Laborator 5 - Maude

2016-2017

Programare Logică

Maude

- Este dezvoltat la:
  - University of Illinois at Urbana-Champaign (UIUC)
  - Stanford Research Institute (SRI)
- `http://maude.cs.uiuc.edu/`
- Aparține familiei OBJ al cărei inițiator a fost

  Joseph Goguen (1941-2006).
- Manual de Maude

Maude

- Este un limbaj bazat pe logica rescrierii și logica ecuațională.
- Semantica operațională este bazată pe rescriere, ceea ce permite utilizarea limbajului Maude în demonstrarea automată.
- Poate fi un instrument util în dezvoltarea de software, deoarece permite atât specificarea, cât și analiza unui limbaj de programare. Faptul că este executabil oferă și avantajul unei implementări indirecte.

Maude

- Este un interpretor.
- Comenzile sunt introduse una câte una și sunt executate imediat.
- Un program în Maude este o colecție de module.
- Un modul în Maude are o semnificație matematică clară:
  - sintactic: un modul este o specificație algebrică,
  - semantic: un modul definește o algebră de termeni (un tip abstract de date).

  Pentru mai multe detalii, consultați suportul de curs.

Maude

- Caracteristici ale limbajului Maude:
  - modularizare și parametrizare,
  - definirea tipurilor de date este independentă de implementare,
  - extensibilitate,
  - permite tratarea erorilor și supraîncărcarea operațiilor,
  - poate fi folosit ca demonstrator.

Instalare

- Metoda directa
- Eclipse
- **Maude REPL**

Laboratorul 5

TODO

- [ ] Cum arată un program în Maude?
- [ ] Cum funcționează Maude?
- [ ] Cum scriem specificații în Maude?

Un program în Maude

- Un program în Maude este o colecție de module.
- Structura generală a unui modul în Maude:
  - numele modulului,
  - importuri de alte module,
  - declararea sorturilor și a subsorturilor,
  - declararea operațiilor,
  - declararea variabilelor,
  - ecuații.
- Comentarii: *** înaintea unei linii.

Un prim exemplu

Numerele naturale cu adunare definite folosind axiomatizarea lui Peano:

```
fmod MYNAT-SIMPLE is
     sort Nat .
     op 0 : -> Nat .
     op succ : Nat -> Nat .
     op plus : Nat Nat -> Nat .
     vars N M : Nat .
     eq plus(0, M) = M .
     eq plus(succ(N), M) = succ(plus(N, M)) .
 endfm
```

Un prim exemplu - reprezentarea matematică

Specificația algebrică $\texttt{MYNAT}= (S, \Sigma, \Gamma)$:

- Signatura multisortată (S, Σ) (sorturi + operații):
  - $S = \{\texttt{Nat}\}$
  - $\Sigma = \{\texttt{0}, \texttt{succ}, \texttt{plus}\}$
- Γ (ecuații):
  - $\Gamma = \{\texttt{plus}(\texttt{0}, \texttt{M}) \doteq \texttt{M},$
    $\texttt{plus}(\texttt{succ}(\texttt{N}), \texttt{M}) \doteq \texttt{succ}(\texttt{plus}(\texttt{N}, \texttt{M}))\}$

Termeni:

- "orice combinație bine formată între variabile și operații"
- date de tip `MYNAT-SIMPLE`
- exemple:
  - $\texttt{0}, \texttt{succ}(\texttt{0}), \texttt{plus}(\texttt{succ}(\texttt{0}), \texttt{succ}(\texttt{0}))$
  - $\texttt{plus}(\texttt{succ}(\texttt{0}), \texttt{succ}(\texttt{succ}(\texttt{0})))$
  - $\texttt{plus}(\texttt{plus}(\texttt{succ}(\texttt{0}), \texttt{succ}(\texttt{0})), \texttt{succ}(\texttt{succ}(\texttt{0})))$

Un `modul` în Maude

- Declararea unui modul:
  - `fmod <nume> is ... endfm`
- Declararea sorturilor:
  - `sort <nume> .`
  - `sorts <nume1> ... <numeN> .`
- Declararea operațiilor:
  - `op <nume> : <aritate> -> <sort rezultat> .`
  - `ops <nume1> ... <numeN> : <aritate> -> <sort rezultat> .`
- Declararea variabilelor:
  - `var <nume> : <sort> .`
  - `vars <nume1> ... <numeN> : <sort> .`
- Declararea ecuațiilor:
  - `eq <termen1> = <termen2> .`

Comanda `red`

- Maude trateaza toate ecuațiile ca reguli de rescriere (sunt aplicate de la stânga la dreapta).
- Orice termen bine-format fie se poate rescrie de o infinitate de ori, fie poate fi adus la forma normală.
- Comanda Maude pentru a aduce un termen la forma sa normală este

  `reduce` < `termen` > sau `red` < `termen` >.

- De exemplu, în modulul `MYNAT-SIMPLE` formele normale sunt:
  `0`, `succ(0)`, `succ(succ(0))`, ...
- Pentru a vedea/ascunde pașii de rescriere făcuți de o comanda `red` se folosesc comenzile

```
set trace on .
set trace off .
```

Exercițiul 1

Încărcați în Maude modulul `MYNAT-SIMPLE` și testați comenzile:

- `red plus(succ(0),succ(0)) .`
- `red plus(succ(0),succ(succ(0))) .`
- `red plus(plus(succ(0),succ(0)),succ(succ(0))) .`
- `red succ(succ(0)) == plus(succ(0),succ(0)) .`

Import de module

Modulele pot fi importate folosind cuvintele cheie:

- `protecting`
  - se folosește atunci când datele definite în modulul importat nu sunt afectate de operațiile sau de ecuațiile noului modul
  - nu se construiesc date noi pe sorturi vechi (no junk) și nu sunt identificate date care în modulul inițial erau diferite (no confusion)
- `extending`
  - permite apariția unor date noi pe sorturile vechi (junk) dar nu permite identificarea datelor care în modulul inițial erau diferite (no confusion)
- `including`
  - nu are restricții

Import de module

- Toate definițiile dintr-un modul importat se văd în modulul care importă.
- Atenție: variabilele nu se importă!
- Totuși, diferențele între importuri sunt destul de subtile și țin mai mult de semantică.
- Modulul de mai jos extinde `MYNAT` adăugând și înmulțirea numerelor naturale:

```
fmod MYNAT-SIMPLE* is
     including MYNAT-SIMPLE .
     op mult : Nat Nat -> Nat .
     vars M N : Nat .
     eq mult(0, M) = 0 .
     eq mult(succ(N), M) = plus(mult(N, M), M) .
endfm
```

Notația infixă

- Amintiți-vă forma prefixă, infixă și postfixă a unei expresii.
- În Maude, puteți declara operații în forma infixă astfel:

```
op _+_ : Int Int -> Int .
op _! : Nat -> Nat .
op if_then_else_ : BoolExp Stmt Stmt -> Stmt .
op _?_:_ : BoolExp Exp Exp -> Exp .
```

Exercițiul 2

- Rescrieți modulele `MYNAT-SIMPLE` și `MYNAT-SIMPLE*`, renumind operația `succ` cu `s`, operațiile `plus` și `mult` cu $+$ și, respectiv, $*$ și folosind notația infixă pentru ele.
- Observați ce se întâmplă dacă încercați să reduceți o expresie ce conține atât $+$, cât și $*$, fără paranteze.

Comanda `parse`

- Forma infixă dă naștere unor probleme de parsare (vezi expresia `x + y * z`).
- Comanda `parse` parsează sintaxa unui termen (stabilește dacă este bine-format sau nu).
- Pentru a vedea/ascunde parantezele dintr-un termen se folosesc comenzile

```
set print with parentheses on .
set print with parentheses off .
```

Atributul `prec`

- Pentru a da priorități operațiilor în vederea reducerii numărului de paranteze se folosește atributul care stabilește precedența:

```
op _+_ : Nat Nat -> Nat [prec 33] .
op _*_ : Nat Nat -> Nat [prec 31] .
```

- Cu cât precedența este mai mică, cu atât operația este "mai puternică".

Atributele `assoc, comm, id`

- Multe operații binare sunt asociative (A), comutative (C) sau au identitate (I).
- De exemplu adunarea numerelor naturale este asociativă, comutativă și are identitatea `0`.
- Aceste trei proprietăți se declară ca atribute:
  - `assoc`
  - `comm`
  - `id: <term>`
  - `left id: <term>`
  - `right id: <term>`
- Exemplu:

```
op _+_ : Nat Nat -> Nat [assoc comm prec 33] .
op _*_ : Nat Nat -> Nat [assoc comm prec 31] .
```

Atributele `assoc, comm, id`

Deși proprietățile (A), (C), (I) pot fi declarate prin ecuații, ele se declară ca atribute:

- ecuațiile care definesc asociativitatea și comutativitate duc la neterminarea rescrierii.
- efectul atributelor este acela că rescrierea se face pe clase de termeni modulo asociativitate și comutativitate.

Subsorturi

- Unele sorturi pot fi declarate ca fiind subsorturi ale altor sorturi.
- Specificație algebrică ordonat sortată.
- Se folosește cuvântul cheie `subsort` și $<$.

```
fmod MYNAT is
   sorts Zero NzNat Nat .
   subsort Zero NzNat < Nat .
   op 0 : -> Zero .
   op s_ : Nat -> NzNat .
   op _+_ : Nat Nat -> Nat [assoc comm prec 33] .
   op _*_ : Nat Nat -> Nat [assoc comm prec 31] .
   vars N M : Nat .
   eq 0 + M = M .
   eq s(N) + M = s(N + M) .
   eq 0 * M = 0 .
   eq s(N) * M = (N * M) + M .
endfm
```

Atributul `ditto`

- Atributul `ditto` poate fi dat unei operații care este deja supraîncarcată pentru un subsort (în același modul sau într-un submodul).
- Acest atribut spune că operația supraîncărcată are aceleași atribute ca varianta sa pentru subsort.

```
ops _+_ _*_ : Nat Nat -> Nat [assoc comm].
op _+_ : NzNat Nat -> NzNat [ditto] .
op _*_ : NzNat NzNat -> NzNat [ditto] .
```

Constructori

- Operația + din modulul `MYNAT` este definită prin inducție (pe argumentul doi) pe numere naturale de forma `0` și `s(N)`. Operația + este definită complet în functie de `0` și `s`.
- Se poate arăta, folosind ecuațiile date, că orice termen ce conține `0`, `s` și + este echivalent cu un termen ce conține doar `0` și `s` (formă normală).
- Operațiile `0` și `s` sunt suficiente pentru a construi orice număr natural. Din acest motiv se numesc constructori.
- Pentru a preciza într-o specificație că o operație este constructor se poate folosi atributul `ctor`.

Definirea operațiilor

- Nu există o rețetă cum se definesc operațiile 'definite' (pe baza constructorilor), dar o idee principală este:

  Definiți comportamentul operațiilor pe fiecare constructor!

- Această idee a fost aplicată și în cazul modulului `MYNAT`: am definit operatorii + si * intâi pe `0` și apoi pe `s` (inductiv).

- În general, dacă `c1,...,cn` sunt constructorii și `d` este o operație nouă, ar trebui sa definim cel puțin următoarele ecuații:

```
eq d(c1(X1,...)) = ...
...
eq d(cn(Xn,...)) = ...
```

- Aceasta nu este o garanție, dar este un principiu destul de bun pentru a fi urmărit.

Rezumat

- Un program este o colecție de module, adică de specificații.
- Modelul matematic al unei specificații este o algebră de termeni.
- O execuție (reducere) este o rescriere în algebra de termeni asociată.

Practică

Exercițiul 3 - Numere întregi

Completați modulul de mai jos pentru a defini o specificație pentru numere întregi:

```
fmod MYINT is
   sort Int .
   op 0 : -> Int .
   op s_ : Int -> Int .
   op p_ : Int -> Int .
   op _+_ : Int Int -> Int .
   op _-_ : Int Int -> Int .
   op _*_ : Int Int -> Int .
   ...
endfm
```

- Puteți adăuga ce atribute vreți.
- Puteți folosi ce subsorturi vreți.
- Eventual, puteți importa modului `MYNAT`.

Pe săptămâna viitoare!