GILO/Z：オブジェクト指向仕様記述のためのZ記法の拡張

中島 震†

開発上流工程の生産性向上ならびに設計品質向上を進める上で大きな影響を与える技術として，オブジェクト指向技術と形式仕様技術が注目されている．本稿では，形式仕様言語としてZ記法を取り上げて，そのオブジェクト指向拡張記法について提案する．Z記法はモデルベースの仕様言語であり，対象システムを状態と操作の集まりとしてモデル化する．一方，オブジェクト指向概念で用いられる「オブジェクト」は状態と操作を持つ実体である．そのため，オブジェクト指向ソフトウェアに形式性を持ち込む手法としてZ記法は相性が良いと考えることができる．ところが，オブジェクト指向技術は，情報隠蔽，性質継承，多相性，オブジェクトの状態，等といった複雑な概念からなり，すべての側面を厳密に表現する体系がないのが現状である．特に，Z記法に基礎を与えるZF集合上にコード化することは困難である．提案するオブジェクト指向拡張Z記法GILO/Zは，Z記法への構文的な拡張と仕様スタイルの組合せという混合手法を採用し，表層構文から基本構文への構文的な書き換えにより性質継承や多相性を扱う方針を採用した．特に，基本構文からなるZ記法の拡張言語について表示的手法により厳密な意味を与えられることを示す．これはオリジナルのZ記法に対する表示的意味の自然な拡張になっている．

GILO/Z: An Extension of Z Notation for Object-Oriented Specification

SHIN NAKAJIMA †

Object-oriented technology and formal specification technique are expected to have significant impacts on future software development methodology. This paper proposes GILO/Z, an object-oriented extension of Z notation. Z notation is a model-based specification notation, which allows system modeling with respect to states and operations. As object in object-oriented technology is considered as an entity with states and operations, Z notation is considered suitable to be a rigorous basis for object-oriented software. However, it is difficult to encode in ZF axiomatization of set theory, which Z notation has its basis on, most of object-oriented concepts such as encapsulation, property inheritance, polymorphism, and object states. To overcome the above difficulties, GILO/Z has adapted a mixed approach which consists of kernel language as an extension of Z notation, and specification style for describing object-oriented software. In addition, GILO/Z has a set of rewriting rules from surface syntax language to basic syntax. The former provides object-oriented concepts such as inheritance and polymorphism. The semantics of the latter language is given precisely as an extension of the denotional semantics of the original Z notation.

1. はじめに

ソフトウェア開発，特に設計等の開発上流工程の生産性や設計品質の向上に大きな影響を与える技術として，オブジェクト指向技術と形式仕様技術が注目されている[8]．これら2つのアプローチは独立に発展してきたが相入れないものではない．互いを補う技術であり，両者を組み合わせることができる．すなわち，オブジェクト指向技術は形式仕様記述を得るための方法論となる．逆に，形式仕様技術は前者が提供するダイアグラムの意味を厳密に定義するための基礎となる．

本稿では，形式仕様言語としてZ記法[13]を取り上げて，オブジェクト指向仕様記述のための拡張記法を提案する．Z記法はモデルベースの仕様言語であり，対象システムを状態と操作の集まりという視点から記述する考え方を提供する．一方，オブジェクト指向概念の「オブジェクト」は状態と操作を持つ実体である．そのため，オブジェクト指向ソフトウェアの仕様記述に形式性を持ち込む手法としてZ記法の相性が良いと考えられる．さらに，Z記法の背景にあるZF集合論は比較的単純な数学であるため，多くの技術者にとって馴染みやすいという利点がある．同様な観点から，Z記法にオブジェクト指向概念を導入する研究が知られている[3),4),9),14]．しかし，オブジェクト指向ソフトウェアは，

† NEC C&C 研究所
C&C Research Laboratories, NEC Corporation

情報隠蔽，性質継承，多相性，オブジェクトの状態，等の複雑な概念からなり[10]，すべての側面を厳密に表現することが難しい．そこで，本稿では，「Z記法への簡単な拡張＋仕様の書き方の約束事」という混合手法に基づくオブジェクト指向拡張Z記法GILO/Z*を提案する．なお，文献11)にGILO/Zによる具体的な記述例を示したように，GILO/Zの言語仕様は，GILO/Zで記述した仕様がSmalltalkやC++によって実現されることを強く意識したものとした．

本稿の主な成果としては以下の2つである．(1)混合手法によりオブジェクト指向ソフトウェアの仕様を表現しうる記法を提案したこと．(2)Z記法への拡張部分について，Spiveyによる方法[12]を適用してZ記法の意味記述と整合性の良い表示的意味記述を与えることで，Z記法の考え方を何ら変更していないことを具体的に示したこと．以下，第2章でGILO/Zの概要を紹介する．第3章でオブジェクト指向概念の中で特に重要な点を取り上げ，Z記法に融合する際の問題点とGILO/Zにおける解決策について述べる．第4章でGILO/Zの意味を与えるために，Z記法の表示的意味記述を拡張する方法のポイントを具体的に示す．最後に今後の研究課題についてまとめる．

2. オブジェクト指向概念とZ記法

2.1 GILO/Zの概要

Z記法[13]を用いた仕様の記述単位はスキーマである．Z記法をソフトウェアの仕様記述に適用する場合，複数のスキーマが互いに強い関連を持つことが多い．そのため，関連するスキーマ群をまとめて管理する枠組が必要となる．その結果，モジュール概念やオブジェクト指向概念を導入することで，スキーマ群を構造化しようという研究が行われている[5),10),14)]．GILO/Zでは，他のオブジェクト指向拡張Z記法[3),4),9)]と同様に，ある状態スキーマとこれに関連する操作スキーマを一括管理する枠組としてモジュールを導入した．

GILO/Zの目標は，Z記法への簡単な言語的拡張と仕様の書き方の工夫により，オブジェクト指向ソフトウェアの仕様を表現できる記法を提供しよう，というものである．以下，簡単な例を用いて，GILO/Zにおけるモジュールの記述方法を説明する．

図1にモジュールAddressNoteの記述例を示す．先に述べたように，モジュールは互いに強く関連するスキーマを一括管理する枠組であり，スキーマ等のZ記法で許された言語要素を持つことができる．GILO/Zでは，StateとInitと名付けたスキーマは特別な役割を果たす．Stateはモジュールの内部状態を与える状態スキーマであり，InitはStateを初期化する操作スキーマである．これら2つ以外の操作スキーマをメソッドスキーマと呼ぶ．図1のモジュールAddressNoteは内部状態としてknownとaddressの2変数*1と，Add, Find Address, DummyAddという3つのメソッドスキーマを持つ．各メソッドスキーマからは，変数を参照することができ，ΔあるいはΞを付加して指定する．ここで，Δは読み書きの両方を，Ξは読みだし専用の参照を行うことを示す*2．メソッドスキーマAddは2つの入力引数name?とaddr?をとり，変数addressを更新する．ここで，addressは更新前の値をaddress'は更新後の値を表す*3．また，DummyAddは単にAddを起動するためのメソッドであるが，後に多相性の説明で用いる都合上，定義した．

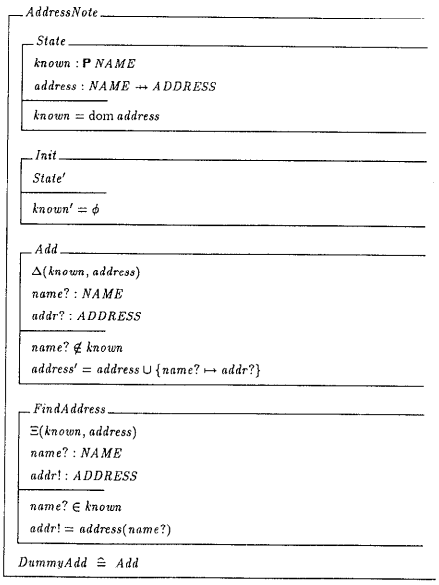

図1 モジュールAddressNoteの定義
Fig. 1 Definition of module AddressNote.

GILO/Zのモジュールは性質継承機能を持つ*4．例としてモジュールAddressNoteのサブモジュールMyAddressNoteを**図2**に示す．これは，都市名を指定して居住者を求めるメソッドを追加したアドレス帳である．モジュール名の<修飾として上位モジュール名を指定する．

性質継承機能を定義するためには，上位モジュール

* Generic Interaction Language for Objectsの略．

*1 インスタンス変数に相当する．
*2 Object-Zの記法を流用した．
*3 Z記法のコンベンションと同じである．
*4 現在，多重継承を考えていない．

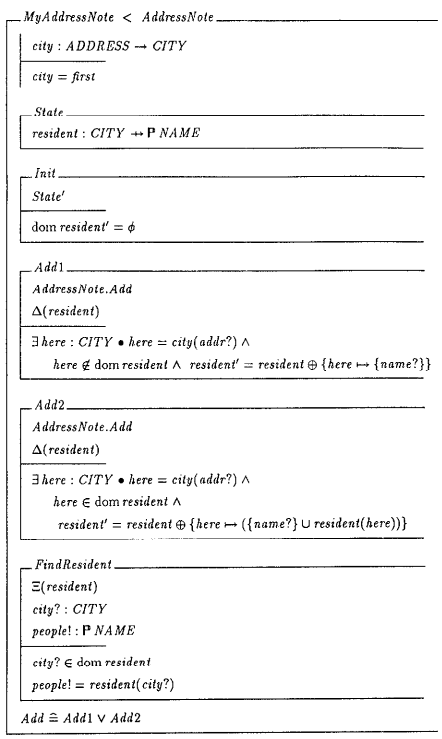

図 2 モジュール MyAddressNote の定義
Fig. 2 Definition of module MyAddressNote.

から下位モジュール定義を得るための規則を明確にしなければならない．GILO/Z で書かれた仕様は，Smalltalk や C++によって実現することを想定している．これらの言語は性質の段階的な追加機構としての継承機能を持つので，GILO/Z も同様な考え方の継承機能を提供することにした．以下，継承規則について述べる．

第1に，状態スキーマ State は上位モジュールの State と追加定義した状態スキーマの ∧ 演算により生成する．第2に，Init スキーマについても同様に ∧ 演算により新しいスキーマを生成する．第3に，メソッドスキーマについては追加定義したスキーマそのものを新たなモジュールの要素とするが，同じメソッド名を持つスキーマについては2つの指定方法を提供する．すなわち，下位モジュールが定義するメソッドで完全に置き換える方法と，下位メソッドの一部として上位メソッドを使用する方法である．後者は CLOS の call-next を用いたメソッド起動に相当し，下位クラス側でメソッドの差分を記述したい場合に有効である*．

* アドホックなメソッドの追加定義になりやすいためこのような機能は好ましくない．しかし，GILO/Z では，Smalltalk や C++との関係を優先させるために導入した．

図2の例では Add において上位の同名メソッドスキーマを AddressNote.Add という形式でモジュール名を付加して取り込んだ．

2.2 関連研究

Z記法とオブジェクト指向技術との関係から見ると，仕様の書き方を工夫することでオブジェクト指向概念を表現しようとする研究とオブジェクト指向概念を提供する言語要素を持つ仕様言語の研究とがある．

第1の流れとしては，HOOD と Z の関係について議論した研究がある[6]．HOOD は継承機能を持たない情報隠蔽性のみを中心とした段階的詳細化技法である．Hall の提案[14]は，Z記法をそのまま用い仕様の書き方を工夫することで，オブジェクト指向スタイルに基づく Z 仕様を得る方法論である．いずれも継承や多相性という重要な側面を扱っていない．

第2の流れとしては，情報隠蔽，性質継承，多相性，オブジェクトの状態，等の概念を仕様言語レベルで提供するもので，Object-Z, ZEST, Z++, 等がある[10),14]．Object-Z[3]は，オブジェクト指向拡張 Z 記法の先駆的な研究であり，インスタンス変数参照を示す $\Delta(a)$ 構文等が他に影響を与えている．しかし，クラスの意味がイベントトレースにより与えられており，ベースになっている Z 記法の部分との整合性がとれていない．ZEST[4]は，段階的な性質付与としての継承とサブタイプ関係とを明確に区別して議論している点が重要である．メソッドの役割を果たす操作スキーマ群を持つ状態スキーマとしてオブジェクトを定義しているが，メソッド起動の意味付けが明確でない．Z++[9]は，クラス継承の意味を詳細化規則として与えているため，サブタイプ関係を明確に定義できる階層関係を持つ．逆に，Smalltalk や C++といったオブジェクト指向言語との関係で重要になる段階的な性質付与としての継承関係を表現できないという問題点がある．また，文献[5]は GILO/Z と同じようにオリジナルの Z 記法に対する意味記述との整合性を重視した拡張提案であるが，オブジェクト指向概念を持つものではない．

本稿で提案する GILO/Z は，モジュールの導入と仕様の書き方の工夫の組合せという混合手法により，オブジェクト指向ソフトウェアの仕様を表現可能にするものである．モジュールを含む基本構文については Z 記法と同じ数学的な構造の上で意味を与えることができる．特に，モジュールは関連するスキーマをまとめる一種の環境のようなものであり，モジュールの構成スキーマには Spivey の Z 記法に対する表示的意味記述と同じ Variety-based Semantics の考え方で意味を与えることができる．したがって，Z 記法に対して得ら

れている種々の成果，たとえば公理系Wや詳細化規則，等を簡単な修正で適用することができると考えられる．

3. オブジェクト指向概念の取り扱い

3.1 方　　針

GILO/Zでは，数あるオブジェクト指向概念の内，特に，オブジェクト識別，メソッドの包含多相性，性質継承，を考える．SmalltalkのMVCモデル等を例にしたケーススタディ[11]を行った結果，最低限これら3つの機能がないと，オブジェクト指向フレームワークの仕様を表現できないことが判明した．以下，3つの機能を簡単に説明する．(a)オブジェクト識別：あるモジュール記述で定義された性質を持つ個々のオブジェクトを参照できること，(b)メソッドの包含多相性：あるモジュールのメソッドがその下位モジュールのオブジェクトに対しても起動できること，(c)性質継承：上位モジュールの定義をもとに段階的に性質を付与して新たなモジュール定義を得られること．

上記の機能は，オブジェクト指向ソフトウェアの仕様記述を行うための必須言語機能であるが，Z記法と整合性のとれた形式意味を持つ仕様言語を設計することは難しい．そこで，図3に示すように2段階で考えることにした．すなわち，上に述べたようなオブジェクト指向概念を持つ表層言語構文から構文的な書き換えにより基本構文を得る*．基本構文は，オリジナルのZ記法に複数スキーマをひとまとめに管理するモジュールを導入した仕様言語である．基本構文に対して，Z記法と同様な手法で表示的な意味記述を与える．以下，上記3点をZ記法に導入する際の問題点とGILO/Zにおける解決方法について説明する．

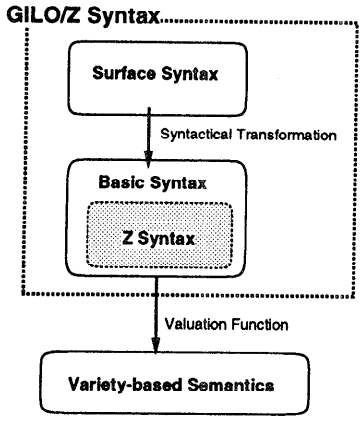

図3　意味記述の与え方
Fig. 3　Illustration of GILO/Z approach to formal semantics.

* 付録にGILO/Zの抽象構文を示す．

3.2 オブジェクトとメッセージ

オブジェクト指向ソフトウェアの仕様記述では，同じ性質を持つオブジェクト，すなわちインスタンス，を複数取り扱いたい場合があり，オブジェクトを識別しなければならない．Z記法では，既定義スキーマをタイプとしてもつ変数を宣言することができる．このスキーマタイプの機能* を用いてある状態スキーマのタイプを持つ変数を宣言すれば，指定した状態スキーマのインスタンスを値として取り扱うことが可能になる．

上のZ記法の考え方をオブジェクトに適用すると，オブジェクト指向拡張Z記法ではモジュールをタイプにみなすことに相当する．一方，オブジェクト指向言語では，Smalltalkの擬変数selfのように自己参照性がある．さらに，Z記法の拡張のような強い型付けを持つオブジェクト指向言語では，クラス定義を構成するインスタンス変数やメソッド中に自身のクラスを参照するという自己参照がいたるところに現れることが知られている[1]．ところが，Z記法では，スキーマ定義の中で，自身に相当するスキーマ・タイプを持つ変数を使うことができない**．したがって，モジュールをタイプとするような拡張を行うとZ記法と整合性の良い意味を付与することが困難となる．

そこで，GILO/Zでは，モジュール内で定義した状態スキーマStateを陽に扱うことにより，この問題に対処することにした．すなわち，仕様記述の約束事として，モジュール定義中の状態スキーマを個々のオブジェクトの実体と見なすこととする．これにより，モジュールをタイプと考える必要がなくなり，数学的な問題を回避することができる．以下，状態スキーマをオブジェクトと同一視する考え方を，メソッド起動の構文とそのインフォーマルな意味付けに関連して説明する．

Z記法では，スキーマ指定*** により，既定義スキーマで定義した仕様を別の文脈で用いることができる．この考え方を延長すると，メッセージ・センド，すなわちメソッド起動を行うためには，起動したいモジュールとメソッドスキーマを，以下のように指定することになる．

* 例えば文献12)のpp. 25-30を参照のこと．
** 再帰定義は自己参照を許しラッセルのパラドックスに相当する．そのため，Z集合に基づくZ記法では再帰的な定義を排除している．
*** Schema Designatorと呼ぶ．

$name = \text{'Nakajima'} \wedge$
$addr = (\text{'ChuoKu'}, \text{'Tsukuda'}) \wedge$
$MyAddressNote.Add[name/name?,$
$addr/address?]$

本スキーマ式を評価することで，MyAddressNoteのAddメソッドを起動して新しいエントリを追加すると考える．

しかし，個々のオブジェクトを区別してメソッド起動を行うためには，上記の構文では不十分である．指定のオブジェクトに対してメソッドスキーマを適用しなければならない．以下のような表層構文で表現することが望ましい．

$|note : AddressNote.State$
$note.Add[name/name?, addr/address?]$

問題となるのはnoteがモジュールAddressNoteのStateスキーマタイプに属する値であるため，note.Addの意味が明確でないという点である．そこで，次のような書き換えを行う．

$note.Add[name/name?, addr/address?] \Rightarrow$
$AddressNote.Add[note, name?, addr/address?]$

すなわち，状態スキーマの値* noteを実引数としてメソッドスキーマに渡すことで実現する．同時に，Addメソッドスキーマの定義を書き換えて，パラメータ付きスキーマにしておく．

Add[*S*]
…

以上，オブジェクトを指定モジュールのStateスキーマとする約束事とメソッドスキーマならびにメソッド起動式の構文的な書き換えを行うことにより，オブジェクトとメッセージをGILO/Zの基本構文で説明可能な表現を得るようにした．

3.3 メソッドの包含多相性

クラス継承機能を用いる第1の目的は仕様を段階的に定義したいからである**．上位クラスで定義済みのメソッドを下位クラスのオブジェクトに対しても実行可能としたい．これをメソッドの包含多相性と呼ぶ．

オブジェクト指向拡張Z記法において，この問題を考察するために簡単な例を示す．Upperは状態変数aの更新メソッドUpdateを持ち，LowerはUpperを継承するが，新たな状態変数bを導入する．

Upper
State
$a : \mathbf{N}$
Update
$\Delta(a)$
$x? : \mathbf{N}$
$a' = a + x?$

Lower < *Upper*
State
$b : \mathbf{N}$

下位モジュールLowerはUpdateメソッドを継承するため，Lowerに属するオブジェクト1に対してもUpdateメソッドを適用可能でなくてはならない．1にUpdateを適用した後，1を構成する変数はa'とb'であるが，Updateがbについては何も言及していないためb'の値が未定義となる．本来，b'=b, と考えなくてはならない．しかし，Updateは上位モジュールで定義されたメソッドであり，下位モジュールがどのような変数を導入するかを予測できない．そのため，包含多相性を持つようにメソッドスキーマの説明を与える方法が必要となる．

Z記法ではスキーマに包含多相性のような概念はない．したがって，包含多相性と等価な機能を持つスキーマを構文的な変換により得る方法を用いた．すなわち，Updateを以下のようなパラメータ付きスキーマに書き換える***．

Update[*S*]
ΔS
$x? : \mathbf{N}$
$a' = a + x?$
$\{a'\} \lhd \theta S' = \{a\} \lhd \theta S$

ここで，SはStateスキーマを実パラメータとしてとる．また，θSはZ記法の基本操作として定義されており，スキーマSの状態変数からなる特性タプルを得る．次に，$\{a\} \lhd \theta S$は，本稿で導入したものであるが，Z記法の「関係」に対する値域削除操作記号$\lhd$をオーバーロードした操作で，特性タプルθSから{a}を削除することを示す．

上例では，UpdateメソッドをLowerのオブジェクトに適用するとSにLowerのStateが束縛されるので，θSはLower.Stateから作られる順序付きタプル{a, b}になる．その結果，$\{a\} \lhd \theta S$は{b}となる．すなわち，制約式$\{a'\} \lhd \theta S' = \{a\} \lhd \theta S$は指定の変数a以外

---

* より正確には，AddressNoteの状態スキーマStateから作られる特性タプル（characteristic tuple）．
** プログラムレベルでは差分プログラミングと呼ぶ技法に相当する．
*** これは，3.2節で述べたメッセージセンドへの対処方法と整合性がとれている．

の変数について，操作の前後で値が不変になるような条件を与える．したがって，制約式は，b'=b, と同じことである．

3.4 性質継承と多相性

モジュール間の性質継承規則は2.2節に示した．これを実現するためには，モジュールをレコード型[2]の値としてモデル化し，レコードの連接操作により下位モジュールを作成すれば良い*．しかし，Z記法ではスキーマが定義された順番がそのスキーマの意味を与える上で重要[12]になるため，図1のDummyAddのようなメソッドの意味つけが問題となる．

DummyAddは単にAddメソッドを起動するだけの簡単なメソッドである．MyAddressNoteでは同名メソッドを持たないため，そのまま「継承」される．すなわち，MyAddressNoteもDummyAddメソッドを持つことになる．

Z記法の意味の与え方によると，DummyAddは自身が定義された時点の環境に含まれるAddメソッドを参照することになる．すなわち，MyAddressNoteにおいて参照されるのは，AddressNoteで定義されたAddメソッドである．一方，オブジェクト指向ソフトウェアの仕様記述ではMyAddressNoteがオーバーロードしたAddメソッドを起動すると考えたいことが多い**．すなわち，Z記法の方法では，上位モジュールで定義した抽象メソッドの意味を意図どおりに与えることができない．

GILO/Zではモジュール定義が与えられた時，構成スキーマ個々の意味を評価する前に，メソッドの依存関係を解析する．上の例では，MyAddressNoteモジュールの意味を与える際，スキーマの依存関係を解析することでDummyAddがAddに依存することが判明する．したがって，MyAddressNoteのAddについて意味を評価した後にDummyAddの意味を評価すれば良い．

4. 表示的意味記述

SpiveyによるZ記法への意味定義を基にして具体的に差を議論することで，GILO/Zの考え方がZ記法と整合性がとれていることを示す．

4.1 方　　針

GILO/Zの基本構文に対して表示的な意味記述を与えることを考える．モジュールの構成要素となるスキーマは基本構文上Z記法と同等であるため，Spiveyによる方法で意味を与えることができる．一方，モジュールは構成スキーマを管理する一種の環境のようなものと考えることができる．したがって，大まかに述べると，以下のように，モジュールの意味を構成スキーマの意味に還元して与えることになる．

$$\mathcal{V}[\![\textbf{module}\ W_1=S_1,\cdots,W_n=S_n\ \textbf{end}]\!]$$
$$=\{W_1\mapsto\mathcal{V}[\![S_1]\!],\cdots,W_n\mapsto\mathcal{V}[\![S_n]\!]\}$$

ここで，$\mathcal{V}$が意味関数，W_iがスキーマ名，S_iが構成スキーマを表すとした．

3章で述べたように，意味関数により評価する前に，構文的な処理を行う必要がある．すなわち，基本構文の抽象構文木を入力とし，オブジェクトに対するメッセージ式に関する書き換え（3.2節），メソッド・スキーマへのパラメータ追加と包含多相性に関する制約式の追加の書き換え（3.3節），スキーマ参照の依存関係に基づく並び換えの書き換え（3.4節），を行う．その結果として得られた基本構文の抽象構文木に対して意味関数を適用することにより表示的な意味を得る．

4.2 Z記法の意味

Z記法ではスキーマが記述の単位であると同時に意味を定義された単位である．Spiveyの方法[12]では，1階述語言語の場合と同様に，記号の解釈を与えるストラクチャを基本とする[7]．すなわち，空でない集合を領域として選び，式に現れる記号から領域へのマップを与える．式全体の意味は記号に課された制約条件を満たすマップである．ここで，Z記法では，意味記述を行うメタ言語としてZF集合論[15]を用いる．以下，**図4**を用いて，Z記法の表示的意味記述の与え方[12]について簡単に説明する*．

図4中，太線で囲んだGENVとSENVがGILO/Zのために導入した意味スキーマであり，他はZ記法の意味を与える意味スキーマである．

VARIETYがZ記法スキーマの意味に相当し，スキーマの構文的な情報を管理するsigとsigに解釈を与えるストラクチャmodelsからなる．SIGは3つの変数からなり，givenはZ記法スキーマが参照しているgiven namesを，varsはZ記法スキーマが定義している変数名を，typeは各変数とそのタイプ名の対応関係を表す．STRUCTはZ記法スキーマの解釈であり，gsetはgiven namesの，valは変数の解釈に相当する．

---

* 実際の連接操作は，後に述べるモジュール・ジェネレータについて行う．

** DummyAddを抽象メソッドと考えることに相当する．ここで，抽象メソッドとは下位モジュールで具体的なメソッドが定義されることを想定した上位モジュールのメソッドのことである．

* 文献12)に準じてZ記法をメタ言語の表現として用いる．混乱を避けるために，意味領域を構成するメタ言語の要素を意味スキーマと呼ぶことにする．また，意味を与えようとしている対象のスキーマをZ記法スキーマと呼ぶ．

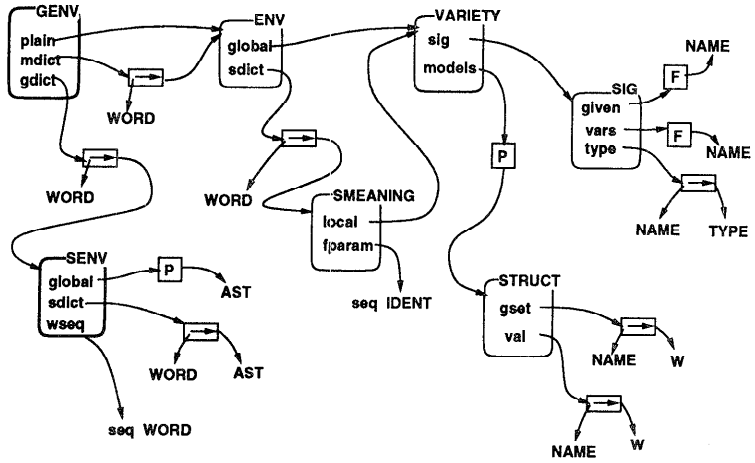

図 4 意味領域のスキーマの関連
Fig. 4 Relationship among schema's for semanic Domains.

spec	:	ENV → SPEC ↛ ENV
sexp	:	ENV → LEVEL → SEXP ↛ VARIETY
sdes	:	ENV → LEVEL → SDES ↛ VARIETY
schema	:	ENV → LEVEL → SCHEMA ↛ VARIETY
pred	:	ENV → LEVEL → PRED ↛ **P** STRUCT
term	:	ENV → LEVEL → TERM ↛ TMEANING
decl	:	ENV → LEVEL → DECL ↛ VARIETY

図 5 Z記法の意味関連
Fig. 5 Valuation functions for original Z notation.

gspec	:	GENV → GILO-SPEC ↛ GENV
mexp	:	GENV → ENV → LEVEL → MEXP ↛ ENV
mspec	:	GENV → ENV → LEVEL → MSPEC ↛ ENV
spec	:	GENV → ENV → LEVEL → SPEC ↛ ENV
sexp	:	GENV → ENV → LEVEL → SEXP ↛ VARIETY
sdes	:	GENV → ENV → LEVEL → SDES ↛ VARIETY
schema	:	GENV → ENV → LEVEL → SCHEMA ↛ VARIETY
pred	:	GENV → ENV → LEVEL → PRED ↛ **P** STRUCT
term	:	GENV → ENV → LEVEL → TERM ↛ TMEANING
decl	:	GENV → ENV → LEVEL → DECL ↛ VARIETY

図 6 GILO/Zの意味関数
Fig. 6 Valuation functions for GILO/Z basic syntax.

ここで，領域WはZF集合である．

Z記法では，Z記法スキーマに名前を付与できることから，スキーマ名と実体の関係を管理する環境ENVが意味領域として必要になる．globalは大域的に定義された無名Z記法スキーマの意味を与え，sdictは名前付きZ記法スキーマに対して名前と意味の対応関係を与える．SMEANINGは名前付きZ記法スキーマひとつに意味を与えるもので，localはZ記法スキーマ内で局所的に定義した性質の意味，fparamはパラメータ付きZ記法スキーマの場合に必要となる情報で，Z記法スキーマ参照時に実パラメータと置換する仮パラメータを持つ．

以上の意味スキーマで定義されたドメイン上に構文要素をマップする意味関数を与えることでZ記法の表示的意味を得ることができる．なお，参考のため，Z記法に対する意味関数のタイプを**図5**に示した．

4.3 GILO/Zの意味

GILO/Zの表示的意味を与えるために，各モジュールの意味を一種のENVと見なし，さらに，モジュールを管理する環境GENVを導入した．以下，GILO/ZとしてZ記法に追加した言語要素の意味関数を中心に説明する．すなわち，モジュール定義を導入する部分とモジュール内のスキーマを参照する部分について説明する．

GENVはGILO/Zの大域的な環境を与える意味スキーマで3つの変数からなる．plainはモジュール外で記述されたZ記法スキーマの環境を与える．mdictはモジュール名とモジュールの意味に相当するENVの対応関係を与える．gdictはモジュール名とモジュール・ジェネレータSENVの対応関係を与えるものである．モジュール・ジェネレータとは依存関係に従って再構成された基本構文である．ここで，ジェネレータ要素を意味関数により評価することでモジュールの意味に相当するENVを得る，という条件を，意味関数mspecを用いて明示した．

GENV

$plain : ENV$

$mdict : WORD \twoheadrightarrow ENV$

$gdict : WORD \twoheadrightarrow SENV$

$\forall w : WORD \bullet w = \mathrm{dom}\ gdict$

$\mid mspec(gdict(w)) = mdict(w)$

SENV はモジュール・ジェネレータを与える意味スキーマで純粋に構文的な情報からなる．global はモジュール内の名前付けされていない Z 記法の基本構文によるノード表現を管理する．ここで，AST により抽象構文木を代表させた．sdict はスキーマ名と基本構文によるノードの対応関係を与える．wseq は依存関係の順序付け (topological sorting) 結果を与える．制約条件として，sdict と wseq が同じ実体 (AST) を管理すること，依存関係 $<$ により与えられる半順序関係を満たすこと，の 2 点を明示した．

$$
\begin{array}{|l}
\textit{SENV} \\
\hline
\textit{global} : \mathbf{P}\ \textit{AST} \\
\textit{sdict} : \textit{WORD} \pfun \textit{AST} \\
\textit{wseq} : \textit{seq}\ \textit{WORD} \\
\hline
\mathrm{dom}\,\textit{sdict} = \mathrm{ran}\ \textit{wseq} \\
\forall a : \mathrm{ran}\ \textit{wseq} \bullet \\
\quad (\exists b : \mathrm{ran}\ \textit{wseq} \bullet (a < b)) \lor \textit{root}(a) \\
\hline
\end{array}
$$

ここで，順序関係 $<$ は 3.4 節に述べたスキーマの依存関係を示すためのもので，以下のような性質を持つ．

$$
\begin{array}{|l}
\_ < \_ : \textit{WORD}\ \textit{WORD} \\
\textit{root} : \textit{WORD} \\
\hline
\forall a, b, c : \textit{WORD} \bullet a < b \land b < c \Rightarrow a < c \\
\forall a : \textit{WORD} \bullet \neg (a < a) \\
\forall a : \quad \textit{WORD}(\textit{root}(a) \\
\qquad \Longleftrightarrow \neg (\exists x : \textit{WORD} \bullet a < x)) \\
\end{array}
$$

以上の意味スキーマで定義されたドメイン上に構文要素をマップする意味関数を与えることで GILO/Z の基本構文に対する表示的意味を得ることができる．**図 6** に示したように，オリジナルの Z 記法の対応する意味関数と異なる箇所は第 1 引数に GENV をとる点である．

メインの意味関数はトップレベルの構文要素を対象とする gspec である．**図 7** に示したようにトップレベルの環境 γ を管理する．定義本体中で γ に対する補助関数を 2 つ用いた．これらは**図 8** に示してある．mexp ならびに mspec は構文要素に対応して定義したが，前節で述べたようにスキーマ定義は基本構文に書き換えられているため，ほとんど何もしなくて良い．**図 9** の意味関数 spec がスキーマ等の仕様の要素に意味を付与するもので，GENV である γ をとる以外は Z 記法のそれとほぼ同じである．

意味関数 sdes がスキーマ参照構文 SDES の意味を与えるもので，若干の修正が必要となる．ここでは，簡単のためパラメータのないスキーマを参照する場合について説明する．**図 10** を参照されたい．

構文上 sdes は 3 つの形を持つ．第 1 に，WORD DECOR の形をしたもので，これは，Z 記法のスキーマ参照と同じである．グローバルな環境下にある定義と被参照スキーマ定義とを combine により組み合わせた結果得られる VARIETY により意味を与えるが，グローバルな環境として γ.plain.global を用いることに注意する．ここで，combine は 2 つの VARIETY の共通部分に相当する VARIETY を得る補助関数であ

$$
\begin{array}{|l}
g\_enrich : \textit{GENV} \times \textit{ENV} \pfun \textit{GENV} \\
\hline
g\_enrich = \lambda \gamma : \textit{GENV};\ \rho : \textit{ENV} \bullet \\
\quad \mu \gamma_1 \mid \gamma_1.\textit{plain} = \rho \land \gamma_1.\textit{mdict} = \gamma.\textit{mdict} \land \gamma_1.\textit{gdict} = \gamma.\textit{gdict} \\
\end{array}
$$

$$
\begin{array}{|l}
\textit{add\_top} : \textit{GENV} \times \textit{WORD} \times \textit{ENV} \pfun \textit{GENV} \\
\hline
\textit{add\_top} = \lambda \gamma : \textit{ENV};\ w : \textit{WORD};\ \rho : \textit{ENV} \mid w \notin \mathrm{dom}\ \gamma.\textit{mdict} \bullet \\
\quad \mu \gamma_1 \mid \gamma_1.\textit{plain} = \gamma.\textit{plain} \land \\
\qquad \gamma_1.\textit{mdict} = \gamma.\textit{mdict} \cup \{w \mapsto \rho\} \land \\
\qquad \gamma_1.\textit{gdict} = \gamma.\textit{gdict} \\
\end{array}
$$

図 8 補助関数
Fig. 8 Auxiliary functions.

$$
\begin{array}{|l}
\textit{gspec} : \textit{GENV} \to \textit{GSPEC} \pfun \textit{GENV} \\
\hline
\textit{gspec}\ \gamma\ [\![\textbf{let}\ w = m]\!] \cong \textit{add\_top}(\gamma,\ w,\ \textit{mexp}\ \gamma\ \textit{arid}\ 0\ [\![m]\!]) \\
\textit{gspec}\ \gamma\ [\![gs_1\ ;\ gs_2]\!] \cong \mu \gamma_1 : \textit{GENV} \mid \gamma_1 \cong \textit{gspec}\ \gamma\ [\![gs_1]\!] \bullet \textit{gspec}\ \gamma_1\ [\![gs_2]\!] \\
\textit{gspec}\ \gamma\ [\![s]\!] \cong g\_enrich(\gamma,\ \textit{spec}\ \gamma\ \gamma.\textit{plain}\ 0\ [\![s]\!]) \\
\end{array}
$$

$$
\begin{array}{|l}
\textit{mexp} : \textit{GENV} \to \textit{ENV} \to \textit{LEVEL} \to \textit{MEXP} \pfun \textit{ENV} \\
\hline
\textit{mexp}\ \gamma\ \rho\ k\ [\![\textbf{module}\ ms\ \textbf{end}]\!] \cong \textit{mspec}\ \gamma\ \rho\ k\ [\![ms]\!] \\
\end{array}
$$

$$
\begin{array}{|l}
\textit{mspec} : \textit{GENV} \to \textit{ENV} \to \textit{LEVEL} \to \textit{MSPEC} \pfun \textit{ENV} \\
\hline
\textit{mspec}\ \gamma\ \rho\ k\ [\![\ s\ ]\!] \cong \textit{spec}\ \gamma\ \rho\ k\ [\![s]\!] \\
\end{array}
$$

図 7 意味関数 gspec, mexp, mspec
Fig. 7 Valuation functions gspec, and mspec for GILO/Z.

$$
\begin{array}{l}
spec : GENV \rightarrow ENV \rightarrow LEVEL \rightarrow SPEC \nrightarrow ENV \\
\hline
spec\ \gamma\ \rho\ k\ [\![\textbf{given}\ x_1, \ldots, x_n]\!] \cong enrich(\rho,\ new\_givens(\rho.global,\ tag\ k\ (\langle x_1, \ldots, x_n \rangle))) \\
spec\ \gamma\ \rho\ k\ [\![\textbf{let}\ s\ \textbf{end}]\!] \cong enrich\ (\rho,\ schema\ \gamma\ \rho\ k\ [\![s]\!]) \\
spec\ \gamma\ \rho\ k\ [\![\textbf{let}\ w\ [x_1, \ldots, x_n] = se\ ]\!] \cong \\
\quad \mu\ \rho_1 : ENV \mid \rho_1 \cong enrich\ (\rho,\ new\_givens(\rho.global,\ tag\ k+1\ (\langle x_1, \ldots, x_n \rangle))) \bullet \\
\quad\quad \mu\ sm : SMEANING \mid sm.local \cong sexp\ \gamma\ \rho_1\ k+1\ [\![\ se\ ]\!] \wedge \\
\quad\quad\quad sm.param = \langle x_1, \ldots, x_n \rangle \bullet add\_schema\ (\rho,\ w,\ sm) \\
spec\ \gamma\ \rho\ k\ [\![\ s_1\ \textbf{in}\ s_2\ ]\!] \cong \mu\ \rho_1 : ENV \mid \rho_1 \cong spec\ \gamma\ \rho\ k\ [\![s_1]\!] \bullet spec\ \gamma\ \rho_1\ k\ [\![s_2]\!]
\end{array}
$$

図 9 意味関数 spec
Fig. 9 Valuation function spec for GILO/Z.

$$
\begin{array}{l}
sdes : GENV \rightarrow ENV \rightarrow LEVEL \rightarrow SDES \nrightarrow VARIETY \\
\hline
sdes\ \gamma\ \rho\ k\ [\![w\ q]\!] \cong \\
\quad \mu\ sm : SMEANING \mid sm \cong \gamma.plain.sdict(w) \wedge \#sm.fparam = 0 \bullet \\
\quad\quad combine\ (\gamma.plain.global,\ retag\ (q,\ k)\ sm.local) \\
sdes\ \gamma\ \rho\ k\ [\![!\ w\ q]\!] \cong \\
\quad \mu\ sm : SMEANING \mid sm \cong \rho.sdict(w) \wedge \#sm.fparam = 0 \bullet \\
\quad\quad combine\ (combine\ (\gamma.plain.global,\ \rho.global),\ retag\ (q,\ k)\ sm.local) \\
sdes\ \gamma\ \rho\ k\ [\![w_1\ q_1\ !\ w_2\ q_2]\!] \cong \\
\quad \mu\ \rho_1 : ENV \mid \rho_1 = \gamma.mdict(w_1) \bullet \\
\quad\quad \mu\ sm : SMEANING \mid sm \cong \rho_1.sdict(w_2) \wedge \#sm.fparam = 0 \bullet \\
\quad\quad\quad combine\ (combine\ (\gamma.plain.global,\ \rho_1.global),\ retag\ (concat(q_1,\ q_2),\ k)\ sm
\end{array}
$$

図 10 意味関数 sdes
Fig. 10 Valuation function sdes for GILO/Z.

る*．第2に，!WORD DECOR の形をしたもので，同一モジュールに属するスキーマ参照を示す．γ.plain.global とモジュール内のグローバル *ρ.global* と被参照スキーマとを combine する．第3に，WORD DECOR!WORD DECOR の形であり，他モジュールのスキーマ参照を示す．γ: plain: global と指定モジュール内のグローバル ρ_1.global と被参照スキーマとを combine する．

5. おわりに

Z記法にオブジェクト指向概念を導入した GILO/Z を提案し，表示的意味を与えた．他のオブジェクト指向拡張Z記法との違いは，仕様の書き方の工夫によりオリジナルのZ記法に最小限度の拡張を加えるだけで，オブジェクト指向概念をZ記法の枠組の中で自然に表現できることを示した点である．技術的には，オブジェクトの識別，メソッドスキーマの包含多相性，モジュールの性質継承，の取り扱いに工夫を要することを指摘し，ジェネレータという考え方を用いた構文的な変換による前処理により解決する方法を導入した点が新しい．一方，GILO/Z の方法では，内部状態値がたまたま同じオブジェクトは区別できなくなる，という問題点がある．Object-Z[3]のようにオブジェクト識別子を表す定数を明示的に導入する考え方もあるが，この場合，一意性を保証したオブジェクト識別子の管理，という別の問題が生じる．この種の gensym 的な問題を解決する方式が今後の課題の一つである．

本稿で報告した言語仕様と意味定義の検討と並行して，GILO/Z をオブジェクト指向設計に適用する試みも行っている．特に，オブジェクトの集団的な振舞いを明示することが重要なオブジェクト指向フレームワークの仕様記述への適用事例がある[11]．また，ソフトウェアの設計仕様で表現したい内容は多岐にわたるため，形式仕様のみですべてを表現することは現実的でない．OMT 等のダイアグラムを中心とするモデリング技法と GILO/Z を融合することで，両者の長所を生かすような方法論の検討が必要と考えている．

謝辞 本研究の機会を与えて下さった NEC C&C 研究所山本所長ならびに吉村 CAD 担当部長，およびZ記法について御教示下さった Oxford 大学 John Nicholls に感謝いたします．

* 文献 12) の p.43 を参照のこと．

参考文献

1) Canning, P., Cook, W., Will, W. and Olthoff, W.: Inheritance for Strongly-Typed Object-Oriented Programming, *Proc. OOPSLA '89*, pp. 457-467 (1989).
2) Cardelli, L.: Semantics of Multiple Inheritance, *Semantics of Data Types*, LNCS 173, Springer-Verlag (1984), also Information and Computation 76, pp. 138-164 (1988).
3) Carrington, D. et al.: Object-Z: An Object-Oriented Extension to Z, *Proc. FORTE '89*, pp. 281-296 (1990).
4) Cusack, E.: Inheritance in Object Oriented Z, *Proc. ECOOP '91*, pp. 167-179 (1991).
5) Duke, D.: Enhancing the Structure of Z Specifications, *Proc. ZUM '91*, pp. 329-351 (1991).
6) Giovanni, R. and Iachini, P.: HOOD and Z for the Development of Complex Software Systems, *Proc. VDM '90*, pp. 262-289 (1990).
7) 萩谷：ソフトウエア科学のための論理学，岩波書店 (1994).
8) 飯島，永田：実世界と形式的記述の接点としてのオブジェクト指向モデル，情報処理，Vol. 35, No. 12, pp. 412-422 (1994).
9) Lano, K. and Haughton, H.: Reasoning and Refinement in Object-Oriented Specification Languages, *Proc. ECOOP '92*, pp. 78-97 (1992).
10) Lano, K. and Haughton, H. (eds.): *Object-Oriented Specification Case Studies*, Prentice Hall (1994).
11) 中島：Zを用いたオブジェクト指向設計，*WOOC '94* (Mar. 1994).
12) Spivey, J.M.: *Understanding Z*, Cambridge University Press (1988).
13) Spivey, J.M.: *The Z Notation*, 2nd edition, Prentice Hall (1992).
14) Stepney, S., Barden, R. and Cooper, D. (eds.): *Object Orientation in Z*, Springer-Verlag (1992).
15) 竹内：現代集合論入門，日本評論社 (1971).

付録：GILO/Z の抽象構文

以下，GILO/Z の基本構文を示す．Spivey による Z 記法の抽象構文に GILO/Z のモジュール構文を追加したものである．下線により Z 記法への追加部分を明示した．

GILO-SPEC ::= **let** WORD = MEXP
| GILO-SPEC ; GILO-SPEC
| SPEC

MEXP ::= **module** MSPEC **end**

MSPEC ::= SPEC

SPEC ::= **given** IDENT, ..., IDENT
| **let** SCHEMA **end**
| **let** WORD [IDENT, ..., IDENT] = SEXP
| SPEC **in** SPEC

SEXP ::= **schema** SCHEMA **end**
| SDES
| ¬ SEXP
| SEXP ∧ SEXP
| SEXP ∨ SEXP
| SEXP ⇒ SEXP
| SEXP ↑ SEXP
| SEXP \ (IDENT, ..., IDENT)
| ∃ SCHEMA • SEXP
| ∀ SCHEMA • SEXP

SDES ::= WORD DECOR [TERM, ..., TERM]
| ! WORD DECOR [TERM, ..., TERM]
| WORD DECOR ! WORD DECOR [TERM, ..., TERM]

SCHEMA ::= DECL | PRED

DECL ::= IDENT : TERM
| SDES
| DECL ; DECL

PRED ::= TERM = TERM
| TERM ∈ TERM
| **true**
| **false**
| ¬ PRED
| PRED ∧ PRED
| PRED ∨ PRED
| PRED ⇒ PRED
| ∃ SCHEMA • PRED
| ∀ SCHEMA • PRED

TERM ::= IDENT
| φ[TERM]
| {TERM, ..., TERM}
| {SCHEMA • TERM}
| SDES
| ℙ TERM
| (TERM × ⋯ × TERM)
| θ WORD DECOR
| TERM . IDENT
| TERM (TERM)
| λ SCHEMA • TERM
| λ SCHEMA • TERM

IDENT ::= WORD DECOR

WORD — デコレーションのない識別子
DECOR — デコレーション

GILO/Zの表層構文は基本構文に以下を追加したものである．

```
GILO-SPEC ::= let WORD inherit WORD = MEXP

MSPEC ::= method WORD [IDENT, ..., IDENT] = SEXP
        | MSPEC in MSPEC
        | state SCHEMA end
        | init SCHEMA end
        | composition SCHEMA end

DECL ::= Δ ( IDENT )
       | Ξ ( IDENT )
```

本文中で参照した図1等で用いた具体的な構文と抽象構文の関係を説明するために，以下に図1のモジュールAddressNoteの抽象構文表現を与える．

```
let AddressNote =
  module
    state known : P NAME ; address : NAME ⇸ ADDRESS
      | known = dom address
    init State' | known' = φ
    method Add =
      schema Δ (known, address) ; name? : NAME ; addr? : ADDRESS
        | name? ∉ known ∧ address' = address ∪ {name? ↦ addr?}
      end
    method FindAddress =
      schema Ξ (known, address) ; name? : NAME ; addr! : ADDRESS
        | name? ∈ known ∧ addr! = address (name?)
      end
    method DummyAdd = Add
  end
```

(平成6年7月19日受付)
(平成7年2月10日採録)

中島 震（正会員）

1979年東京大学理学部物理学科卒業．1981年同大学院理学系研究科修士課程修了．同年，NEC入社．汎用コンピュータCPUの開発を経て，現在，NEC C&C研究所ソフトウェア研究部所属．オブジェクト指向技術，ソフトウェア開発支援技術，ネットワーク管理ソフトウェア，マルチメディアソフトウェア，などに興味を持つ．1988～89年米国オレゴン大学にてソフトウェア要求工学の研究に従事．1992年より東京都立大学工学部非常勤講師．情報規格調査会FDT-SWG委員．日本物理学会，日本ソフトウェア科学会各会員．