Université Mohamed Khider – Biskra
Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'Informatique
2ème année Master
Option: Génie Logiciel et Systèmes Distribués
Module : Logique de Réécriture et ses Applications

'Order-Sorted' Specification & les Modules Prédéfinies

Nb: Une grande partie du matériel présenté dans les diapos est inspirée du cours INF 3230 de l'université d'Oslo.

Contenu

- Comparaison de listes et multi-ensembles
- Ordre-sorted spécification
- Modules paramétrés
- Modules prédéfinies
  - BOOL
  - NAT
  - INT
  - RAT and FLOAT
  - STRING
- Quelques commande pour exécuter Maude

Comparaison Lexicographique

Définition (Comparaison lexicographique)

Comparaison lexicographique : la liste L est plus grande que la liste L' SSI

- L est une extension de L', où
- L'élément i de la liste L est plus grand que l'élément i de la liste L' , et les deux liste sont égaux jusqu'au l'élément i.

- Quelle est la liste la plus grande ?
  - '3 4 5' où '3 4 5 6 7' ?
  - '1' où la liste vide ε ?
  - '1 2' ou '3 4' ?

Multiset (multi-ensemble)

- Un multi-ensemble est un ensemble dont le nombre d'occurrence de chaque élément est important :

- Les ensembles {a, b} et {a, b, b} sont les mêmes
- Les multi-ensembles {a, b} et {a, b, b} sont différents
- Les multi-ensembles {a, b, a, b} et {b, b, a, a} sont les même

- Constructeurs :

```
sort Mset .
op none : -> Mset [ctor] .
op _ _ : Mset Nat -> Mset [ctor] .
```

Comparaison de multi-ensemble

- La comparaison entre les multi-ensembles dans un domaine totalement ordonné :
- M est plus grand que M' SSI le plus grand élément dans M est plus grand que le plus grand nombre dans M' après avoir supprimer (le même nombre d'occurrence) d'éléments qui existent dans M et M' .

- {6, 2, 6} et plus grand que {6, 5, 4, 3}
- Qui entre {5, 2, 3, 3} et {5, 3, 2, 2, 2, 1} et {5,4} est le plus grand ? Le plus petit ?

Rappel

- Il faut rien assumer sur la façon dont Maude exécute les équations, à part qu'il les exécute de ' gauche à droite '.
- Vous devez définir les équations d'une façon qu'elles se terminent et qu'elles donnent un résultat correcte quoique ce soit l'ordre avec lequel Maude les exécute.

Sous-sorte : motivation

- Partiellement :
  - Pourquoi pas minus et division sur Nat ?
  - C'est quoi le first et last d'une liste vide ?
- Parfois, on a besoin de deux sortes Nat et Int
  - Besoin de Nat pour des fonctions tel que factorial
  - Nat et Int doivent être liées.
- Sous-classe en Java
  - Un Orang-utang est aussi un Animal

'Order-sorted' Specification

- Une spécification de sortes ordonnées 'Order-sorted Specification' permet l'utilisation des sous-sortes :

```
sorts Int Nat OrangUtan Ape Bird Animal .
subsort Nat < Int .
subsorts OrangUtan < Ape < Animal .
subsort Bird < Animal .
```

- Sous-sortes correspond à l'inclusion des ensembles :
  - Chaque oiseau est un animal
  - $\mathbb{N} \subseteq \mathbb{Z}$

Spécification ‘Order-sorted’ (2)

- Quelques fonctions sont définies uniquement pour les sous-sortes :

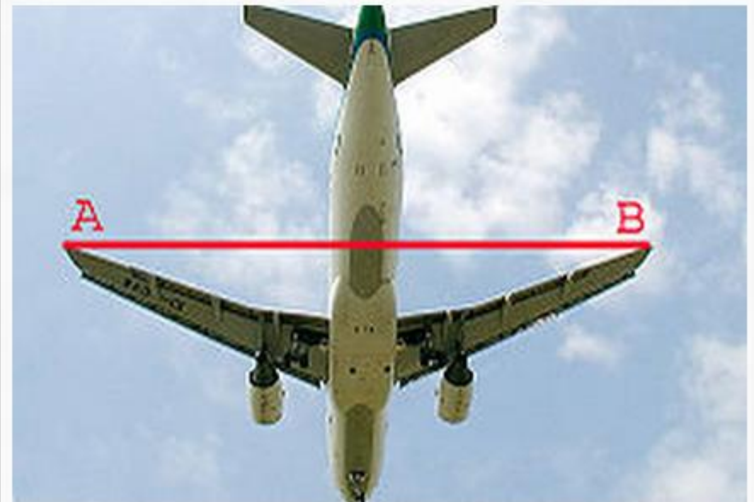
```
ops _+_ _*_ _-_ : Int Int -> Int .
ops fac fib : Nat -> Nat . --- only for Nat
op age : Animal -> Nat .
op IQ : Ape -> Nat .
op wingspan : Bird -> Nat . --- Envergure
```

- wingspan(phoenix), age(phoenix), age(oliver) et IQ(oliver) sont des termes correctes, wingspan(oliver) et IQ(phoenix) ne le sont pas (pour que phoenix est un Bird et oliver est un Orangutan)

- La relation de sous-sortes ≤ est d’ordre partiel

Orangutans

Wingspan



The distance A to B is the wingspan of this Aer Lingus Airbus A320.

Alder Flycatcher

Préliminaires: Ordres Partiels

Définition (Relation binaire)

Une relation binaire R sur un ensemble M est un ensemble $R \subseteq \{(m_1, m_2) \mid m_1, m_2 \in M\}$

On écrit $R(m_1, m_2)$ où $m_1 \; R \; m_2$ pour $(m_1, m_2) \in R$

Définition (Ordre partiel)

Une raltion binaire R est un ordre partiel SSI

- R est *reflexive:* $m \; R \; m$ est vérifiée pour chaque $m \in M$
- R est *antisymmetric:* si $m_1 \; R \; m_2$ et $m_2 \; R \; m_1$ donc $m_1 = m_2$
- R est *transitive:* si $m_1 \; R \; m_2$ et $m_2 \; R \; m_3$, donc $m_1 \; R \; m_3$

Ordre Partiel

Quelles sont les relations d'ordre partiel entre ces relations :

- '≤' sur les nombres naturels
- '<'sur les nombres naturels
- '⊆' sur les ensembles
- $t\,R\,u$ SSI $t \leadsto u$ dans NAT-ADD
- ' ont le même père'
- $x\,R\,y$ SSI y est le grand père de x
- Égalité sur les nombres réels
- 'sous-chaine' de chaine de caractères

Ordre Partiel stricte

Définition (Ordre partiel stricte)

Une relation binaire R est d'ordre partiel stricte SSI R est

- *irreflexive:* $m\ R\ m$ n'est pas vérifiée pour n'import quel m, et
- *transitive*

Theorème

si R est d'ordre partiel stricte , il n'existe aucun m_1, m_2 tel que $m_1\ R\ m_2$ et $m_2\ R\ m_1$

- Qui permet les relations précédentes est d'ordre partiel stricte ?

Ordre-Sorted Specification II

- Termes : si t est un terme de sorte s, et s est une sous-sorte de s', donc t est aussi un terme de sorte s'.
-

```
sorts s s' .                subsort s < s' .
op a : -> s .               op b : -> s' .
op f : s -> s .             op f : s' -> s' .
op g : s -> s' .            op h : s' -> s' .
var X : s' .
```

- Qui parmi

```
a          f(a)          f(b)          f(X)
h(a)       g(f(a))       g(f(b))       g(X)
```

  - sont des termes de sorte s ?
  - sont des termes de sorte s' ?

Utilisation de sous-sortes

- Partiellement :
  - Quelques fonctions (e.g: division et first) ne sont pas défini sur tout le domaine
  - Définie une sous-sorte où les fonctions sont définies
- Division : subsort NzNat pour les nombre naturels positifs

```
sorts NzNat Nat .
   subsort NzNat < Nat .
op 0 : -> Nat [ctor] .
op s : Nat -> NzNat [ctor] .
...
op _/_ : Nat NzNat -> Nat .
```

Utilisation de sous-sortes (2)

- Sous-sorte NeList pour non-empty (non vide) listes :

```
 sorts Liste NeList .
     subsort NeList < List .
op nil : -> List [ctor] .
op _ _ : List Nat -> NeList [ctor] .
...
ops first last : NeList -> Nat .
op length : List -> Nat .
op rest : NeList -> List .
op concat : List List -> List .
op reverse : List -> List .
```

- Variables sur les sous-sortes:

```
 var NEL : NeList .
 eq first (nil N ) = N .
 eq first (NEL N ) = first(NEL) .
```

Sous-sortes : Entiers

- Nombres Naturels :

```
sorts Zero NzNat Nat .
subsort Zero NzNat < Nat .
op 0 : -> Zero [ctor] .
op s : Nat -> NzNat [ctor] .
```

- Nombres non positifs:

```
sorts Neg NzNeg .
subsorts Zero NzNeg < Neg .
```

- Entiers : `subsorts Nat Neg < Int .`

Sous-sortes : Entiers (2)

- Constructeur pour les nombres négatifs

Version 1 : negation d'un nombre positif

```
op -_ : NzNat -> NZNeg [ctor] .
```

Version 2 : 'fonction prédécesseur'

```
op p : Neg -> NZNeg [ctor] .
```

Sous-sortes : éléments de listes

- Éléments dans une liste / multi-ensemble/ ....
- Avant :

```
Sorte Liste .
op nill : -> List [ctor] .
op _ _ : -> List Nat -> List [ctor] .
```

Une liste a la forme `nill s(0) 0 s(s(0))`

- Nouvelle version: un nombre est aussi une liste

```
sorte Liste .
subsort Nat < List .
op nill : -> List [ctor] .
op _ _ : -> List list -> List [ctor] .
```

Sous-sortes : éléments de listes (2)

- Quelques listes

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

C'est plus élégant mais contient des problèmes

- Supprimer les parenthèses
- Problème avec nill .

Autres utilisation des Sous-sortes

- Un nombre est aussi un ensemble / Multiset/ ....

```
subsort Nat < Set .
subsort Nat < MSet .
```

Définir « erreur / valeurs non définie »

```
sort DefNat . --- Nat w/ ''default'' value
subsort Nat < DefNat .
op noNat : -> DefNat [ctor] .
```

Membership : Motivation

- Quelques problème avec algèbre multi-sortées :
  - Problème avec le typage statique :

    Quelques « expressions » ne sont pas bien formées
    - S(s(0)) / (s(0) – 0)
    - fac(s(0) - 0)
  - On cherche « sous-sortes » sémantique tel que OrdredList et BinarySearchTree
    - Impossible dans l'algèbre d'ordre-sorted
  - Une solution élégante : membership equational logic (logique équationnel d'appartenance)

Membership spécification (spécification d'appartenance)

- Étendre les spécification order-sorted avec 'axiomes d'appartenances' :

```
mb t : s .
```

Et

```
cmb t : s if condition .
```

Exemple

- Tri pour les listes ordonnées

```
protecting LIST-NAT1 .
sort OrderedList .
subsort OrderedList < List .
var N : Nat . var OL : OrderedList .
mb nil : OrderedList .
cmb OL N : OrderedList if (last(OL) <= N) = true .
```

Ou

```
var L : List .
cmb L : OrderedList if isSorted(L) = true .
```

Programmation paramétrée

- On a besoin des listes des entiers, listes des booléens, listes de listes de nombre naturels, ...
- Deux possibilités :
  - Des sortes paramétriques (parametric sorts), module :
    - Module paramétré LISTE{X :: ELEM} peut être instancié à LIST{Nat} et LIST{Boolean}
  - Ecrire la spécification pour chaque domaine,
- Maude a des mécanismes puissants de paramétrisation
- .... Toutefois, on va pas les voir dans ce cours

Modules prédéfinis (Built-in modules)

- Le fichier prelude.maude est lit automatiquement par Maude durant le démarrage. Ce fichier contient les modules :
  - Valeurs booléennes,
  - Nombres naturels, entiers, rationnels, nombres flottantes du standard IEEE-754 double précision …
    - Non limité à l’exception des flottants
    - Efficacement implémentés
  - Chaines de caractères

Fonctions génériques

- Le module BOOL du Maude (avec les valeur true et false) est les fonction

```
if_then_else_fi
_==_
_=/=_
```

- (Pour toutes les sortes) les valeurs booléennes sont automatiquement inclut dans chaque module
  - Exemple : (dans) fonction pour listes :

```
vars M N : Nat . var L : List .
eq N in nil = false .
eq N in L M = if N == M then true else N in L fi .
--- better: N in L M = (N == M) or (N in L) .
```

- Ainsi, On peut avoir une expression booléenne dans une condition

```
ceq M monus N = 0 if M <= N .
```

BOOL

- Le module BOOL est inclus dans tout les modules

```
fmod TRUTH-VALUE is
    sort Bool .
    op true : -> Bool [ctor special ...] .
    op false : -> Bool [ctor special ...] .
Endfm

fmod TRUTH is
    protecting TRUTH-VALUE .
    ...
endfm
```

BOOL (2)

- Le module BOOL est inclus dans tout les modules

```
fmod BOOL is
    protecting TRUTH .
    op _and_ : Bool Bool -> Bool [assoc comm prec 55] .
    op _or_ : Bool Bool -> Bool [assoc comm prec 59] .
    op _xor_ : Bool Bool -> Bool [assoc comm prec 57] .
    op not_ : Bool -> Bool [prec 53] .
    op _implies_ : Bool Bool -> Bool [prec 61 gather (e E)] .
    ...
    Endfm
```

- Les attributs assoc et com pour ‘associativité’ et ‘commutativité’
- L’attribut special signifie une Implementation C++

Nombre Naturels

- Implémentation des nombres naturels :

```
fmod NAT is
   sorts Zero NzNat Nat . subsort Zero NzNat < Nat .
   op 0 : -> Zero [ctor] .
   op s_ : Nat -> NzNat [ctor iter special (...)] .
   op _+_ : Nat Nat -> Nat [...] .
   op sd : Nat Nat -> Nat [...] .
   op _*_ : Nat Nat -> Nat [...] .
   op _quo_ : Nat NzNat -> Nat [...] .
   ...
   op _<_ : Nat Nat -> Bool [...] .
   op _<=_ : Nat Nat -> Bool [...] .
endfm
```

Nombre Naturels (2)

- ’3’ peut être représente s s s 0 ou 3
- Autres fonctions incluent operations sur les bits
- N’est pas inclut automatiquement dans tous les modules
- Pas de soustraction, mais difference symétrique sd
- Exemple

```
fmod FACTORIAL is protecting NAT .
    op _! : Nat -> Nat .
    var N : Nat .
    eq 0 ! = 1 .
    eq (s N) ! = s N * (N !) .
endfm
```

Entiers

- Les entiers construits utilisent – comme constructeur

```
fmod INT is protecting NAT .
sorts NzInt Int .
subsorts NzNat < NzInt Nat < Int .
op -_ : NzNat -> NzInt [ctor special (...)] .
op -_ : Int -> Int [...] .
op _+_ : Int Int -> Int [...] .
op _-_ : Int Int -> Int [...] .
op _*_ : Int Int -> Int [...] .
op abs : Int -> Nat [...] .
op _<_ : Int Int -> Bool [...] .
...
endfm
```

Entiers (2)

- '-3' peut etre ecris – s s s 0 ou – 3 ou -3
- Uniquement la declaration
- op -_ : NzNat -> NzInt [ctor special (...)] .

  de – est un constructeur

Nombre rationnels et flottants

- Module RAT est une implentation des nombres rationnels
- Module FLOAT est une implentation des nombres flottants
  - Limites
  - Non précis
  - Utilises les nombres rationnels si c’est possible

Strings (les chaines de caractères)

- Les chaines de caractères sont des constantes de sorte String de la forme "exemple de chaine", "Don Quijote" et "module logique de reecriture"

- Les chaines de longueur 1 sont des constantes de sous sorte Char
- Le module STRING définie tout les chaines de caractères et les opérations sur les chaines.
  - Nb : Concaténation : "Don " + "Quijote"

Strings (2)

```
fmod STRING is protecting NAT .
  sorts String Char FindResult .
  subsort Char < String .
  subsort Nat < FindResult .
  op <Strings> : -> Char [special (...)] .
  op <Strings> : -> String [ditto] .
  op notFound : -> FindResult [ctor] .
  op _+_ : String String -> String [...] .
  op length : String -> Nat [...] .
  op substr : String Nat Nat -> String [...] .
  op find : String String Nat -> FindResult [...] .
  op _<_ : String String -> Bool [...] .
  ...
endfm
```

Quelques conseils pour exécuter des Spécifications Maude

- Un fichier peut lire un autre fichier (in ou Load)
- Définir les modules avec des termes de teste
- Un fichier peut contenir des commandes
- 'Le module actuel' est le dernier module introduit
  - Toute les exécutions se faites sur le module courant
  - Commande

  Select MIN-MODULE .

  Définie MIN-MODULE comme le module courant.

Quelques conseils pour exécuter des Spécifications Maude

- Exemple :

```
in nat-add.maude
in boolean.maude

fmod LIST is ... Endfm

fmod TEST-CASES is protecting LIST .
ops list1 list2 : -> List .
eq list1 = nil s(s(0)) s(0) .
eq list2 = nil s(0) s(s(0)) s(s(s(0))) .
endfm

red concat(list1, list2) .
red reverse(list2) .
red first(list1) .
```