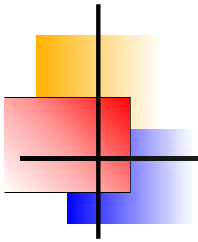
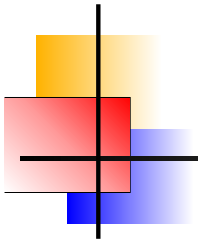


Introducere în limbajul Maude

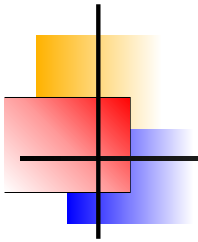
http://maude.cs.uiuc.edu/



Maude

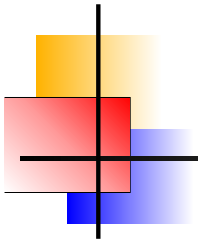
- Este dezvoltat la:
  - University of Illinois at Urbana-Champaign (UIUC)
  - Stanford Research Institute (SRI)
- Aparține familiei OBJ, al carei inițiator a fost Joseph Goguen(1941 - 2006).

All About Maude - A High-Performance Logical Framework, LNCS 4350, Springer, 2007



Maude

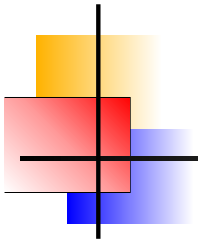
- Este un limbaj bazat pe logica rescrierii. In acest curs vom aprofunda fragmentul bazat pe logica ecuațională.
- Un program este o mulțime de module. Un modul în Maude are o semnificație matematică:
  sintactic, un modul este o specificație algebrică,
  semantic, un modul definește o algebră de termeni (un tip abstract de date).
- 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 este o mulțime de module.
- Modulele pot fi scrise în fișiere sau direct în linia de comandă.
- Modulele pot fi importate. Modulul predefinit `BOOL` este importat de orice modul.

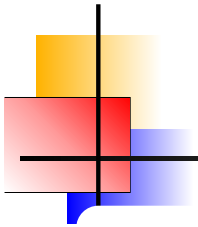
Lectie introductiva (Denisa Diaconescu)



Comenzi sistem

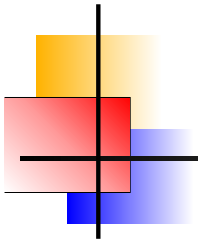
in *../nume-fisier.mod*
show module *nume-modul* .
select *nume-modul* .
show *sorts* .
show *ops* .
reduce *termen* .
reduce *termen1* == *termen2* .
parse *term* .
set trace on .
set trace off .
quit

Manual de Maude (html)



```
Maude> in PL/Exemple-Curs1.maude
==========================================
fmod MYNAT
Maude> show module MYNAT .

 fmod MYNAT is
  sort Nat .
  op 0 : -> Nat .
  op s_ : Nat -> Nat .
 endfm

Maude> select MYNAT .
```

```
Maude> parse s s s s s 0 .
Nat: s s s s s 0

Maude> reduce s s s s s 0 .
reduce in MYNAT : s s s s s 0 .
rewrites: 0 in 6641663802ms cpu (0ms real) (0 rew/sec)
result Nat: s s s s s 0

Maude> reduce  s s s 0 ==  s s 0 .
reduce in MYNAT : s s s 0 == s s 0 .
rewrites: 1 in 10534570934ms cpu (0ms real) (0 rew/sec)
result Bool: false
```



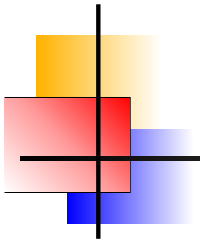
Exemplul 1

```
fmod MYNAT is
  sorts Nat .
  op 0 : -> Nat .
  op s_ : Nat -> Nat .
endfm
```

Acest modul introduce tipul de date Nat.
Datele de tip Nat sunt:

`0, s 0, s s 0, s s s 0, s s s s 0, s s s s s 0, ...`

Reprezentarea matematică:
$S = \{Nat\}$, $\Sigma = \{0 :\rightarrow Nat, s : Nat \rightarrow Nat\}$
(S, Σ) este o signatură și definește o clasă de algebre (structuri, structuri algebrice). Algebra $(\mathbb{N}, 0, succesor)$ este un obiect "privilegiat" al acestei clase.

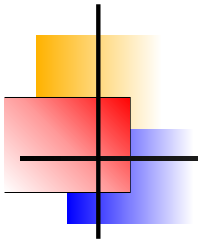


fmod ... endfm

```
fmod MYNATLIST is
protecting MYNAT .
sort ListNat .
subsort Nat < ListNat .
op nil : -> ListNat .
op _;_ : ListNat ListNat -> ListNat [ assoc ] .
var L : ListNat .
eq nil ;  L = L .
eq L ; nil = L .
endfm
```

Structura generală a unui modul:

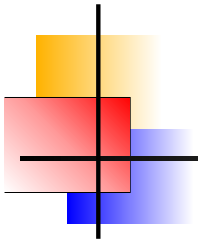
- importuri (`protecting`, `extending`, `including`),
- declararea sorturilor și a subsorturilor,
- declararea operațiilor,
- declararea variabilelor,
- ecuații.



Importuri

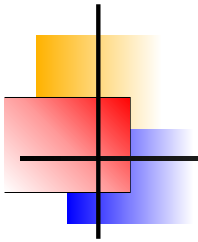
Există trei modalități de importare a modulelor

- **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 in modulul initial 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



Exemplul 2

```
fmod MYNATLIST is
protecting MYNAT .
sort ListNat .
subsort Nat < ListNat .
op nil : -> ListNat .
op _;_ : ListNat ListNat -> ListNat [ assoc ] .
var L : ListNat .
eq nil ;  L = L .
eq L ; nil = L .
endfm
```

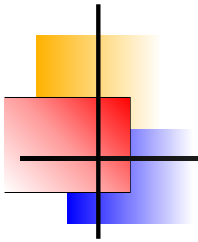


Exemplul 2

Reprezentarea matematică:

```
sort Nat ListNat .
subsort Nat < ListNat .
op 0 : -> Nat .
op s_ : Nat -> Nat .
op nil : -> ListNat .
op _;_ : ListNat ListNat -> ListNat .
```

Signatura (ordonat-sortată): $((S, \leq), \Sigma)$
$S = \{Nat, ListNat\}$, $\leq \subseteq S \times S$, $\leq = \{(Nat, ListNat)\}$
$\Sigma = \{0 :\rightarrow Nat, s : Nat \rightarrow Nat, nil :\rightarrow ListNat,$
$\quad ; : ListNat\ ListNat \rightarrow ListNat\}$



Exemplul 2

Reprezentarea matematica:

```
op _;_ : ListNat ListNat -> ListNat [ assoc ] .
var L : ListNat .
eq nil ;  L = L .
eq L ; nil = L .
```

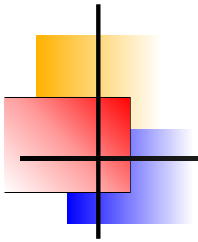
Prezentare (mulțime de ecuații):

$(\gamma_1)\forall L.{}_{ListNat}(nil; L = L)$

$(\gamma_2)\forall L.{}_{ListNat}(L; nil = L)$

$(\gamma_3)\forall L.{}_{ListNat}\forall P.{}_{ListNat}\forall Q.{}_{ListNat}((L; P); Q = L; (P; Q))$

$\Gamma = \{\gamma_1, \gamma_2, \gamma_3\}$

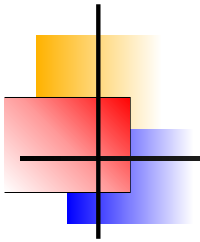


Exemplul 2

```
fmod MYNATLIST is
protecting MYNAT .
sort ListNat .
subsort Nat < ListNat .
op nil : -> ListNat .
op _;_ : ListNat ListNat -> ListNat [ assoc ] .
var L : ListNat .
eq nil ;  L = L .
eq L ; nil = L .
endfm
```

Reprezentarea matematica a unui modul este o specificatie algebrica ordonat-sortată

$$((S, \leq), \Sigma, \Gamma)$$



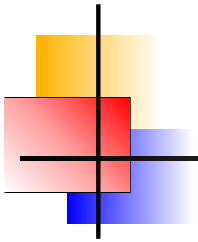
Exemplul 2

Date de tipul ListNat sunt:

```
 nil ; s 0
 s s 0 ; s 0 ; 0
(nil ; s 0) ; s s 0 ; nil ; 0
```

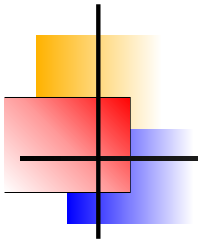
Aceste date le numim **expresii** sau **termeni**.
Un program este un modul, adică o specificație; modelul matematic al unei specificații este o algebră de termeni; o execuție este o rescriere în algebra de termeni asociată.

```
Maude> select MYNATLIST .
Maude> reduce  (nil ; s 0) ; s s 0 ; nil ; 0 .
reduce in MYNATLIST : (nil ; s 0) ; s s 0 ; nil ; 0 .
rewrites: 2 in 2753904389ms cpu (0ms real)
result ListNat: s 0 ; s s 0 ; 0
```



Exemplul 2

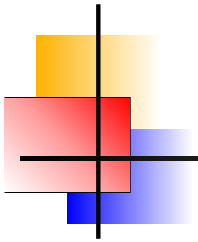
```
Maude> set trace on .
Maude> reduce  (nil ; s 0) ; s s 0 ; nil ; 0 .
reduce in MYNATLIST : (nil ; s 0) ; s s 0 ; nil ; 0 .
********** equation
eq nil ; L = L .
L --> s 0
nil ; s 0 ---> s 0
********** equation
eq nil ; L = L .
L --> 0
nil ; 0 ---> 0
rewrites: 2 in 2753904388ms cpu (12ms real)
result ListNat: s 0 ; s s 0 ; 0
```



Exemplul 2

```
fmod MYNATLIST is
protecting MYNAT .
sort ListNat .
subsort Nat < ListNat .
op nil : -> ListNat .
op _;_ : ListNat ListNat -> ListNat [ assoc ] .
var L : ListNat .
eq nil ;  L = L .
eq L ; nil = L .
endfm
```

Ce date defineste MYNATLIST?

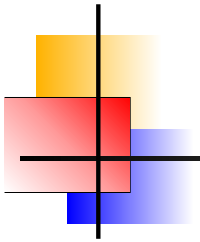


Exemplul 2

```
fmod MYNATLIST is
protecting MYNAT .
sort ListNat .
subsort Nat < ListNat .
op nil : -> ListNat .
op _;_ : ListNat ListNat -> ListNat [ assoc ] .
var L : ListNat .
eq nil ;  L = L .
eq L ; nil = L .
endfm
```

Modulul MYNATLIST definește $\bigcup_{k\geq 0} \mathbb{N}^k$
(secvențele ordonate finite de numere naturale).

```
Maude> reduce (0 ; s 0 ) == (0 ; s 0 ; 0) .
...
result Bool: false
```



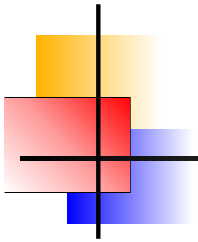
[assoc]

```
op _;_ : ListNat ListNat -> ListNat [ assoc ] .
```

Atributul [assoc] înlocuiește ecuațiile:

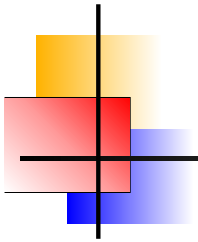
```
eq (L;P);Q = L;(P;Q) .
eq L;(P;Q) = (L;P);Q .
```

Ecuațiile definite cu **eq** sunt transformate în reguli de rescriere. Cele două ecuații care definesc asociativitatea duc la neterminarea rescrierii, deci ele nu pot fi adăugate în secțiunea **eq**. Efectul atributului **[assoc]** este faptul că rescrierea se face pe clase de termeni "modulo asociativitate". În mod asemănător, atributul **[comm]** se folosește pentru a indica faptul că o operație este comutativă.



Exemplul 3

```
fmod MYNATLIST1 is
protecting MYNAT .
sort ListNat .
subsort Nat < ListNat .
op nil : -> ListNat .
op _;_ : ListNat ListNat -> ListNat [ assoc comm ] .
var L : ListNat .
var I : Nat .
eq nil ;  L = L .
eq L ; nil = L .
eq I ; I = I .
endfm

Maude> reduce (0 ; s 0)  == (0 ; s 0 ; 0) .
...
result Bool: true
```



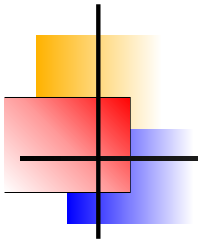
Exemplul 4

```
fmod COMPLEXINT is
protecting INT .
sort ComplexInt .
subsort Int < ComplexInt .
op _+i_ : Int Int -> ComplexInt .
op i_ : Int -> ComplexInt .
op _+_ : ComplexInt ComplexInt -> ComplexInt [ditto] .
var z : Int .
eq 0 +i z = i z .
eq z +i 0 = z .
...
endfm
```

Observați supraîncărcarea operației $+$ si atributul [ditto]

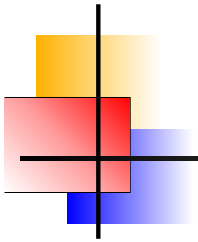
$+ : Int\ Int -> Int$

$+ : ComplexInt\ ComplexInt -> ComplexInt$



Exemplul 5

```
fmod GRUP is
sort  Element .
op e : -> Element .
op  _+_  : Element Element -> Element [ assoc ] .
op -_  : Element -> Element .
vars x y : Element .
eq e + x = x .
eq x + e = x .
eq (- x) + x = e .
eq x + (- x)  = e .
eq - - x = x .
eq - e = e .
eq - (x + y) = (- y) + (- x) .
endfm
```

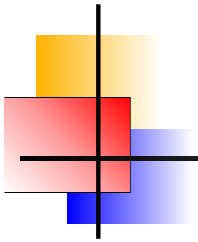


TRS canonic

O ecuație $l = r$ se transformă, prin orientare de la stânga la dreapta, într-o regulă de rescriere $l \to r$. Ecuațiile modulului GRUP determină astfel un sistem de rescriere **canonic**:

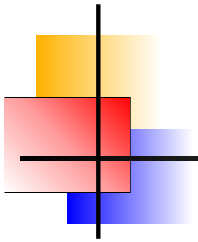
- orice șir de rescrieri se termină,
- ordinea de aplicare regulilor de rescriere nu schimbă rezultatul final.

O ecuație $t_1 = t_2$ din teoria de ordinul I a grupurilor este verificată în orice grup dacă și numai dacă există un termen t astfel încât $t_1 \stackrel{*}{\to} t$ și $t_2 \stackrel{*}{\to} t$.
Maude-ul poate fi utilizat ca demonstrator.



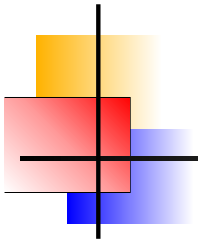
Exemplul 5

```
Maude> select GRUP .
Maude> reduce  x + (-( - y + x)) == y .
reduce in GRUP : x + - (- y + x) == y .
********** equation
eq - (x + y) = - y + - x .
x --> - y
y --> x
- (- y + x) ---> - x + - - y
********** equation
eq - - x = x .
x --> y
- - y ---> y
```



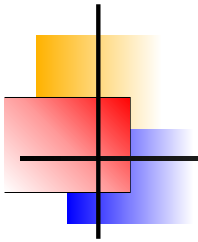
Exemplul 5

```
********** equation
eq x + - x = e .
x --> x
x + - x + y ---> e + y
********** equation
eq e + x = x .
x --> y
e + y ---> y
********** equation
(built-in equation for symbol _==_)
y == y ---> true
rewrites: 5 in 179156223ms cpu (23ms real)
result Bool: true
```



Exemplul 6

```
fmod MYNATLIST is
protecting MYNAT .
sort ListNat .
subsort Nat < ListNat .
op nil : -> ListNat .
op _;_ : ListNat ListNat -> ListNat [ assoc ] .
var L : ListNat .
eq nil ;  L = L .
eq L ; nil = L .
endfm

fmod MYNATLIST2 is
protecting MYNATLIST .
op _<_ : Nat Nat -> Bool .
...
endfm
```



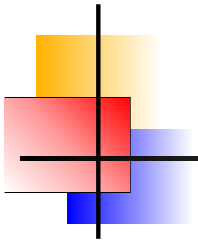
Exemplul 6

```
fmod MYNATLIST2 is
protecting MYNATLIST .
op _<_ : Nat Nat -> Bool .
vars I J : Nat .
eq s I < s J = I < J .
eq  0 < s I  = true .
eq  I < 0 = false .
ceq I ; J = J ; I if (J < I ) .
endfm
```

Observați **ecuația condițională**

```
ceq I ; J = J ; I if (J < I ) .
```

Ce tip de date definește modulul MYNATLIST2 ?

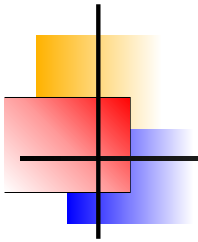


Exemplul 6

```
Maude> select MYNATLIST2 .
Maude> reduce s s s 0 ; s s 0 ;  s s s 0 ; 0 .
...
result ListNat: 0 ; s s 0 ; s s s 0 ; s s s 0
```

Modulul MYNATLIST2 definește liste de numere naturale ordonate crescător.

O problemă de sortare ..., D. Dragulici, Revista de logica

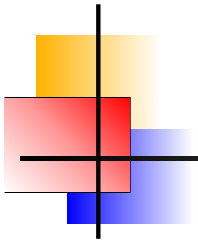


Exemplul 7

```
fmod STACK{X :: TRIV} is
sorts EmptyStack{X} NonEmptyStack{X} Stack{X} .
subsorts EmptyStack{X} NonEmptyStack{X} < Stack{X} .
op empty : -> EmptyStack{X} .
op push : X$Elt Stack{X} -> NonEmptyStack{X} .
op pop_ : NonEmptyStack{X} -> Stack{X} .
op top_ : NonEmptyStack{X} -> X$Elt .
var E : X$Elt .
var S : Stack{X} .
eq top push (E , S) = E .
eq pop push (E , S) = S .
endfm
```

Modulul STACK$\{X\}$ crează o stivă generică.
Parametrul formal X este descris de teoria TRIV

```
fth TRIV is sort Elt. endfh
```

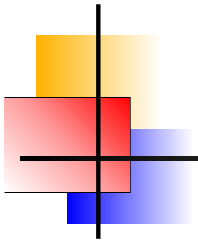


Exemplul 8

```
fth TRIV is
sort Elt.
endfh

fmod STACK{X :: TRIV} is ... endfm
```

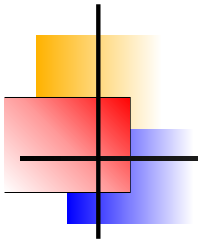
Instanțiind modulul parametru X cu modulul predefinit NAT definim stivele de numere naturale.

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

fmod STACKNAT is
protecting STACK{Inst} .
endfm
```



Exemplul 8

```
Maude> select STACKNAT .
Maude> show sorts .
sort Bool .
sort Zero .      subsort Zero < Nat .
sort NzNat .     subsort NzNat < Nat .
sort Nat .       subsorts NzNat Zero < Nat .
sort EmptyStack{Inst} .
subsort EmptyStack{Inst} < Stack{Inst} .
sort NonEmptyStack{Inst} .
subsort NonEmptyStack{Inst} < Stack{Inst} .
sort Stack{Inst} .
subsorts NonEmptyStack{Inst} EmptyStack{Inst} < Stack{Inst}
```



Limbajul Maude este un limbaj de specificație bazat pe logica ecuațională. Un program este o colecție de module. Execuția este o rescriere. Câteva din caracteristicile acestui limbaj sunt:

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