であった動的なシステムの状態遷移や，実装をブラックボックスと見なしたオブジェクトの記述が可能となった．本稿では CafeOBJ を等式論理の範囲で利用することを想定している．

CafeOBJ での記述単位はモジュールと呼ばれる．モジュールにはソート，演算子，等式などを以下のように宣言する．

```
mod モジュール名 {
  ソートの宣言
  モジュールの輸入宣言
  演算子の宣言
  等式の宣言
}
```

ソートの宣言は通常のプログラミング言語でいう型宣言を意味する．モジュールの輸入宣言は，既に提供されているモジュールを再利用する場合に用いる．これにより仕様記述の細分記述，再利用などが可能になる．

代数仕様においては仕様記述対象はすべて項で表現する．演算子 (オペレータ) の宣言では，項の構成子を定義する．計算は項を変形することで実現されるが，項を変形する規則を等式によって宣言する．等式は左辺から右辺への書き換え規則とみなし，与えられた項に書き換え規則を適用することで仕様を実行していると考える．

3. Z 仕様から代数仕様への変換

Z の推論体系を用いることで Z 仕様に基づく検証が形式的に扱うことが可能となる．これらを代数仕様の表現に変換することで，実行可能な処理系で扱うことができる記述の生成と，検証作業の自動化が可能となる．本節では，Z 仕様を代数仕様に変換するための大まかな方針について述べる．

基本型は通常のプログラミング言語においてはオブジェクトの型に相当するので，ここではソート宣言として扱う．

スキーマはスキーマの名前，宣言部，述語部から構成されていることに着目する．スキーマ名，宣言部および述語部をそれぞれN, D, P と略記すると，スキーマを，スキーマ名と宣言部の組，スキーマ名と述語部の組に分解してそれぞれ< N | D >および[D | P] と表現する．このとき，前件はこれらのリストの組であったので，

```
< N | D >, ... , < N' | D' > |
[ N | P ], ... , [ N' | P' ]
```

のように表現できる．また，後件をCon で表すことにすると，シーケントは，

```
< N | D >, ... , < N' | D' > |
[ N | P ], ... , [ N' | P' ] |- Con
```

と表現できる．これらの表現は代数仕様記述において項で表現する．そこで，代数仕様言語 CafeOBJ では，前述のような項を生成するために以下のようなソート (型) と演算子 (オペレータ) を宣言する．

```
-- ソート宣言
[ Expression, Declaration, SchemaName, Sequent ]
[ Predicate < Predicates ]
[ NamedDec < NamedDecs ]
[ NamedPred < NamedPreds ]

-- オペレータ宣言
op _=_    : Expression Expression -> Predicate
op <_|_>  : SchemaName Declaration -> NamedDec
op [_|_]  : SchemaName Predicate -> NamedPred
op  _,_   : NamedDec   NamedDecs -> NamedDecls
op  _,_   : NamedPred  NamedPreds -> NamedPreds
```

```
op _|_|-_ : NamedDecs NamedPreds
            Predicate -> Sequent
```

ここでソート宣言およびオペレータ宣言について簡単に説明しておく．ソート宣言

```
[ NamedDec < NamedDecs ]
```

では，記号 < を用いて，ソート間の関係を規定している．この場合，ソート NamedDec の要素からなるリストのソートを NamedDecs として宣言している．

オペレータ宣言

```
op <_|_> : SchemaName Declaration -> NamedDec
```

では，スキーマ名と宣言部の記述を記号 <, | および > で囲み，それらで構成される項はスキーマ名付き宣言部 (NameDec) を表現することを規定している．

なお，記号 , が宣言部のリストと述語部のリストの区切り記号として重複使用されているが，CafeOBJ では，演算子の多重定義を許しているので特に問題はない．

また集合演算などの数学的記法を利用するために，以下のような宣言を用意しておく．

```
-- ソート宣言
[ NAME < NAMES ]
[ DATE BIRTHDAYBOOK ]
[ NAMES BIRTHDAYBOOK < SET ]
[ NAME+>DATE ]

-- オペレータ宣言
op {_}        : NAME+>DATE -> SET
op {_}        : NAME -> SET
op _\mapsto_  : NAME DATE -> NAME+>DATE
op _\cup_     : SET SET -> SET
op \dom       : BIRTHDAYBOOK -> NAMES

-- 変数宣言
vars U V : BIRTHDAYBOOK
var N : NAME
var D : DATE

-- 等式宣言
eq \dom ( U \cup V ) = (\dom U) \cup (\dom V) .
eq \dom { N \mapsto D } = { N } .
```

バックスラッシュで始まる演算子は Z 仕様を記述するための LaTeX 用スタイルファイル [11] で提供されている記法であり，代数仕様記述でも同じ形で使用している．これらの記号の意味をこれらの記号の意味を表 1 に示す．

表 1　記号とその意味

Table1 Symbols and their semantics.

\power	巾集合
\pfun	部分関数
\dom	定義域
\notin	非要素
\mapsto	順序対
\cup	和集合

4. 検証例

本節では 2 節で紹介した誕生日帳の例を用い，その中の 2 つのスキーマ宣言から，等式

$$known' = known \cup \{name?\}$$

が成り立つことを，3 節で導入した記法に基づいて示す．ちなみにこの等式は，誕生日帳に新たなデータを追加したとき，誕生日帳の状態スキーマの述語部の式が成立しているか否かを確認するものである．

シーケントカリキュラスでは，推論規則を用いて結論から空式を導き出し，その導出過程は証明図として提示される．したがって，

$$\frac{\vdots}{\text{結論}}$$

のような証明図が得られたらその結論は証明されたことになる．この例においてもこのような証明図が得られることを示す．

まず，所与のスキーマと示すべき等式のシーケントによる表現は次のようになる．

```
< BirthdayBook|known:\power NAME >,
< BirthdayBook|birthday: NAME \pfun DATE >,
< BirthdayBook'|known':\power NAME >
< BirthdayBook'|birthday': NAME \pfun DATE >
< AddBirthday|name? : NAME >,
< AddBirthday|date? : DATE > |
[ BirthdayBook|known = \dom birthday ],
[ BirthdayBook'|known' = \dom birthday' ]
[ AddBirthday|name? \notin known ],
[ AddBirthday|birthday' = birthday
                \cup { name? \mapsto date?}]
|- known' = known \cup { name? } .
```

$$\dfrac{\dfrac{\dfrac{\dfrac{}{D \mid P \vdash known \cup \{name?\} = known \cup \{name?\}}}{D \mid P \vdash \mathrm{dom}\, birthday \cup \mathrm{dom}(\{name? \mapsto date?\}) = known \cup \{name?\}}}{D \mid P \vdash \mathrm{dom}(birthday \cup \{name? \mapsto date?\}) = known \cup \{name?\}}}{D \mid P \vdash \mathrm{dom}\, birthday' = known \cup \{name?\}}}{D \mid P \vdash known' = known \cup \{name?\}}$$

図 1 証明図

Fig.1 Proof Figure

この中の記号のうちバックスラッシュで始まるものは前節で導入した意味を持つ演算子である.

まず, 前件の述語

```
[ BirthdayBook'|known' = \dom birthday' ]
```

を後件の known' に代入すると次のシーケントを得る.

```
< BirthdayBook|known:\power NAME >,
< BirthdayBook|birthday: NAME \pfun DATE >,
< BirthdayBook'|known':\power NAME >
< BirthdayBook'|birthday': NAME \pfun DATE >
< AddBirthday|name? : NAME >,
< AddBirthday|date? : DATE > |
[ BirthdayBook|known = \dom birthday ],
[ BirthdayBook'|known' = \dom birthday' ]
[ AddBirthday|name? \notin known ],
[ AddBirthday|birthday' = birthday
                \cup { name? \mapsto date?}]
|- \dom birthday' = known \cup { name? } .
```

次に, 前件の述語

```
[ AddBirthday|birthday' = birthday
                \cup { name? \mapsto date?}]
```

を後件の birthday' に代入すると次のシーケントを得る.

```
< BirthdayBook|known:\power NAME >,
< BirthdayBook|birthday: NAME \pfun DATE >,
< BirthdayBook'|known':\power NAME >
< BirthdayBook'|birthday': NAME \pfun DATE >
< AddBirthday|name? : NAME >,
< AddBirthday|date? : DATE > |
[ BirthdayBook|known = \dom birthday ],
[ BirthdayBook'|known' = \dom birthday' ]
[ AddBirthday|name? \notin known ],
[ AddBirthday|birthday' = birthday
                \cup { name? \mapsto date?}]
|- \dom(birthday \cup { name? \mapsto date?})
   = known \cup { name? } .
```

後件は演算子\dom の性質により,

```
\dom birthday \cup \dom { name? \mapsto date? }
   = known \cup { name? } .
```

と展開できる.

さらに, 前件の述語

```
[ BirthdayBook|known = \dom birthday ],
```

を後件の \dom birthday に適用すると次のシーケントを得る.

```
< BirthdayBook|known:\power NAME >,
< BirthdayBook|birthday: NAME \pfun DATE >,
< BirthdayBook'|known':\power NAME >
< BirthdayBook'|birthday': NAME \pfun DATE >
< AddBirthday|name? : NAME >,
< AddBirthday|date? : DATE > |
[ BirthdayBook|known = \dom birthday ],
[ BirthdayBook'|known' = \dom birthday' ]
[ AddBirthday|name? \notin known ],
[ AddBirthday|birthday' = birthday
                \cup { name? \mapsto date?}]
|- known \dom { name? \mapsto date? }
   = known \cup { name? } .
```

また, 演算子\dom の性質により,

```
\dom { name? \mapsto date? } = { name? }
```

となるので, シーケントは次のように変化する.

```
< BirthdayBook|known:\power NAME >,
< BirthdayBook|birthday: NAME \pfun DATE >,
< BirthdayBook'|known':\power NAME >
< BirthdayBook'|birthday': NAME \pfun DATE >
< AddBirthday|name? : NAME >,
< AddBirthday|date? : DATE > |
[ BirthdayBook|known = \dom birthday ],
[ BirthdayBook'|known' = \dom birthday' ]
```

```
[ AddBirthday|name? \notin known ],
[ AddBirthday|birthday' = birthday
                \cup { name? \mapsto date?}]
|- known \cup { name? } = known \cup { name? } .
```

これらの推論過程を証明図で表現すると図 1 のようになる．この図中においては，前件の宣言部を D，述語部を P，表 1 にまとめたバックスラッシュで始まる表記については，通常の数学記号を用いている．

5. 関連研究

Z の仕様記述をコンピュータで実行する研究が幾つか報告されている [1] [2] [10] [12] [14] [18] [19]．

本研究は事例 [19] を基礎としている．この事例においては，Z 仕様を手動で CafeOBJ の記述に変換し，検証作業の自動化を試みている (図 2)．Z 仕様の記述量が増えると代数仕様への変換作業は膨大，煩雑になる．Z 仕様を代数仕様へ変換する部分を自動化し，検証作業の効率化を図るのが本研究の目的である．

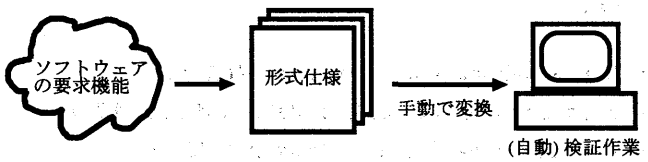

図 2　形式仕様記述に基づく検証作業
Fig.2 Verification based on the Formal Specifications

事例 [1] [12] においては，Z 仕様を自動定理証明器 Isabelle/HOL の記述に変換し，検証作業の自動化を試みている．これらの事例は本研究のアプローチに近い．

事例 [2] [18] では，Z 仕様を論理型言語 Prolog のプログラムに手動で変換し実行している．いずれの事例に置いても，限量子を含むような複雑な論理式を変換の対象とはしておらず，極めて素朴なアプローチであると考えられる．

事例 [10] では，Z 仕様の型宣言をチェックするツール ZTC の入力でもある LaTeX 形式に変換された Z 仕様を読み込み，対話的に実行するシステムを提案してる．このアプローチにおいては，Z 仕様を実行するインタプリタを実装しており，本研究のような Z 仕様を他の仕様言語記述に変換するものではない．

事例 [14] も上記と同様に独自の処理系を提供しており，Z 仕様の分析ツールでもあって，文法チェック，型の正当性，スキーマの展開，事前条件の計算，洗練の証明 (refinement proofs)，定理証明などが可能なシステムである．

6. まとめと今後の課題

最後に本稿のまとめと今後の課題について述べる．

6.1 まとめ

本稿では，まず形式仕様記述に基づいた検証作業の重要性と問題点について述べている．上流工程における仕様の検証作業は，要求仕様を明確にし，その後の作業で生じる可能性のある手戻りを極力さけるために必要である．要求仕様が多ければ検証作業も煩雑，複雑となり，作業の効率化が求められる．

本研究では，形式仕様記述に基づく検証作業の具体例とてして形式仕様言語 Z で記述された仕様に基づいた検証作業を実行可能な処理系を持つ代数仕様言語 CafeOBJ で実現する方法に関して言及した．このアプローチは既に提案されており，Z 仕様を手動で代数仕様の記述に変換し，Z の論理体系を実行可能な処理系で実現することで，変換された記述を実行し，検証作業を自動化している．

本研究では上記のアプローチを基礎とし，Z 仕様を代数仕様の記述に変換する部分の自動化を提案した．Z 仕様を代数仕様の記述に変換するためには，シーケントと呼ばれる形式に変換する必要があり，その記述方法を提案している．

また，提案した記法に基づいて手動で検証例を示した．

6.2 今後の課題

今後は，本稿の提案に基づき Z 仕様を代数仕様の記述に変換する部分を実装を試みる．2002 年に Z が ISO 標準化されたので，それに伴い，Z の論理体系 [13] も変わりつつある．新しい Z の論理体系についてはまだ報告されていないが，事例 [19] を参考にする場合は注意を必要とする．

仕様だけが変換されても，検証すべき事柄が与えられなければ検証作業は行えないのは当然である．そこで，代数仕様記述への変換対象は Z 仕様だけでなく，検証すべき論理式も含まれるよう考慮する必要がある．

また，検証結果において，

$$\dfrac{\dfrac{A}{B}\quad\dfrac{C}{D}}{E}$$

のような証明図が得られた場合を考える．このとき，部分証明図

$$\dfrac{A}{B} \quad と \quad \dfrac{C}{D}$$

を得るための機構を考慮する必要がある．それは，(1) 結論 E から前提 B および D をどのようにして得るのか，(2) 検証作業の並列，分散化が可能かどうか，の 2

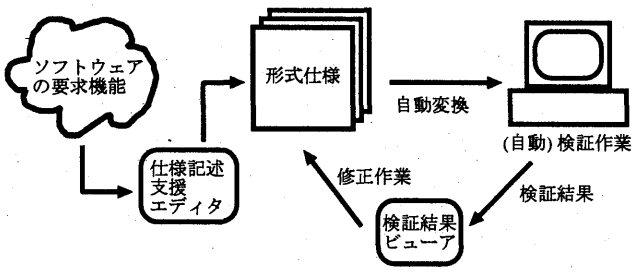

図3 検証支援環境
Fig.3 An Environment for Verification

点である．このような機構が実現できれば，検証作業の並列化，分散化が可能となり，検証作業の一層の効率化が期待できる．

現状では，検証作業の自動化として

- Z 仕様の代数仕様への変換
- 代数仕様での検証作業

の 2 点を考慮しているが，検証作業をより効率良く行うためには検証作業支援環境として提供することが望ましい．具体的には

- Z 仕様記述を支援するエディタ
- 検証結果ビューア

の 2 点が加わることで，検証作業支援環境を整備，提供することができる．

Z 仕様記述を支援するエディタとは，仕様を記述する際に使用する記号をテーブルとして備え，それを選択することで効率よく仕様を作成するツールのことをいう．このツールについては記述可能範囲を限定した試作版が存在する [7] ので，今後はより広範囲の記述が可能となるように拡張したいと考えている．

検証作業を CafeOBJ で実行した場合，その実行結果は書き換えた項をテキストとして表示するだけであり，本稿で提示したような証明図を作成するものでもない．したがって検証結果を的確に提示しているとは言い難い．より利用しやすい検証結果が提示できるよう，検証結果ビューアの設計，実装も考慮したい．

参考文献

[1] R.D. Arthan, "On Formal Specification of a Proof Tool," *Proceedings of Formal Software Development Methods(VDM'91)*, LNCS 551, pp.356–370, 1991.

[2] Doma and Nicholl, EZ: A System for Automatic Prototyping of Z Specifications. In *Proc. VDM'91: Formal Software Development Methods*, LNCS 551(1991), pp. 189–203.

[3] R. Diaconescu and K. Futatsugi, CafeOBJ *Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*, World Scientific, 1998.

[4] J. Goguen, T. Winkler, J. Meseguer, K. Futatsugi, and J.P. Jouannaud, *Introducing OBJ*, Technical Report, SRI–CSL–92–03, 1992.

[5] J. Goguen and G.Malcolm. "A Hidden Agenda," *Theoretical Computer Science*,Vol 245, No 1, pp.55–101, 2000.

[6] I. Hayes, *Specification Case Studies Second ed.*, Prentice-Hall, 1993.

[7] 石川，GUI を用いた Z 仕様検証支援環境の提案．電子情報通信学会技術研究報告 Vol.100 No.63, SS2000–2, pp.9–16, 2000.

[8] H. Ishikawa, "An Approach to Providing a Proof Tool for Z Specifications," *Proceedings of The 2003 International Technical Conference on Circuits/Systems, Computers and Communications*, volume 1, pp.345–348, 2003.

[9] 石川，形式仕様言語 Z の自動検証化の試み．情報科学技術フォーラム (FIT2003) 一般講演論文集第 1 分冊, pp.141–142, 2003.

[10] X. Jia, "An Approach to Animating Z Specifications," *Proceedings of the 19th Annual IEEE International Computer Software and Application Conference(COMPSAC'95)*, pp.108–113, 1995.

[11] X. Jia, *ZTC: A Z Type Checker, User's Guide, version 2.0*, available on Internet via anonymous ftp at `ise.cs.depaul.edu`.

[12] Kolyang, Santen. and Wolff, *A Structure Preserving Encoding of Z in Isabelle/HOL*. In Ninth international Conference on Theorem Proving in Higher Order Logics TPHOL, LNCS 1125, pp.283–298, 1996.

[13] A. Martin, "Encoding $\mathcal{W}$: A logic for Z in 2OBJ," *Proceedings of Industrial-Strength Formal Methods(FME'93)*, LNCS 670, pp.462–481, 1993.

[14] Irwin Meisels and Mark Saaltink, *The Z/EVES Reference Manual(for Version 1.5)*. Technical Report TR–97–5493–03d, ORA Canada, 1997.

[15] J. Meseguer, "Rewriting as a Semantic Framework for Concurrency: A Progress Report," *Proceedings of 7th International Conference of Concurrency Theory*, LNCS 1119, pp.331–372, 1996.

[16] A.T. Nakagawa, T. Sawada and K. Futatsugi, CafeOBJ User's Manual — ver.1.4 —, 1998.

[17] J.M. Spivey, *The Z Notation: A Reference Manual. 2nd ed.*, Prentice-Hall, 1992.

[18] West and Eaglestone, Software development: two approaches to animation of Z specifications using Prolog. In *Software Engineering Journal, Vol.7, No.4*, 1992, pp.264–276.

[19] 谷津, 二木, 代数仕様によるZ仕様の検証支援. In コンピュータソフトウェア, Vol.13, No.6, pp.26–42, 1996.