# Introduction to Maude

Xiaoxi Dong

March, 2010

# Maude: A reflective language and system

Maude is a high-performance reflective language and system.

Maude supports equational specification and programming and rewriting logic computation in a systematic and efficient way

Some of the most interesting applications of Maude are **metalanguage** applications, in which Maude is used to create **executable** environments for different logics, theorem provers, languages, and models of computation.

Developed by UIUC, US

# Maude System

- Module
  - 1 functional module:
  - 2 System module:
  - 3 Oo module
- Maude
  - Core Maude:
    - 1 2
  - Full Maude:
    - 3

# Maude

- Key concepts
  - Module: a set of definitions
  - Algebra: a set of sets and the operations on them
  - Import: protecting, including, extending
  - Sort: a category of values
  - Subsort: further specific groups all belonging to the same sort.
  - Variable: an indefinite value of a sort. Only a placeholder used in equations **be never used as constants. Never have a definite value assign to them. Cannot carry from one operation to another,
  - Kind:super sort. Maude automatically defins a kind when a sort is defined. Include all the connected components. Important in error handling.

# Maude

- Key concepts
  - Operation: pathway between the sorts. Operator overloading
  - Constructor: ctor, fundamental operation that defines the basics of the algebra. Bare minimum. constant operation is usually a constructor.

    *Maude constants are functions with no independent variable.
  - Flags of equational attributes: assoc comm id (idem) on binary operators, iter for unbinary repeated operators (sssss0=s_^5(0))

# Maude

- Sorts and operations are the core foundation of Maude.
- Common Supplied Modules
  - Library : Nat , QID, STING

# Functional Module

- Creating and Mapping out structures
- Equation: rules to simplifies an expression. Same to normal mathematical equations.
  - Expression with only constructors can not be simplified.
  - Canonical form
  - Non-termination problem of idempotency and identity in recursion
- Conditional equation:  ceq
  - build-in Bool is lack of comparison operations, aka need an extra NAT-BOOL module to define these operations.
  - Maude2 can use **pattern-match** as condition
  - := is used as pattern transformation, or say assignment

# Function Module, contd.

- Membership axiom,membership logic, not simplification
  - Conditional membership axiom
  - Importing and building on other modules
  - matching condition: used in conditional membership axioms.
- Operator and Statement Attributes
  - Equations can carry flags, or *attributes* → should it be only ops can but not eqs??
  - memo: memorization table, quicker reduce, useful when the same expression appears many times
  - prec + a number: precedence
  - Gather + (key symbol): declare gathering pattern for same precedence but not associative operators.
  - owise/otherwise: for negtive pattern

# Importing

- Junk, confusion: allowed but not nice (voilation).
- Junk is to new ground terms, ie ctor and constants
- Confusion is to redefining the already extant terms

- Protecting
  - Maude compiler does not accept either junk or confusion
- Extending
  - Accept junk but not confusion
- Including
  - Accept both

# Operator overloading

- Subsort overloading
- Ad-hoc overloading: No, it is not a good idea.

# kind

- Error term vs non-error term vs kind
- Reduce error term to non error
- Kind  allows us to ignore error expressions sometimes. It might prove constructive in some situations.

# System Module

- Transition that occurs within or between structures
- Rewriting logic rl: state, transition
- Rewriting laws vs equations: p40
- States are constructors
- Order of operations:
  - Deterministic and non-deterministic
  - The *rewrite* command is a default strategy provided by Maude.
  - Other ordering strategies: fair rewrite,
- Conditional Rewrite Logic crl

# Maude Environment

- Fmod
  - set trace on .
  - reduce in MODULE : expr .
- Mod
  - rewrite/rew
  - frewrite/frew: fair rewrite, pick up which rewrite laws to apply such that no law goes ignored.
  - continue x: continue current law for x more rewrites
  - search: the path of laws from the beginning to the end states. =>+ solution involve at least one laws, =>! terminal state, M:State, show path x,

# A Simple State Machine

- A Machine

```
fmod MACHINE is
sort Machine MachineId .
protecting QID .
subsort Qid < MachineId .

op Terminate : -> Machine .
op Start : -> Machine .
op <_> : MachineId -> Machine [ctor] .
op id_ : Machine -> MachineId [ctor] .

var X : MachineId .
var Y : Machine .
eq id < X > = X .
eq < id Y > = Y .

endfm
```

- An Event

```
fmod EVENT is
sort Event EventId .
protecting QID .
subsort Qid < EventId .


op None : -> Event[ctor] .
op evt_(_) : EventId Qid -> Event [ctor]  .
op id_ : Event -> EventId [ctor] .
op e_ : EventId -> Event .


var X : EventId .
var Z : Qid .
eq id evt X ( Z )  = X .

endfm
```

# A Simple State Machine

- The Soup of events and states

```
mod SOUP is
sort Soup .
protecting EVENT .
protecting MACHINE .
subsort Event Machine < Soup .

op null : -> Soup .
op __ : Soup Soup -> Soup [assoc comm id: null] .

endm
```

# A Simple State Machine

- The State Machine

```
mod STATEMACHINE is
including SOUP .
var X : MachineId .
var Y : EventId .
op fire : Soup -> Soup [ctor]

rl [rule1] : evt '3' ('0') < '0' > => < '1' >  .
rl [rule2] : evt '2' ('1') < '1' > => < '0' >  .

rl [rule4] : evt '1' ('0') < '0' > => < '2' >  .

crl [rule3] : evt Y ( X ) < X > => Terminate if X == Y  .

endm
```

# View

- Theories are used to declare module interfaces.
- Eg: Functional theory TRIV -->membership equational logic theories
- The theory TRIV is used very often, for instance in the definition of data structures.
- We use views to specify how a particular target module or theory  is claimed to satisfy a source theory
- There can be many different views, each specifying aparticular interpretation of the source theory in the target

# Analyzing the Model

- Rewriting: depth first
- Search: width first
- LTL: model checking

# Object Oriented Modules

- It is part of the Full Maude: omod
  - *Warning: Expertise and finesse required. It complicates. The programmers may break the safe zone and crash the Maude.
- Declare an object is to declare all the attributes of the Class.
- To use the object, use the symbol

  “<obj# : Sort | attribute, attributes >”
- Objects are above all data structures.
- Inheritance is implemented through class and subclasses. Keyword 'subclass'

# Meta-Programming

- It is a standard library module.
- It is not a part of either Core or Full Maude
- META-LEVEL module, the metamodel of fmod and mod
- Full Maude does not have meta representation in the standard library
- META-TERM META-MODULE