Laborator 4

2015-2016

Programare Logică

Cuprins

Parametrizări

Privire de ansamblu

- Parametrizarea modulelor în Maude se face cu ajutorul a două elemente de bază:
  - teorii
  - view-uri
- Un modul parametrizat poate avea unul sau mai mulți parametri.
- Fiecare parametru este exprimat printr-o teorie.
- În concluzie, un modul poate fi parametrizat prin una sau mai multe teorii.
- Pentru a obține "un caz particular" al modulului parametrizat, instanțiem parametrii acestuia.
- Instanțierea parametrilor necesită definirea unui view.

Privire de ansamblu

Exemplu: Vrem să specificăm o *listă de elemente oarecare*.

- definim un modul `LIST` parametrizat printr-o teorie.
- teoria exprimă cerințele privind tipul elementelor din listă.
- apoi, dacă vrem să specificăm liste de întregi de exemplu, instanțiem parametrul modulului `LIST` cu numere întregi.

Teorii

Teorii

- Teoriile sunt folosite pentru a declara moduluri interfață.
- Teoriile specifică proprietățile sintactice și semantice ce trebuie îndeplinite de către parametrii.
- Teoriile pot avea:
  - sorturi
  - subsorturi
  - operații
  - variabile
  - ecuații
- Teoriile pot importa teorii sau module.

Teorii

- Sintaxa unei teorii:

```
fth <nume> is
...
endfth
```

- Dacă o teorie conține o ecuație declarată cu atributul `nonexec` (nu se execută), atunci această ecuație nu trebuie să satisfacă nicio regula. De exemplu:
  - poate conține variabile în membrul drept care nu apar în membrul stâng,
  - membrul stâng poate fi o singură variabilă etc.

Teoria TRIV

Teoria predefinită TRIV are doar un sort:

```
fth TRIV is
  sort Elt .
endfth
```

Teoria TRIV este des folosită ca parametru în definițiile structurilor de date parametrizate, cum sunt listele, mulțimile, arborii, care conțin elemente de un tip ce nu necesită cerințe speciale.

Teoria MONOID

O teorie pentru monoizi - o operație binară asociativă, cu element neutru:

```
fth MONOID is
  including TRIV .
  op 1 : -> Elt .
  op _ _ : Elt Elt -> Elt [assoc id: 1] .
endfth
```

- Observați că teoria MONOID importă teoria TRIV.
- Atentie! Un modul nu poate importa niciodată o teorie!
- O teorie poate fi folosită doar ca parametru pentru un modul!

Exercițiul 1

Scrieți o teorie `RING` pentru inele comutative.

Un inel comutativ este o structură $(A, +, *, -, z, e)$ astfel încât:

- $(A, +, -, z)$ este un grup comutativ,
- $(A, *, e)$ este un monoid comutativ,
- distributivitatea lui $*$ față de $+$.

View-uri

View-uri

- Un view este folosit pentru a specifica în ce mod o țintă satisface o teorie sursă.
- Există mai multe moduri în care se întamplă acest lucru, fiecare mod specificând o interpretare particulară a teoriei sursă în țintă.
- În definiția unui view trebuie să indicăm:
  - numele view-ului
  - teoria sursă
  - ținta
  - "traducerea" fiecărui sort și operație din teoria sursă în țintă.

$$\text{view: teorie sursă} \to \text{țintă}$$

- Teoria sursă apare ca parametru al unui modul.

View-uri

- Sintaxa unui view:

```
view <numeView> from <teorieSursa> to <tinta> is
  <traduceri>
endv
```

- Întotdeauna
  - sursa este o teorie!
  - ţinta poate fi un modul sau o teorie!

View-uri

- Sorturile se traduc astfel:

```
sort <identificator> to <identificator> .
```

- Pentru fiecare sort s din teoria sursă trebuie să existe un sort s' în țintă, care să fie traducerea sortului s prin acel view.

View-uri

- Operațiile se traduc astfel:

```
op <identificator> to <identificator> .
op <identificator> : <sorturi> -> <sort> to <identificator> .
op <identificator> to term <termen> .
```

- Traducerile trebuie să păstreze aritatea și tipul operațiilor.
- Traducerile de sorturi și operații trebuie să fie compatibile.

View de la teoria RING la RAT

```
fth RING is
  sort Ring .
  ops z e : -> Ring .
  op _+_ : Ring Ring -> Ring [assoc comm id: z] .
  op _*_ : Ring Ring -> Ring [assoc comm id: e] .
  op -_ : Ring -> Ring .
  vars A B C : Ring .
  eq A + (- A) = z [nonexec] .
  eq A * (B + C) = (A * B) + (A * C) [nonexec] .
endfth

 view RingToRat from RING to RAT is
  sort Ring to Rat .
  op e to term 1 .
  op z to 0 .
endv
```

View de la teoria `RING` la `RAT`

- Observați că am omis părțile "evidente" din traduceri
  - `+` din `RING` se traduce în `+` din `RAT`.
  - `*` din `RING` se traduce în `*` din `RAT`.
- Am tradus constanta e într-un termen
  - `1` nu este definit ca o constantă în `RAT` (este definit ca succesor de 0)
- Atenție! Trebuie verificat faptul că axiomele din sursă sunt păstrate și în țintă!
  - acest fapt nu este vizibil în view.
  - cel care definește view-ul trebuie să se asigure că axiomele sunt păstrate (i.e. că view-ul este definit corect)

View-uri

Există mai multe view-uri predefinite de la teoria `TRIV` către principalele module de numere predefinite:

```
view Bool from TRIV to BOOL is
  sort Elt to Bool .
endv

view Nat from TRIV to NAT is
  sort Elt to Nat .
endv

view Int from TRIV to INT is
  sort Elt to Int .
endv

view Rat from TRIV to RAT is
  sort Elt to Rat .
endv

view Float from TRIV to FLOAT is
  sort Elt to Float .
endv
```

Module parametrizate

Module parametrizate

- Sintaxa unui modul parametrizat:

```
fmod <nume>{X1 :: T1,...,Xn :: Tn} is
...
endfm
```

- Partea `{X1 :: T1,...,Xn :: Tn}` se numește interfața.
- Fiecare pereche `Xi :: Ti` este un parametru (formal):
  - `Xi` este *numele parametrului*
  - `Ti` este tipul parametrului, i.e. o *teorie*

Module parametrizate

- Într-un modul parametrizat, toate sorturile care provin din teoriile din interfață trebuie adresate cu numele parametrului.
- Exemplu:
  pentru un parametru `Xi :: T`, fiecare sort `s` din `T` va fi adresat cu `Xi$s`.
- Într-un modul parametrizat cu interfața `{X1 :: T1,...,Xn :: Tn}`, orice sort nou `s` (i.e. care nu provine din teoriile din interfață) trebuie declarat și folosit sub forma:

  `s{X1,...,Xn}`.

Exemplul 1

Amintim următorul modul pentru liste de numere întregi:

```
fmod LIST-INT is
   protecting INT .
   sort List .
   subsort Int < List .
   op nil : -> List .
   op _ _ : List List -> List [assoc id: nil] .
endfm
```

Definim un modul pentru liste de elemente oarecare:

```
fmod LIST{X :: TRIV} is
   sort List{X} .
   subsort X$Elt < List{X} .
   op nil : -> List{X} .
   op _ _ : List{X} List{X} -> List{X} [assoc id: nil] .
endfm
```

Instanțierea modulelor parametrizate

- Instanțierea este procesul prin care parametrii formali ai unui modul parametrizat sunt legați de parametrii actuali.
- Pentru instanțiere, avem nevoie de un view de la tipul unui parametru formal (i.e. de la teoria respectiva; sursa) către tipul parametrului actual corespunzător (i.e. ținta).
- Instanțierea constă în înlocuirea parametrilor formali cu view-uri adecvate.
- După o instanțiere a unui modul parametrizat este creat un nou modul.

Exemplul 1 (cont.)

Putem defini independent modulul `LIST-INT` prin instanțierea modul parametrizat `LIST{X::TRIV}` astfel:

```
view Int from TRIV to INT is
  sort Elt to Int .
endv

fmod LIST-INT is
  protecting LIST{Int} .
endfm

red 1 2 4 nil 3 5 nil .
```

Exemplul 2

Fie o teorie obținută din TRIV prin adăugarea unei operații unare #:

```
fth TRIV# is
  including TRIV .
  op #_ : Elt -> Elt .
endfth
```

Definim un modul pentru liste de elemente oarecare, dar care include o nouă operație care folosește operatia # din teorie:

```
fmod LIST#{X :: TRIV#} is
   sort List{X} .
   subsort X$Elt < List{X} .
   op nil : -> List{X} .
   op _ _ : List{X} List{X} -> List{X} [assoc id: nil] .
  op apply# : List{X} -> List{X} .
  var I : X$Elt . var L : List{X} .
  eq apply#(nil) = nil .
  eq apply#(I L) = (# I) apply#(L) .
endfm
```

Exemplul 2

Instanțiem LIST# cu un view de la TRIV# la INT:

1. în care # este interpretată ca - unar pentru întregi.

```
view MyInt# from TRIV# to INT is
  sort Elt to Int .
  op #_ to -_ .
endv

fmod LIST#-INT is
  protecting LIST#{MyInt#} .
endfm

red apply#( 2 3 4 ) .
```

2. în care # este interpretată ca adunarea numărului respectiv cu 2.

```
view MyInt#2 from TRIV# to INT is
  sort Elt to Int .
  op # X:Elt to term (X:Int + 2) .
endv

fmod INT-LIST#2 is
  protecting LIST#{MyInt#2} .
endfm

red apply#( 2 3 4 ) .
```

Pe săptămâna viitoare!