代数仕様言語
CafeOBJ

平成 12年 12月 9日

日本大学文理学部応用数学科
岡崎理恵子

1.1 テーマの説明

代数仕様言語CafeOBJ

- ソフトウエア開発のために考案された言語
- 代数仕様の手法を採用

1. 2　背景

1. 3　目的

１ 代数仕様の手法、並びに、代数仕様言語CafeOBJ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

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．

抽象データ型
項の生成子と，項に対する操作の関係を等式で
表現したもの．

代数仕様
データ型を代数として表現したもの．

```
1  module NAT{
2     [NzNat< Nat]
3   signature{
4    op 0:->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+0=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２．２　自然数の仕様

```
 7      {associative commutative prec:40}
10    {associative commutative prec:35}
```

Ⓐ associative :結合側

(N+N')+N''=N+(N'+N'')

Ⓑcommutative :交換側

N+N'=N'+N

Ⓒprec :結合優先度

重複関数

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

例_*_の場合

（1）引数がともに，NzNatであれば，NzNatで宣言された関数を適用.

（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適用する.

- できるだけ小さい集合を引数に選ぶ

NAT=(N,0,s,+ ,*)

NATは自然数という抽象データ型を定義

スタック

最後に格納されたデータが最初に取り出されていくという,先入れ後出し型のデータ構造

```
 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. axioms {
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

パラメータつきのモジュール

複数のモジュールに共通する構造を抽出したもの

- 整数→整数のスタック
- 文字列→文字列のスタック
- 整数と文字列→整数と文字列スタック

topの適用

空のスタック先頭要素をとりだしたときの出力方法

a．何も定義せず,どうなるかまったく分からない,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定義する.

静のスタック

スタックの容器の性質を,デアル調で記述したもの.

動のスタック

スタックの状態の遷移を表したもの.

```
 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

実行におけるルールの役割

例)

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

top(pushing(s(0),pushing(0,empty)))->
top(pushing(s(0),push(0,empty)))->
top(push(s(0),push(0,empty)))->
s(0)

待ち行列

最初に格納されたデータが最初に取り出されていくという,先入れ先出し型のデータ構造

```
 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

```
 1.module QUEUE&QUEUE{
 2. protecting(QUEUE[X<-view to NAT{sort 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 INT{sort 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

輸入構文
Protecting:他のモジュールを輸入する構文

```
2.protecting(QUEUE[X<-view to NAT{sort Elt->Nat}]
9.protecting(QUEUE[X<-view to INT{sort Elt->Int}]
```

- toのあとに束縛するモジュールの名前.
- {,} の中にソートと関数の対応を記述.

名前替えの構文

1つの仕様に,2つ以上のモジュールを定義する際,名前を付けて区別するための構文

＊{替えたい名前->新しい名前}

おわりに

- CafeOBJの処理系の一番の特徴
  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

順序ソート: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

今後，数学的な扱いをする上でかなり難しい問題があるが,実現される世界は広がることが期待される.

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

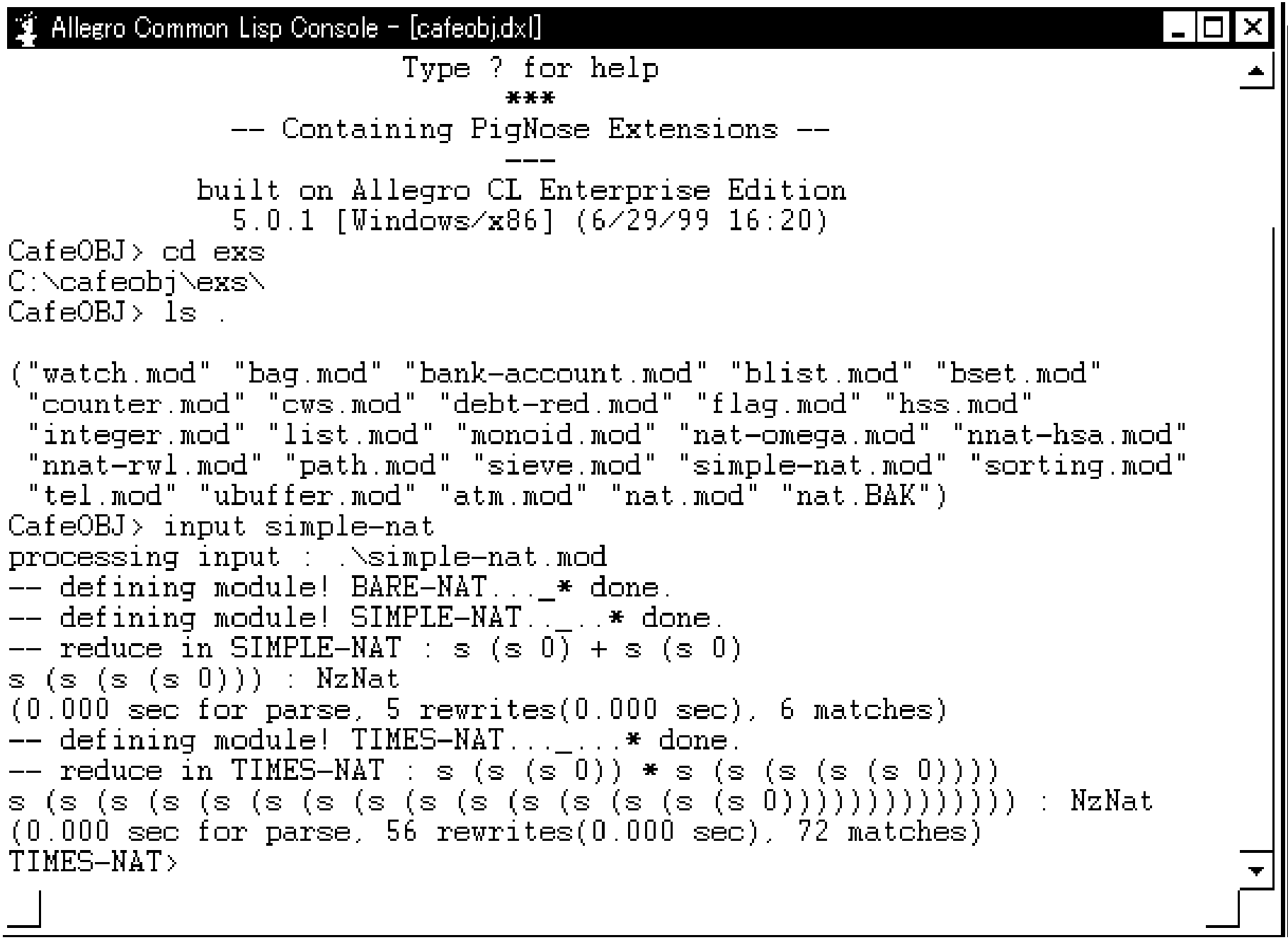
[2] CafeOBJ Report:The Language,Proof Techniques,and Methodlogies,for Object-oriented Algebraic Specification, Morld Scientific,AMAST¥ Series¥ in¥ Computing 6,1998. (with razvan Diaconescu)

[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 (999)

[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,


Allegro Common Lisp Console - [cafeobj.dxl]
Type ? for help
***
-- Containing PigNose Extensions --
---
built on Allegro CL Enterprise Edition
5.0.1 [Windows/x86] (6/29/99 16:20)
CafeOBJ> cd exs
C:\cafeobj\exs\
CafeOBJ> ls .
("watch.mod" "bag.mod" "bank-account.mod" "blist.mod" "bset.mod"
"counter.mod" "cws.mod" "debt-red.mod" "flag.mod" "hss.mod"
"integer.mod" "list.mod" "monoid.mod" "nat-omega.mod" "nnat-hsa.mod"
"nnat-rwl.mod" "path.mod" "sieve.mod" "simple-nat.mod" "sorting.mod"
"tel.mod" "ubuffer.mod" "atm.mod" "nat.mod" "nat.BAK")
CafeOBJ> input simple-nat
processing input : .\simple-nat.mod
-- defining module! BARE-NAT...* done.
-- defining module! SIMPLE-NAT...* done.
-- reduce in SIMPLE-NAT : s (s 0) + s (s 0)
s (s (s (s 0))) : NzNat
(0.000 sec for parse, 5 rewrites(0.000 sec), 6 matches)
-- defining module! TIMES-NAT...* done.
-- reduce in TIMES-NAT : s (s (s 0)) * s (s (s (s 0)))
s (s (s (s (s (s (s (s (s (s (s (s (s (s (s (s (s (s (s (s (s (s (s (s 0)))))))))))))))))))))) : NzNat
(0.000 sec for parse, 56 rewrites(0.000 sec), 72 matches)
TIMES-NAT>