ソフトウェア工学 102-4
（1995．1．26）

形式的仕様記述言語Zの統合支援ツールの構築について

曹 建軍† 久野 茂†† 藤野 喜一†
†電気通信大学大学院情報システム学研究科
††日本総合研究所

形式的仕様記述言語Zは集合論と一階述語論理など数学的体系を基づくモデル指向仕様記述手法である。高い信頼性を持つシステムを構築する有力な仕様記述手段の一つと考えられている。仕様設計者に便利かつグラフィカルなZ言語の開発支援環境を提供し、Z言語の仕様記述手法としてのメリットと不十分な点を検討する目的で、GUIエディター、構文パーザと意味チェッカー、検証支援モデルと部品ライブラリを備える統合支援ツールを試作したので、本稿では、その概要を述べる。さらに、この試作で得られたZの実用化のための課題についても議論する。

Construction of An Integrated Support Tool for Formal Specification Language Z

Jianjun *Cao*† Shigeru *Kuno*†† Kiichi *Fujino*†
†Graduate School of Information Systems,
The University of Electro-Communications
††The Japan Research Institute, Limited

The Z notation is a model-oriented formal method,which is based on set theory and first order predicate logic,and considered to be one of powerful specification methods for constructing a high reliable system. In order to supply specification designer with a development environment of Z notation,and to examine the merit and demerit of Z notation as a specification method,we have been constructing an integrated support tool for Z notation including a GUI editer,a parser & type checker, a proving support model and a parts-libary. In this paper,we describe the outline of this system and discuss some problems for us to make Z notation practicable.

1 はじめに

形式的仕様記述言語Zは集合論と一階述語論理など数学的体系を基礎としている。従来の仕様記述手法より、Z言語は、顧客の要求を明確にすると同時に設計者のなすべき作業の基本的な面を正確かつ厳密に表現できるという利点がある。しかしながら、現状では、適切な編集手段、構文と意味の間違いを取り除くチェックツール、そして、仕様の正確性を証明する検証ツールなどが実用化されていないため、正確性の高いZ仕様が作り難く、Zの大きなメリットを十分活用することができていない。これらの問題が、Z言語の応用とZ言語自身の進歩にとって大きな障害となっている。本稿では、設計者に便利な開発環境を提供するためのZ言語の統合支援ツールの構築について述べる。

本ツールの基盤は、X windowsのグラフィカルな機能を用いて、スキーマボックスなどZ言語の特有な記号を見ながら、Zの仕様を直観的に編集できるGUIエディターである。入力したZ仕様に対して、構文パーザと意味チェッカーによって、構文上の間違いを指摘したり、仕様の異なった部分から参照されている対象がその型に関して正しい使い方をしているかどうかをチェックすることができる。この部分では、変数、スキーマなどZ仕様の構成要素間の相互参照一覧表など作ることにより、個々のスキーマがどこで定義され、参照されているかを示し、検証と部品化のベースを作り出す。また、仕様記述言語Zの大きな長所として、仕様についての検証推論能力を伴っていることがある。そのため、本ツールでは、事前条件の簡単化、仕様特性の取り出しなどの検証作業がコンピューター支援により半自動的に行うことのできる検証支援モデルも実験する。

さらに、システム仕様の再利用のために、Z言語で書かれたシステム仕様の構成要素、属性などを部品として管理し、部品ライブラリに蓄積する必要がある。ここでは、部品ライブラリの構造について議論する。

2 形式的仕様記述言語Z

Z言語では、システムはスキーマと呼ばれる構造により記述される。スキーマには、状態スキーマ、初期スキーマ、オペレーションスキーマがある。状態スキーマには、システムの抽象状態が記述され、初期スキーマには、システムの初期状態が現される。オペレーションスキーマには、システム状態に対しての操作が記述され、システムが事前状態から事後状態に変化することも表される。スキーマは次のような図形的記法により表現される。

スキーマ名

宣言部
述語部

スキーマ名はスキーマを代表する唯一な記号である。宣言部には、スキーマ成立条件とスキーマに属する変数を宣言する。述語部には、スキーマに属する各要素が満たす条件と要素間の関係を表す。スキーマに対してスキーマ修飾、離接、合接などの演算があり、これらの演算をとおして、スキーマから新しいスキーマの作成、多数のスキーマ間の関係づけることができる。これらのスキーマ特徴によって、システム仕様は階層的、厳密的に記述することができる。

Zの一例として、誕生日帳のZ仕様を例挙する。

$[NAME, DATE]$

BirthdayBook

$known : NAME$ $birthday : NAME \mapsto DATE$
$known = \mathrm{dom}\, birthday$

AddBirthday

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

. . .

ここでは、NAMEとDATEは基本型あるいは所与集合であり、システムにおける最も原始的な型である。誕生日帳という抽象的な状態は状態スキーマBirthdayBookで記述する。オペレーションスキーマAddBirthdayは誕生日帳に項目追加という操作を表すスキーマである。変数の修飾記号？、！、'はそれぞれ、入力、出力、事後状態を表すものである。また、宣言部には、スキーマの修飾記号Δはスキーマの状態が変化することを、Ξは変化しないことを表す。

3 システム概要

本節では、システム全体の構造と機能について述べる。このシステムは以下4つの機能を含む。

- エディター
- 構文パーザと型チェッカー
- 検証支援モデル
- 部品ライブラリ

まず、基盤としてのエディターを通して、Zの仕様を編集する。編集された仕様は構文パーザと型チェッカーにより、構文と型の誤りを取り出す。そして、検証支援モデルより、意味的なエラーを検出して、ユーザーに提示する。ユーザーの修正により、より正しい仕様が作成される。さらに、他の仕様で再利用できるように、完成した仕様は部品として部品ライブラリに保存する。システム構成図は図 1 で示している。

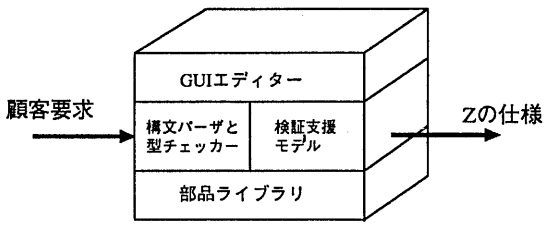

図 1: システムの構成図

3.1 GUI エディター

本ツールの中心は、X windows 上で実現したスキーマボックスなどZ言語の特有な記号を見ながら編集できるエディターであり、従来のZ編集手段より、便利で、直観的である。

3.1.1 画面の構成

図 2 で示すように、「鼎」というツールにより作られたこのエディターはエディター領域、メニューバー、パターンバー、入力ボックスとメッセージボックスの五つの部分から構成されている。各部分の機能は以下のとおりである。

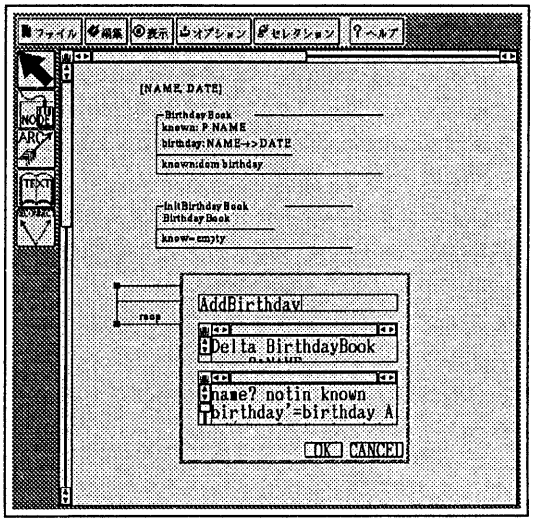

図 2: 画面のイメージ

- **エディター領域**
  紙のように編集したものを表示する場所である。表示したものがそのまま Postscript の形式で印刷できる。場合によって、Zoom in、Zoom out が可能である。

- **メニューバー**
  メニューバーにはいくつかの機能メニュー項目がついている。これらのメニュー項目を選ぶことにより、ファイル管理、編集機能、構文パーザと型チェッカーなどの機能が実行できる。

- **パターンバー**
  パターンバーには、総称ボックス、スキーマボックスと型の宣言などZ言語の記述パターンを揃えている、これらのパターンを選ぶことによりスキーマが編集することができる。

- **入力ボックス**
  各パターンの内容を入力する場所である。たとえば、スキーマボックスの場合は、スキーマの名称、宣言部と述語部が分けて入力したり、編集することができる。

- **メッセージボックス**
  操作の間違いや構文パーザと型チェッカーの結果を表示する場所である。

3.1.2 操作の手順

まず、パターンバーで入力するパターンを選んで、エディター領域に移動して、適当なところにクリックすると、一つ空きパターンが現れる。そのパターンをダブルクリックすると、入力ボックスが現れて、そこで名称、宣言、述語それぞれを入力する。そして、OKボタンを押すと、システムが入力ボックスを閉めて、スキーマの内容を合わせて適切なサイズで文字とボックスと一緒に表示させる。

修正するときは、エディター領域にあるスキーマをダブルクリックすると、入力ボックスと入力した内容が現れて、ユーザーがその上で修正することができる。

3.1.3 フォントについて

Δ、ΞなどZ言語特有な演算子はASCIIコードまたはJISコードに含まれていないため、エディターに属する演算子フォントを作るべきである。用いたツールの都合より、今回は[4]で提案されたコードを利用することになった。すなわち、Δのかわりに Delta、Ξのかわりに Xi を入力して、ΔあるいはΞの意味を表す。

3.2 構文パーザと型チェッカー

他の言語のコンパイラーと違って、本ツールでは、構文規則に従う構文分析パーザを持つ。所与集合で宣言された型が仕様の異なった部分で正しく参照されているかどうかをチェックするための型チェッカーも設置されている。

入力されたZ仕様の文字列から、変数（全域、局部）、スキーマ名、型、定数、演算子などのトークンを識別して、記号表に保存し、構文規則と型の使い方によりチェックする。構文あるいは型の誤りがあった場合は、エラーメッセージを表示する。

3.2.1 記号表

パーザの中心は記号表である。図3で示すように、記号表はヘッダ部、全域記号表、局部記号表により構成される。ヘッダ部には、記号表のヘッダ、カレントスキーマ、参照するスキーマへのポインターが保存されている。総称スキーマ、全域変数、型、スキーマが現れる順番で全域記号表に保存されている。各スキーマのノードには、スキーマ内に定義された変数を保存する局部記号表へのポインターが保存される。これにより、スキーマと局部変数を関係づけることができる。

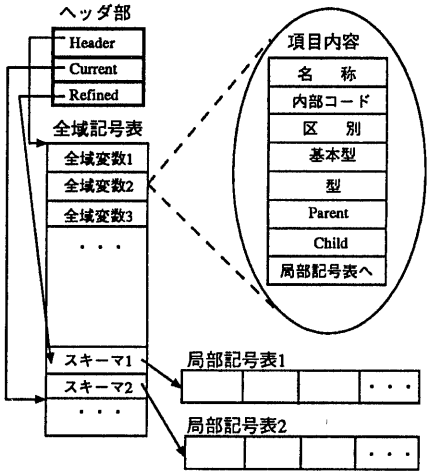

図3: 記号表の構造

3.2.2 局部変数参照に関する処理

スキーマの概念はZ言語において中心的なものであり、仕様の構成における最も重要な素材の一つである。スキーマ計算の一つであるスキーマ取り込みという機能は、あるスキーマAの宣言部で前に定義したスキーマBを取り込むことにより、スキーマAのスコープ内で、スキーマBで定義された局部変数のすべてへの参照が許される。この点で、局部変数参照に関する処理はより複雑になる。具体的なアルゴリズムは図4で示す。

3.2.3 型チェッカー

Z言語には、変数のすべては基本型あるいは基本型から構成された型を持つ[1]。型チェッカーが必要となる理由は以下のとおりである。

- 演算子により型が変わる。
  2章の誕生日帳の例で、birthdayという変数の型は基本型NAMEとDATEから構成した複合型NAME↦DATEである（NAME↦DATEというのは、定義域の型がNAMEで、値域の型がDATEである関数型である）。domは関数の定義域を求める演算子である。dom birthdayの型はbirthdayの定義域の型NAMEになる。

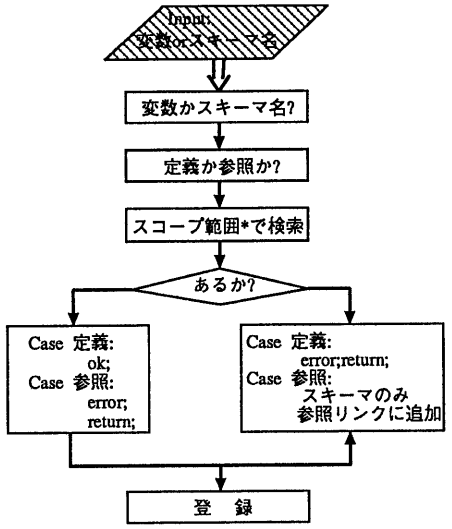

図 4: 参照処理のアルゴリズム

- **各演算子は特定な型に適用される。**
  集合、関数、列、バグなどの型がそれぞれ違う演算子を持つ。

本ツールでは、各変数の型をチェックするためには、以下の作業が行わている。

- 記号表に、各変数に対して複合的な型とそれを構成する基本型の両方とも保存する。例えば、birthday の場合は、複合型が関数で基本型は NAME と DATE である。
- 演算子のデーブルには、操作対象の型及び結果になる型に関する情報を保存する。

3.3 検証支援モデル

Z 言語は一階述語論理をベースしている、一階述語論理の推論機能を用いて、Z で書かれた仕様が意味的な誤りを含んでいるかどうかを証明できる。これにより、仕様段階で誤りを取り出すことができ、安全性の向上とコストの削減が実現できる。[1] に述べられているように、一般的に、Z における検証作業は以下のように行なわれる。

- **初期設定のチェック**
  仕様に対して、初期状態スキーマで記述された初期状態が少なくとも 1 つ存在するかどうかを検証する。
- **事前条件の簡単化**
  仕様を理解しやすいために、各オペレーションスキーマの成立条件と解釈できるの事前条件を簡単化する。
- **仕様特性の検出**
  仕様に対しては、特別に望まれる特性を例示することがあるかも知れないので、特性の検出も望ましい。
- **詳細化の正当性チェック**
  仕様の詳細化を通して、抽象的な仕様から、実行可能なプログラムに近い詳細な仕様を作成することができる。詳細化された仕様はもとの仕様と内容的に一致しているかどうか、すなわち、詳細化の正当性の検証が必要である。

いままでのところ、Z における検証は手作業で行なわれているが、本ツールでは、検証支援モデルについて一階述語論理に関する対話型の推論システムと計画している。システムの構成は図 5 で示す。

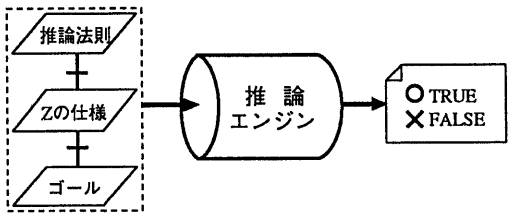

図 5: 検証支援モデル

1. 入力部分
   (a) Z 仕様：推論の前提としての Z 仕様
   (b) ゴール：検証すること
   (c) 法則：使用する推論法則。推論エンジンで使われる推論法則がすべて機械により選ぶことは難しいかもしれないので、ここで、ユーザーにより推論法則を提示する
2. 推論エンジン：一階述語論理を基づいて、前提と推論法則からゴールを検証する部分
3. 出力部分：検証の結果などのメッセージを表示する部分

以下で示すように、いくつかの問題の解決が必要であり、今後これらの問題に対して検討したい。

- 検証の対象としての仕様自身はどのような形に変換すれば、推論エンジン部の処理がより簡単に行なわれるか。つまり、中間コードの問題である。
- 推論エンジンの制御。
- 推論に使う論理法則をどの形式で推論エンジンに提供するかという問題である。

3.4 部品ライブラリと再利用

Z言語で書かれた仕様は仕様対象からモデル化されたスキーマなどの数学記号により構成される。その中には、型、状態、操作などの仕様構成要素が他の仕様記述手段を比べると、より独立的な形式で記述されている。これらの特性により、Z仕様の部品化、そして部品化により仕様レベルの再利用が可能になる。今回は、Zについて部品化の概念を提案し、Zの部品ライブラリに仕様の構成情報を保存、管理して、他の仕様でライブラリから呼び出すことにより、仕様レベルの再利用を行うことができる。

ここで、部品ライブラリの構造について述べる。ライブラリの最小構成要素はZの仕様である。本ツールを用いて設計されたZの仕様は型と総称スキーマ、状態スキーマ、オペレーションスキーマ四つの部分に分けて部品ライブラリに保存する。仕様の名称は主キーとして管理される。部品を再利用する際に、仕様の名称を呼びだすことにより、その仕様を組み込むことができる。

4 考察

本稿では、Z言語の統合管理ツールの試作について述べた。Z言語に対して、実用的な仕様記述手段とするには、より便利な編集手段、構文パーザと型チェッカー並びに検証支援機能が必要である。その上、仕様再利用のために、Zの部品ライブラリの構築により部品の再利用も考えられる。今後の課題としては、Zで書かれた仕様から実行可能なプログラム言語までの変換機能が必要である。

5 おわりに

Zは最も有力な形式的仕様言語であり、最近には、国際標準化の活動も進んでいる。ソフトウェアライフサイクルの初期段階である仕様設計段階に、形式的手法を採用することにより、システムの誤りの減少、コストの削減、そして仕様レベルの再利用などはかなり魅力的なものである。今後、Z言語の理論的な成熟と実用化を伴って、ソフトウェア工学に大きな影響を与えるであろう。

本ツールは、設計者により便利なZ言語の開発環境を提供することを目的として開発された。エディターを通して仕様を入力、構文パーザと型チェッカーをかけて構文と型エラーを取り出し、その上、検証支援機能により、意味的な誤りを検出する。仕様設計者はこのような作業を繰り返すことにより、Z言語の把握、仕様の適合性を向上することができる。

参考文献

[1] Ben Potter、田中武二訳「ソフトウェア仕様記述の先進技法、Z言語」トッパン

[2] J.M.Spivey *The Z Notation - A Reference Manaul, Second Ed.*, Prentice Hall, 1992

[3] Antoni Diller *An Introduction to Formal Methods*

[4] Xiaoping Jia *ZTC: A Type Checker for Z User's Guide (Version1.3)*

[5] 「鼎プログラミングの手引き」日本電気株式会社

[6] 久野 茂、藤野 喜一「ソフトウェア工学における形式仕様言語Zの応用について」信学技報(1994-11)、電子情報通信学会