卒業論文

代数仕様言語CafeOBJ
Algebraic Specification Language -CafeOBJ-

岡崎理恵子
Rieko Okazaki

あらまし 代数仕様言語である CafeOBJ は,. 代数仕様, つまり代数という思考体系を採用した手法を取り入れている. ここでは, 典型的な問題を CafeOBJ らしい構文を使って記述したらどのように表現されるか, ということに着目し, 解説する.

キーワード データ型, 代数仕様

1 はじめに [1,3]

1.1 テーマの説明

代数仕様言語である CafeOBJ は, ソフトウエア開発のために考案された人工言語である. 代数仕様, つまり代数という思考体系を採用した手法を取り入れている. ここでは, 典型的な問題を CafeOBJ らしい構文を使って記述したらどのように表現されるか, ということに着目し, 解説する.

1.2 背景

ソフトウエアの要求仕様をまとめて文書にするためには, 形式手法, すなわち, より良い思考体系を採用し、進めていくようなソフトウエア工学の手法が重要である.

形式手法の一流派, それが代数仕様の手法である. 代数仕様の手法とは, 代数の思考体系を採用した手法である.CafeOBJ の起源は, この代数仕様の手法にある.

1.3 目的

1 代数仕様言語の手法, ならびに CafeOBJ という代数仕様言語の理解

2 CafeOBJ アプリケーションの作成

3 CafeOBJ の実用化に向けての考察

2 準備 [4,5]

2.1 用語

- データ型

  データとそれに対応する操作を一体化したもの.

  情報システム一般の汎用モデルで, 計算機数学での基本的なモデルのひとつである.

- 抽象データ型

  代数モデルにおいては, 項の生成子 (constructor) とそれに対する操作 (observer) の関係等式で表現したもの.

- 代数仕様

  データ型を代数で表現したもの

  一般に, 台集合 (carrir set) と呼ばれるその台集合を構成する集合, その集合上に定義されたいくつかの関数, そして, その関数が生成される項が存在し, 項の間に成立するいくつかの等式. これらを与えることによって定義される.

3 $CafeOBJ$ の仕様 [1,2,3,5]

代数は汎用的な構造をもったシステムであり, 計算機科学に現われる多くの概念が代数としてモデル化できる. 代数としてデータ型を表現したものが, 代数仕様の考え方である.

3.1 自然数の仕様

自然数の仕様は、次のようなデータ型

NAT=(Nat;0,s,+,*)

を定義する.

```
 1 module NAT
 2   [NzNat Nat]
 3  signature
 4   op O:→Nat
 5   op s:Nat→NzNat
 6   op_ + _:Nat Nat→Nat
 7    { associative commutative prec:40 }
 8   op_ + _:NzNat Nat→NzNat
 9   op_ * _:Nat Nat-→Nat
10    { associative commutative prec:35}
11   op_ * _:NzNat NzNat→NzNat
12
13 axioms
14  vars N N':Nat.
15  eq N+O=N.
16  eq N+s(N')=s(N+N').
17  eq N*0=0.
18  eq N*s(N')=N+N*N'.
19
20
```

図 1 自然数の仕様

modure は予約語で, ひとまとまりの記述を表し, そのあとに、モジュール名を書く. ここでは NAT である。モジュール名のすぐ後の { から } までがモジュールの全体である. モジュールは 2 つの部分から構成されていて, それぞれは, シグニチャーと公理系である. 2 行目の $NzNat$,Nat はいずれも集合で, 包含関係を表す.$NzNat$ は O に s を 1 回以上適用して得られた要素の集合. Nat はそれに 0 を加えた集合である. よって, $NzNat$ Nat の関係が成り立つ. 代数仕様の手法では代数をソートと呼ぶ. これはシグニチャーの一部である. 3 から 12 行目のシグニチャーの部分は, 関数宣言である.

op ではじまる行は関数とランクの宣言である.:の左に関数名, 右にランクがくる.CafeOBJ では, ただ集合（ソート）名を並べる.

4 行目の o は,→Nat となり, 引数のないランクを持っている. これは引数がないのだから、o はいつも同じ値を返す. 固定された数, つまり定数である.CefeOBJ ではこのように宣言する.

6 行目の＋は足し算の左右に _ がついている. この _ は引数の位置を表している.2+3 のように,+は中置するのが普通である.CafeOBJ では, 表記法も宣言できる.

5 行目の s は後者関数と呼ばれ, 集合 N の要素に 1 を加える関数である. s は表記法の宣言がない. これは標準の関数表記法であり s(0) とかく. もし 9 行目の _ * _ が単に * だけなら、* (0, 0) のように書くという宣言になる。

シグニチャーの 7, 10 行目に , でかこまれた構文がある.

7. { associative comutative prec:40 }
10. { associative comutative prec:35 }

(A) *asociative* は, 結合則である. つまり,

$$(N + N') + N" = N + (N' + N")$$

が等しいという公理の記述と同じ意味を持つ.

(B) *comutative* は, 交換則である. つまり,

$$N + N' = N' + N$$

が等しいという公理の記述と同じ意味を持つ.

(C) *prec*40 は, 結合優先度が 40, という意味である.
数字が低いほど結合度が高い.
つまり,_+_ より _ * _ のほうが結合度が高い.

9,11 行目のように, 同じ記号, 同じ表記を使って別々の集合上に宣言された関数を
重複関数 という.
_ * _
を例にとると,

(1) 引数がともに NzNat であれば,NzNat で宣言された関数を適用する.

(2) いずれか一方の引数が NzNat でなければ,Nat 上で宣言された関数を適用する.

このように, できるだけ小さい集合を引数に選ぶ.
自然数の仕様は, 値が属する集合を区別している. 例えば

① $0 + s(0)$ では, 両引数の属する最小の集合は,Nat ¥ $NzNat$ である._ + _ の交換則より,$NzNat$ Nat 上に宣言された _ + _ を適用する. よって返されるのは,$NzNat$ の要素の値である.

② 0+0 は Nat Nat 上の _+_ の適用対象となる. よって返されるのは,Nat の要素の値である.

③ s(0)*s(0) は両引数が $NzNat$ 上に属する. よって $NzNat$ $NzNat$ 上の _*_ が適用され,$NzNat$ 上の値を返す.

④ s(0)*0 は一方の引数が $NzNat$ 上に属さない。よって Nat Nat 上の _*_ が適用され,Nat 上の値を返す.

13 から 19 行は, 公理系である.

vars は変数宣言である.
eq から. までは等式を示す. これらの式一つ一つを公理 (axiom) という. 等式の左右には signature で宣言された関数を何回か適用した結果を表す, 項と呼ばれる記号列がくる. 集合 N、定数 0、関数 s 、+、*を組み合わせて自然数と呼び, この代数を表すのに (N,O,s,+ ,*) のような組の表記を使う. 従って, モジュール NAT は, この 5 つの組の代数を定義していることになる. つまり, 自然数という抽象データ型の定義である.

3.2 静のスタック

スタックとは, 最後に格納されたデータが最初に取り出されていく先入れ後出し型のデータ構造である. このデータ構造は, 図 2.1,2.2 のように表現される.

```
1. module STACK [X::TRIV]
2.  [NeStack Stack]
3,  signature
4.   op empty:→Stack
5.   op push:Elt Stack→NeStack
6.   op pop:NeStack→Stack
7.   op top:NeStack→Elt
8.
9.  axiom
10.   var S :Stack
11.   var E :Elt
12.   eq pop(push(E,S))=S.
13.   eq top(push(E,S))=E.
14.
15.
```

図 2．1 スタックの仕様

```
1. module TRIV
2.  [Elt]
3.
```

図 2．2 トリビアルな仕様

push は Elt と Stack の要素をもらい,Nestack の要素を返す関数.
pop と top は Nestack の要素をもらい,pop は Stack,top は Elt の要素を返す関数. (Nestack は Non-empty Stack の頭文字)

任意の Stack の要素と Elt の要素 E において, 等式が成り立つ.

module 名 Stack の後の ， で囲まれた部分である. この部分をパラメータと呼ぶ.CafeOBJ のパラメータはモジュールで型付けられている. 今の例では,TRIV がモジュールで,X が仮の名前ある.STACK のなかでは TRIVE のソート, 関数を全部参照できる.

パラメータつきのモジュールとは, 複数のモジュールに共通する構造を抽出したものである.

- 整数であれば整数のスタック
- 文字列であれば文字列のスタック
- 整数と文字列が混在したスタック

いずれのせよ, スタックという容器の対する操作は, 要素が何であっても同じである. したがって, 要素に依存しない操作だけを抽出し, 要素 (の集合) をパラメータとしておき, パラメータを与えて具体的なモジュールとなる.

空のスタックの先頭要素を取り出す (top を適用する) とどうなるか, を表す方法として,

a. 何も定義せず, どうなるか全く分からない, としておく.

b. top を全関数ではなく部分関数とし, 空スタックには未定義であると明示する.

c. エラーとし特別のエラー要素を返すように定義する.

などが考えられる. CafeOBJ では, これらのいずれも使える.STACK は b. によっている.top の本来の定義域が STACK 全体であるとすると,

op top:NeStack→Elt

という宣言は,top が部分集合にしか定義されてない事を表すといえる. 形式上は全関数なのであるが, ソートの包含関係を考慮すると, 上位ソートの部分関数であるとも考えられる. 特に空スタックを表す empty は stack の要素であるが,NeStack の要素ではない. よって, 空スタックには未定義である.

3.3 動のスタック

STACK は, スタックの容器の性質を, デアル調で記述したものである.

それに対して, スタックの状態の遷移を表したものが動のスタック（図 3.1）であり, 図 3.1 は図 2.1 の矢印を表現したモジュールである. r *ule* で始まる行は *eq* で始まる行とほぼ同じ構文で,= を → に代えるだけである. この構文はルールと呼ばれ公理の一種であり, ルールの左辺が右辺に変化することを表す. この例では *pushing* という動作を行うと, スタックが $push(N, S)$ という状態に変化することなどがあらわされている.

実行におけるルールの役割は等式と同じである. たとえば,

$top(pushing(s(0), pushing(0, empty)))$ は,

$top(pushing(s(0), pushing(0, empty)))\rightarrow$
$top(pushing(s(0), push(0, empty)))\rightarrow$
$top(push(s(0), push(0, empty)))\rightarrow$
$s(0)$

のように書き換えられる. 最初の 2 回の書き換えではルールを規則として使った.

```
1. module STACKING
2.  protecting(NAT-STACK)
3.  signature
4.   op pushing:Nat Stack → NeStack
5.   op poping:Stack → Stack
6.
7.  axiom
8.   var N :Nat
9.   var S :Stack
10.  var S' :NeStack
11.  rule pushing(N,S) → push(N,S).
12.  rule popping(S') → pop(S').
13.  rule popping(empty) → empty.
14.
15.
```

図 3.1 スタックの状態変化の仕様

3.4 待ち行列

先着優先で処理する待ち行列を, 図 4.1 に定義した. 空でない行列からしか先頭を得ることができないなど, 構造はスタックとほぼ一緒である. 違うのは *top* と *pop* の定義に 2 つの等式を使っている点である.*push* が何回も適用された後でこれらの関数を適用すると, 一番最初の適用まで下って調べる必要があり, 一番最初にたどり着いたかどうかを区別するためである. また,*push* の他にもうひとつの関数を宣言し, あらかじめ先頭を得やすい作っておく方法（図 4.2）もある.

```
1. module QUEUE[X::TRIV]
2.  [NeQueue¡Queue]
3.  signature
4.   op empty: → Queue
5.   op push:Elt Queue → NeQueue
6.   op pop:NeQueue → Queue
7.   op top:NeQueue → Elt
8.
9.  axiom
10.  var E :Elt
11.  var Q :NeQueue
12.  eq pop(push(E,empty))=empty.
13.  eq pop(push(E,Q))=push(E,pop(Q)).
14.  eq top(push(E,empty))=E.
15.  eq top(push(E,Q))=top(Q).
16.
17.
```

図 4.1 待ち行列の仕様その 1

```
1. module QUEUE-2[X::TRIV]
2.  [NeQueue Queue]
3.  signature
4.   op empty: → Queue
5.   op q:Elt Queue → NeQueue
6.   op push:Elt Queue → NeQueue
7.   op pop:NeQueue → Queue
8.   op top:NeQueue → Elt
9.
10.  axiom
11.  vars E E' :Elt
12.  var Q :Queue
13.  eq push(E,empty)=q(E,empty).
14.  eq push(E,q(E',Q))=q(E',push(E,Q)).
15.  eq pop(q(E,Q))=Q.
16.  eq top(q(E,Q))=E.
17.
18.
```

図 4.2 待ち行列の仕様その 2

3.5 2 つの待ち行列

ここでは *QUEUE* を基に, 同時に 2 つの待ち行列を持つモジュールを示してみると, 図 5.1 のようになる.*QUEUE QUEUE* は自然数, 整数の待ち行列を持つモジュールである. それぞれは別のモジュールとしたいが, 両方とも *QUEUE* を使うと同じ名前になってしまう. それを避けるため, 名前替えの構文を使った.

(1) STACK の時は, モジュールを束縛するために, ビューを定義した. そしてその名前をパラメータの仮の名前に束縛した. ここでは, ビューを直接パラメータの名前に束縛する記述を使っている. この記述では,to のあとに束縛するモジュールの名前を書き, そのあと {,} の中にソートと関数の対応を記述する.

(2) *のあとの, の部分は名前替えである.Sort で始まる行がソート,op で始まる行が関数の名前替えを示す. ビューの構文のように-¿の前後に替えたい名前, 新しい名前を書く.

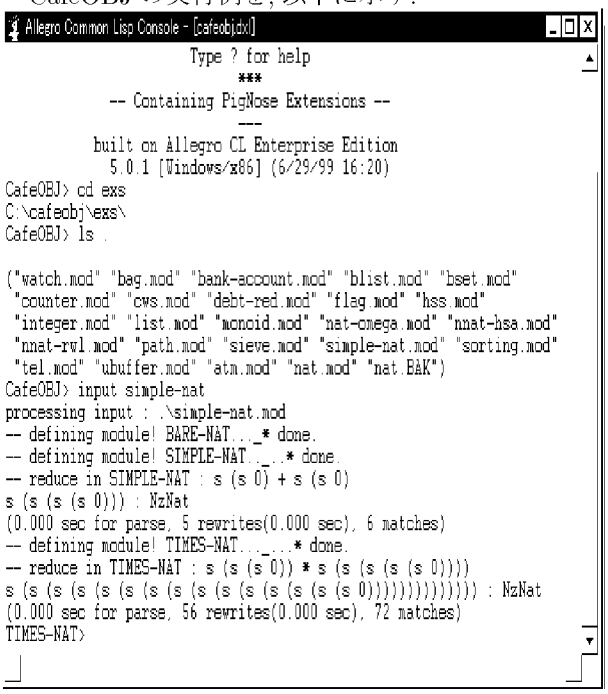
```
1. module QUEUE QUEUE
2.  protecting (QUEUE[X← view to NATsort Elt→Nat]
3.   *sort Queue →NatQueue,
4.   sort NeQueue→NatNeQueue,
5.   op empty →natEmpty,
6.   op push →natPush,
7.   op pop →natPop,
8.   op top →natTop,)
```

```
9.  protecting (QUEUE[X← view to INTsort Elt→Int]
10.   *sort Queue →IntQueue,
11.    sort NeQueue→IntNeQueue,
12.    op empty →intempty,
13.    op push →intPush,
14.    op pop →intPop,
15.    op top →intTop,
16.
```

図 5.1 待ち行列の仕様その 3

4 実行 [6]

CafeOBJ の実行例を, 以下に示す.



5 おわりに

代数仕様言語である CafeOBJ などの OBJ の機能をほぼ継承した処理系の一番の特徴は, 順序ソート (Order Sort) な TRS(書き換え系を基礎とした操作的意味論) であることである. 順序ソートとは, 台集合を構成する複数の集合間に包含関係が存在するものである.Unisort や Mainsort な TRS はかなり前から研究が進められたが, 処理系として Order Sort なものを採用したのはかなり稀である. これにより数学的な扱いをする上で (計算体系は数学的に強力であるため) かなり難しい問題があるが, 実現される世界はますます広がりを持つだろう.

参考文献

[1] 中川中, 谷津弘一, 本間毅寛, CafeOBJ への誘い http://www.ipa.go.jp/STC/ CafeP/invit-cafe-jp.html

[2] CafeOBJ Report:The Language,Proof Techniques,and Methodlogies,for Object-oriented Algebraic Specification Morld Scientific,AMAST Series in Computing 6,1998.(with razvan Diaconescu)

[3] 丸山恵, 代数仕様言語 CafeOBJ, 日本大学文理学部応用数学科 卒業研究報告書（1999）

[4] 坂田江衣子, オブジェクト指向の代数仕様による形式化, 日本大学文理学部応用数学科 卒業研究報告書 (1997)

[5] 土井亜紀子, 代数仕様記述, 日本大学文理学部応用数学科 卒業研究報告書 (1998)

[6] Ataru T.Nakagawa Toshimi Sawada Kokichi Futatsugi,CafeOBJ User's Manual -ver.1.4- p22, pp.51-86,