CafeObj valoda

Pēteris Lediņš

2006. g. 3. janvārī

1. OBJ valodu saime [1]

OBJ algebrisko specifikāciju valodu saimē ietilpst tādas valodas kā OBJ2, OBJ3, CafeOBJ, BOBJ. Tās bāzētas uz sakārtoto vienādojumu loģiku [1], bet dažkārt papildinātas ar citām loģikām, piemēram pirmās kārtas loģiku.

OBJ valodu programmas ir teikumu kopas loģiskā sistēmā, to semantiku apraksta pamatā esošā loģika.

OBJ valodas pārsvarā rakstītas LISP valodā, pārsvarā *nix sistēmām. No OBJ valodām Microsoft Windows videi tiek piedāvāta CafeOBJ valoda, kuras interpretators pieejams `http://www.ldl.jaist.ac.jp/cafeobj/index.html`. Izskaidrojums - daudzviet OBJ valodas tiek lietotas šaurā lokā, bet CafeOBJ veidota ar lielu finansējumu no Japānas valdības puses - [3]. Savā vēsturiskā izcelsmē tā ir Maude's sekotājs [2].

2. CafeOBJ lejupielāde un instalācija

CafeOBJ pieejams bez maksas, tas tiek aktīvi uzturēts - pašlaik pēdējā versija ir no 17. novembra, 2005. gadā. Windows instalācija ir vienkārša - atliek atarhivēt arhīvu un palaist setup.exe programmu.

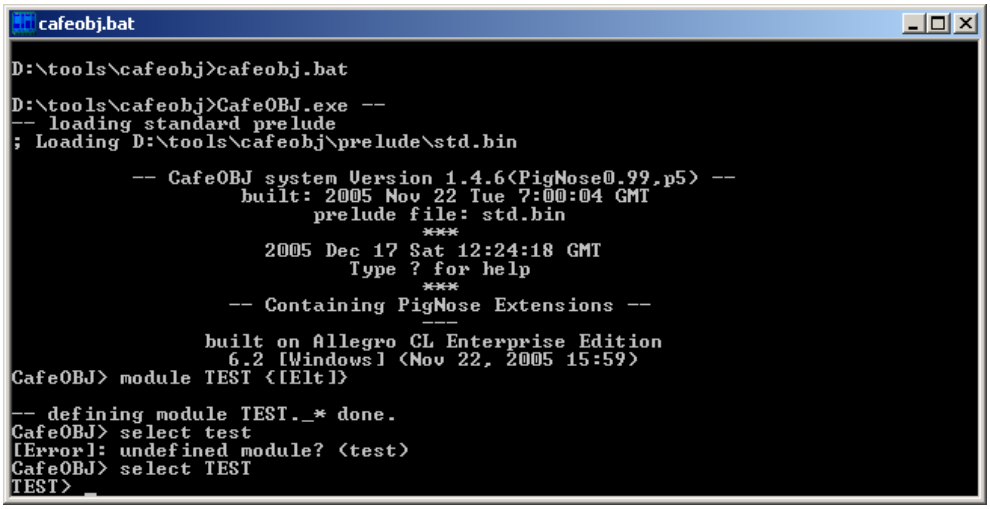
3. CafeOBJ hello world

Pamatdarbības ar CafeOBJ aprakstītas [4]. Operēšana ar CafeOBJ notiek caur teksta modes ievadu - skat Zīmējumu 1.. Šajā zīmējumā parādīta vienkāršākā moduļa izveide un pieprasījums ar viņu darboties. ("Hello world").

Ja rodas vēlme ielasīt datni, tad to dara ar komandu *input datne*. Pa direktoriju katalogu pārvietošanās notiek par *cd, pwd, ls* komandām, līdzīgi kā DOS'ā vai *nix'ā. *dir* komanda nomaina tekošo direktoriju, *ls* parāda tekošās direktorijas saturu, bet *pwd* nosauc direktoriju.

Datnes saturs tiek ielasīts līdz "eof" simbolam vai datnes beigām. No CafeOBJ var iziet ar komandu *quit* vai *eof* - t.i. ievada beigām.

[1] Order sorted equational logic, angliski.



1. zīm.: Pats sākums.

4. Moduļi [4]

[2] Moduļi ir CafeOBJ pamata elementi. Neparametrizēta moduļa definīcija:

```
module module_nosaukums
{
  module_element *
}
```

T.i. - šajā modulī ir neviens vai vairāki moduļa elementi, kas var būt importa deklarācija, sorta (tipa) deklarācija, operatora deklarācija, ieraksta deklarācija, mainīgā deklarācija, vienādojuma deklarācija vai pārejas deklarācija. Uz vecākiem moduļiem atsauce notiek tikai tad, ja tie ir definēti iepriekš, atsaukties uz jaunākiem moduļiem nevar.

4.1. Moduļu aizsardzība

Svarīgus moduļus aizsargā - *protect modulename* - šādus moduļus nevar pārdefinēt. Tādi pēc noklusēšanas ir daži pamatmoduļi, bet to var darīt arī ar paša aprakstītiem. Pretējā komanda: *unprotect modulename*. Citādi katram modulim spēkā ir tā tekstuāli pēdējā definīcija.

4.2. Moduļu strukturizācija

Moduļus iespējams (bet nav obligāti) iekšēji strukturizēt importos, parakstos (signatures), aksiomās. Tad

[2] Šis avots ([4]) ir galvenais informācijas avots referātā, ja nav dota atsauce; tad informācija nāk no šī avota vai autora izmēģinājumiem.

```
module SIMPLE-NAT {
  signature {
    [ Zero NzNat < Nat ]
    op 0 : -> Zero
    op s : Nat -> NzNat
    op _+_ : Nat Nat -> Nat
  }
  axioms {
    vars N N' : Nat
    eq 0 + N = N .
    eq s(N) + N' = s(N + N') .
  }
}
```

ir tas pats, kas

```
module SIMPLE-NAT {
  [ Zero NzNat < Nat ]
  op 0 : -> Zero
  op s : Nat -> NzNat
  op _+_ : Nat Nat -> Nat
  vars N N' : Nat
  eq 0 + N = N .
  eq s(N) + N' = s(N + N') .
}
```

Iekšējos blokus var sadalīt vairākos - piemēram aprakstīto signature bloku sadalīt trijos "paralēlos".

Pēc šāda moduļa ievadīšanas var prasīt, lai parāda:

```
CafeOBJ> show SIMPLE-NAT
module SIMPLE-NAT
{
  imports {
    protecting (BOOL)
  }
  signature {
    [ Nat, Zero NzNat < Nat,
      NzNat, NzNat < Nat,
      Zero, Zero < Nat ]
    op 0 : -> Zero
    op s : Nat -> NzNat
    op _ + _ : Nat Nat -> Nat { strat: (1 0 2) }
  }
  axioms {
```

```
    var N : Nat
    var N' : Nat
    eq 0 + N = N .
    eq s(N) + N' = s(N + N') .
  }
}
CafeOBJ>
```

CafeOBJ parāda šo moduli strukturizētā formā.

4.3. Importēšana

Sintakse:

```
{ protecting | extending | using } module_expression
```

Piemērs:

```
module BARE-NAT {
  [ Nat ]
  op 0 : -> Nat
  op s : Nat -> Nat
}
```

un

```
module BARE-NAT-AGAIN {
  protecting (BARE-NAT)
}
```

ir tas pats, kas

```
module BARE-NAT-AGAIN {
  [ Nat ]
  op 0 : -> Nat
  op s : Nat -> Nat
}
```

Daži no moduļiem kā, piemēram, BOOL tiek importēti noklusēti. To var noliegt ar komandu *set include BOOL off*.

5. Paraksts [4]

Paraksts (signature) pasaka, kas ir labi veidots terms.
Sortu deklarācijas sintakse:

```
1:  [ Nat ]
2:  [ Nat Int Rat ]
3:  [ Nat < Int < Rat]
4:  [ Nat Int < Rat ]
    [ Nat < Int ]
```

Pirmais saka, ka tiks lietot naturālo skaitļu sorts, otrais - naturālie, veselie un racionālie. Trešais sorts saka to, ka Nat ir apakškopa Int un Int - Rat. 4. - Nat un Int ir apakškopa Rat, bet papildus tam Nat ir apakškopa Int.

Jānorāda, kādi sorti tiks lietoti un tikai tad, kādas ir to attiecības, ja netiek norādīts, kādi sorti tiek lietoti tad apakšsortu definēšanas laikā noklusēti definē arī pāreju.

5.1. Operatori

Operatorus definēt kā

```
op standard_operator_symbol : list_of_sort_names -> sort_name
```

Operatora simbols ir jebkura simbolu virkne, arī tāda, kurā iekļautas atstarpes.

Piemēram:

```
op _+_ : Nat Nat -> Nat
op if_then_else_fi : Bool Nat Nat -> Nat
```

Attiecīgi sekojoši drīkst rakstīt:

```
if N < M then N else M fi
N + M
s(0) + M
```

Iespējams pārlādēt operatorus:

```
op _+_ : Nat Nat -> Nat
op _+_ : Nat Set -> Set
```

5.2. Predikāti

Predikāti ir operācijas, kas atgriež Bool vērtības.

Abas definīcijas ir ekvivalentas:

```
pred _<_ : Nat Nat
op _<_ : Nat Nat -> Bool
```

6. Ieraksti

Ierakstus un to laukus (slots) definēt šādi:

```
record:
  record record_name
  {
    list_of_slot_declarations
  }
slot_declaration:
  slot_name : sort_name
```

Piemēram,

```
  record Date {
    year : Nat
    month : Nat
    day : Nat
  }
```

Ieraksts arī ir sorts. Automātiski tiek izveidots operators, kas izveido elementu no šī tipa ir:

```
  Date { year = 123, month = 12, day = 58 }
```

Dažus no parametriem drīkst izlaist.

Automātiski tiek izveidotas divas funkcijas katram slotam:

```
  op year : Date -> Nat
  op set-year : Date Nat -> Date
```

Iespējama arī ierakstu mantošana.

7. Aksiomas

7.1. Mainīgie

Mainīgos definēt šādi:

```
  var N : Nat
  vars A, B, C : Nat
```

7.2. Vienādojumi

7.2.1. Beznosacījuma

```
  eq [ nosaukums ] term = term .
```

Piemēram (parasta deklarācija un uz-vietas-deklarācija):

```
  var N : Nat
  eq [ r-id ] : N + 0 = N .

  eq [ r-id ] : N:Nat + 0 = N .
```

7.2.2. Nosacījuma

```
  ceq [ nosaukums ] term = term if boolean_term .
```

, kur term - parasti termi, bet boolean term - tāds, kas atgriež Būla vērtību.

7.3. Pāreju deklarācijas

Vienādībām spēkā (pēc definīcijas) refleksivitāte, simetrija un tranzitivitāte. Kādai no īpašībām neesot spēkā, rodas interesantas sekas. Ja atliek refleksivitāte un transitivitāte, tā ir pārejas (transition) attiecība.

```
  trans [ nosaukums ] term => term .
  ctrans [ nosaukums ] term => term if boolean_term .
```

(nedeterminēts) Piemērs:

```
module CHOICE {
  [ State ]
  ops a b : -> State
  op _|_ : State State -> State
  vars X Y : State
  trans X | Y => X .
  trans X | Y => Y .
}
```

8. Inspicēšana

Ievieto SIMPLE-NAT aprakstu failā.

```
module SIMPLE-NAT {
  signature {
    [ Zero NzNat < Nat ]
    op 0 : -> Zero
    op s : Nat -> NzNat
    op _+_ : Nat Nat -> Nat
  }
  axioms {
    vars N N' : Nat
    eq 0 + N = N .
```

```
    eq s(N) + N' = s(N + N') .
  }
}
```

Sekojošā komunikācija ar CafeOBJ:

```
CafeOBJ> input simple-nat
processing input : D:\tools\cafeobj\simple-nat
-- defining module SIMPLE-NAT.....*..* done.
CafeOBJ> show SIMPLE-NAT
module SIMPLE-NAT
{
  imports {
    protecting (BOOL)
  }
  signature {
    [ Nat, Zero NzNat < Nat,
      NzNat, NzNat < Nat,
      Zero, Zero < Nat ]
    op 0 : -> Zero
    op s : Nat -> NzNat
    op _ + _ : Nat Nat -> Nat { strat: (1 0 2) }
  }
  axioms {
    var N : Nat
    var N' : Nat
    eq 0 + N = N .
    eq s(N) + N' = s(N + N') .
  }
}
CafeOBJ> show sorts SIMPLE-NAT
* visible sorts :
  Zero, Zero < Nat
  NzNat, NzNat < Nat
  Nat, Zero NzNat < Nat
CafeOBJ> show ops SIMPLE-NAT
................................(0)................................
  * rank: -> Zero
................................(s)................................
  * rank: Nat -> NzNat
..............................(_ + _)..............................
  * rank: Nat Nat -> Nat
  - attributes:  { strat: (1 0 2) }
    - axioms:
```

```
      eq 0 + N = N
      eq s(N) + N' = s(N + N')
.................................(Nat)................................
  * rank: -> SortId
  - attributes:  { constr }
.................................(NzNat)..............................
  * rank: -> SortId
  - attributes:  { constr }
.................................(Zero)...............................
  * rank: -> SortId
  - attributes:  { constr }
CafeOBJ>
```

Papildus iespējami vaicājumi aprakstīti [4].

9. Termu skaitļošana

Termu izvērtēšanai CafeOBJ valodā pamatā ir TRS (Term rewriting system - tātad termu pārrakstīšanas sistēma). Katrs vienādojums dod iespēju pārrakstīt termu.

9.1. Dialogi

Kārtējā sarunāšanās ar CafeOBJ:

```
CafeOBJ> input simple-nat
processing input : D:\tools\cafeobj\simple-nat
-- defining module SIMPLE-NAT.....*..* done.
CafeOBJ> show SIMPLE-NAT
module SIMPLE-NAT
{
  imports {
    protecting (BOOL)
  }
  signature {
    [ Nat, Zero NzNat < Nat,
      NzNat, NzNat < Nat,
      Zero, Zero < Nat ]
    op 0 : -> Zero
    op s : Nat -> NzNat
    op _ + _ : Nat Nat -> Nat { strat: (1 0 2) }
  }
  axioms {
    var N : Nat
```

```
    var N' : Nat
    eq 0 + N = N .
    eq s(N) + N' = s(N + N') .
  }
}
CafeOBJ> show sorts SIMPLE-NAT
* visible sorts :
  Zero, Zero < Nat
  NzNat, NzNat < Nat
  Nat, Zero NzNat < Nat
CafeOBJ> show ops SIMPLE-NAT
.................................(0).................................
  * rank: -> Zero
.................................(s).................................
  * rank: Nat -> NzNat
..............................(_ + _)..............................
  * rank: Nat Nat -> Nat
  - attributes:  { strat: (1 0 2) }
    - axioms:
      eq 0 + N = N
      eq s(N) + N' = s(N + N')
................................(Nat)................................
  * rank: -> SortId
  - attributes:  { constr }
..............................(NzNat)..............................
  * rank: -> SortId
  - attributes:  { constr }
...............................(Zero)...............................
  * rank: -> SortId
  - attributes:  { constr }
CafeOBJ> select SIMPLE-NAT
SIMPLE-NAT> reduce 0 .
-- reduce in SIMPLE-NAT : 0
0 : Zero
(0.000 sec for parse, 0 rewrites(0.000 sec), 0 matches)
SIMPLE-NAT> reduce 0 + s(0) .
-- reduce in SIMPLE-NAT : 0 + s(0)
s(0) : NzNat
(0.000 sec for parse, 1 rewrites(0.000 sec), 1 matches)
SIMPLE-NAT> reduce s( s(0) + s(0) ) .
-- reduce in SIMPLE-NAT : s(s(0) + s(0))
s(s(s(0))) : NzNat
(0.000 sec for parse, 2 rewrites(0.000 sec), 3 matches)
SIMPLE-NAT> show all rules
```

```
 -- rewrite rules in module : SIMPLE-NAT
  1 : eq 0 + N = N
  2 : eq s(N) + N' = s(N + N')
  3 : eq false and A:Bool = false
  4 : eq true and A:Bool = A
  5 : eq A:Bool and A = A
  6 : eq A:Bool xor A = false
  7 : eq false xor A:Bool = A
  8 : eq A:Bool and (B:Bool xor C:Bool)
 = A and B xor A and C
  9 : eq A:Bool or A = A
 10 : eq false or A:Bool = A
 11 : eq true or A:Bool = true
 12 : eq A:Bool or B:Bool = A and B xor A xor B
 13 : eq not A:Bool = A xor true
 14 : eq A:Bool implies B:Bool = A and B xor A xor true
 15 : eq A:Bool iff B:Bool = A xor B xor true
 16 : eq A:Bool and-also false = false
 17 : eq false and-also A:Bool = false
 18 : eq A:Bool and-also true = A
 19 : eq true and-also A:Bool = A
 20 : eq A:Bool and-also A = A
 21 : eq false or-else A:Bool = A
 22 : eq A:Bool or-else false = A
 23 : eq true or-else A:Bool = true
 24 : eq A:Bool or-else true = true
 25 : eq CXU :is Id:SortId = #!! (coerce-to-bool (test-term-sort-membership cxu
id))
 26 : eq if true then CXU else CYU fi
 = CXU
 27 : eq if false then CXU else CYU fi
 = CYU
 28 : eq CXU == CYU = #!! (coerce-to-bool (term-equational-equal cxu cyu))
 29 : eq CXU =b= CYU = #!! (coerce-to-bool (term-equational-equal cxu cyu))
 30 : eq CXU =/= CYU = #!! (coerce-to-bool (not (term-equational-equal cxu cyu))
)
SIMPLE-NAT>
```

Iespējamās komandas ir reduce un execute (bet ne tikai), kur reduce ņem vērā visus vienādojumus, bet execute - tam papildus arī pārejas.

Ar komandu *set trace whole on* iespējams skatīties līdzi kā tiek pārveidots terms:

```
SIMPLE-NAT> reduce s(s(s(0))) + (s(0) + s(s(0))) .
-- reduce in SIMPLE-NAT : s(s(s(0))) + (s(0) + s(s(0)))
```

```
[1]: s(s(s(0))) + (s(0) + s(s(0)))
---> s(s(s(0)) + (s(0) + s(s(0))))
[2]: s(s(s(0)) + (s(0) + s(s(0))))
---> s(s(s(0) + (s(0) + s(s(0)))))
[3]: s(s(s(0) + (s(0) + s(s(0)))))
---> s(s(s(0 + (s(0) + s(s(0))))))
[4]: s(s(s(0 + (s(0) + s(s(0))))))
---> s(s(s(s(0) + s(s(0)))))
[5]: s(s(s(s(0) + s(s(0)))))
---> s(s(s(s(0 + s(s(0))))))
[6]: s(s(s(s(0 + s(s(0))))))
---> s(s(s(s(s(s(0))))))
s(s(s(s(s(s(0)))))) : NzNat
(0.000 sec for parse, 6 rewrites(0.020 sec), 10 matches)
SIMPLE-NAT>
```

10. Praktisks piemērs [4]

```
module SIMPLE-NAT+ {
  protecting (SIMPLE-NAT)
  op _-_ : Nat Nat -> Nat
  op _<_ : Nat Nat -> Bool
  vars N M : Nat
  eq 0 - M = 0 .
  eq N - 0 = N .
  eq s(N) - s(M) = N - M .
  eq 0 < s(N) = true .
  eq N < 0 = false .
  eq s(N) < s(M) = N < M .
}
module GCD {
  protecting (SIMPLE-NAT+)
  op gcd : Nat Nat -> Nat
  vars N M : Nat
  ceq gcd(N, M) = gcd(M, N) if N < M .
  eq gcd(N, 0) = 0 .
  eq gcd(N, N) = N .
  ceq gcd(s(N), s(M)) = gcd(s(N) - s(M), s(M)) if (not (N < M)) and (not (M == N))
}
```

Piemērs ņemts no [4], bet uzlabots - divas rindiņas: “eq gcd(N, N) = N .”
un pēdējai pielikts klāt and nosacījums. Citādi atbilde vienmēr 0.

Lai pārbaudītu mēģiniet šādi:

```
select GCD
reduce gcd( s(0), s(s(0)) ) .
```

11. Operatoru īpašības

Operatoriem iespējams definēt īpašības:

1. asociativitāte $((a + x) + c = a + (x + c))$-

   ```
   op _and_ : Bool Bool -> { assoc }
   ```

2. komunikativitāte $(a + x = x + a)$ -

   ```
   op _and_ : Bool Bool -> { comm }
   ```

3. idempotence $(x + x = x)$ -

   ```
   op _and_ : Bool Bool -> { idem }
   ```

4. nulles eksistence $(x + 0 = x)$

   ```
   op _and_ : Bool Bool -> { id: true}
   ```

12. Parametri

Lietosim sekojošu ILIST moduli ar diviem parametriem.

```
module ILIST ( IDX :: TRIV, DAT :: TRIV ) {
  [ Ilist ]
  [ Elt.DAT < ErrD ]
  op undef : -> ErrD
  op empty : -> Ilist
  op put : Elt.IDX Elt.DAT Ilist -> Ilist
  op _[_] : Ilist Elt.IDX -> ErrD
  -- ---------------------------- saakot ar ‘‘--’’ - tas ir komentaars
  vars I I’ : Elt.IDX
  var D : Elt.DAT
  var L : Ilist
  eq put(I,D,L) [ I’ ] = if I == I’ then D else L [ I’ ] fi .
  eq empty [ I ] = undef .
}
```

TRIV - iebūvētais modulis ar sortu Elt. Ilist - iebūvēts saraksts. ErrD - tips ar iespējamām kļūdām, implicēti definēts; Elt.DAT tips ir apakšsorts tam.

Kā to lietot? Importējam arī SIMPLE-NAT (skat. iepriekš) moduli un rakstām:

```
module NAT-ILIST {
  protecting (ILIST(IDX <= view to SIMPLE-NAT { sort Elt -> Nat },
                   DAT <= view to SIMPLE-NAT { sort Elt -> Nat }))
}
```

Šādi Elt "aizstājam" kopā ar Nat skaitļiem abos gadījumos. Izveidojas jauns modulis, kurša darbojas kā Ilists, bet ar naturāliem skaitļiem.

Iespējams definēt skatus ar

```
view V from TRIV to SIMPLE-NAT { sort Elt -> Nat }

module NAT-ILIST {
  protecting (ILIST(IDX <= V, DAT <= V))
}
```

Sekojoši iespējams rakstīt arī vēl īsāk:

```
module NAT-ILIST {
    protecting (ILIST(V,V))
  }
```

Izvēloties NAT-ILIST, iespējama šāda saziņa ar CafeOBJ:

```
NAT-ILIST> reduce put(s(0), s(s(0)), empty) [s(0)] .
-- reduce in NAT-ILIST : put(s(0),s(s(0)),empty) [ s(0)
                         ]
s(s(0)) : NzNat
(0.000 sec for parse, 3 rewrites(0.000 sec), 3 matches)
NAT-ILIST> reduce put(s(0), s(s(0)), empty) [s(s(0))] .
-- reduce in NAT-ILIST : put(s(0),s(s(0)),empty) [ s(s(0))
                         ]
undef : ErrD
(0.000 sec for parse, 4 rewrites(0.000 sec), 6 matches)
NAT-ILIST>
```

13. Teorēmu pierādīšana

Dažkārt ir ērti pārliecināties par kāda moduļa īpašībām. T.i. - izvēlēties kādu hipotēzi un to pierādīt.

Ar *open* komandu iespējams atvērt moduli labošanai - t.i., dažādu papildus sortu vai vienādojumu pievienošanai, bet ne labošanai. Šīs izmaiņas darbojas, līdz tiek pielietota komanda *close*. Attiecīgi šajā laikā iespējams veikt pierādījumus.

Piemēram lieto SIMPLE-NAT moduli. Turpmākais ir saziņa, kuras laikā pierādām, ka katram naturālam skaitlim n ne tikai $0+n = n$, bet arī $n+0 = n$. Pierādījums balstās uz bāzi pie $n = 0$, tad induktīvais pieņēmums, ka $a + 0 = a$, kam seko $s(a) + 0 = s(a)$, tātad pēc indukcijas seko k.b.j.

```
SIMPLE-NAT> open SIMPLE-NAT .
-- opening module SIMPLE-NAT.. done.
%SIMPLE-NAT> op a : -> Nat .
%SIMPLE-NAT> eq a + 0 = a .
-
%SIMPLE-NAT> reduce 0 + 0 .
*
-- reduce in %SIMPLE-NAT : 0 + 0
0 : Zero
(0.000 sec for parse, 1 rewrites(0.000 sec), 2 matches)
%SIMPLE-NAT> reduce s(a) + 0 .
-- reduce in %SIMPLE-NAT : s(a) + 0
s(a) : NzNat
(0.000 sec for parse, 2 rewrites(0.000 sec), 4 matches)
%SIMPLE-NAT>
```

14. Tālāki ieteikumi

Lielākā daļa no šī materiāla sagatavoti pēc [4] - tas ir pietiekami elegants dokuments, kas spēj visu izskaidrot. Lietojiet to!

Papildus piemērus iespējams apskatīt [5].

Literatūra

[1] The OBJ Family, `http://www.cs.ucsd.edu/users/goguen/sys/obj.html`, pārbaudīts 2005. g. 17. decembrī.

[2] CafeOBJ `http://tunes.org/wiki/CafeOBJ`, pārbaudīts 2006. g. 3.

[3] CafeOBJ Press release, `http://www-cse.ucsd.edu/~goguen/projs/cafepr.html`, pārbaudīts 2006. g. 3. janvārī.

[4] CafeOBJ Manual, A.T. Nakagawa, Toshimi Sawada, Kokichi Futatsug, `http://www.ldl.jaist.ac.jp/cafeobj/documents.html`

[5] CafeOBJ Official Homepage `http://theta.theta.ro/cafeobj/`, pārbaudīts 2006. g. 3. janvārī.