Teaching Formal Methods Based on Rewriting Logic and Maude

Peter C. Ölveczky

University of Oslo

Nov. 6, 2009

Content

Use of rewriting logic and Maude to teach introductory formal methods at University of Oslo

"Widening the Access to Formal Methods"

- Motivate use of formal methods (FM)
- No formal/logic background
- Like to program!

Rewriting Logic

- "Algebraic specifications" for distributed systems

Rewriting Logic

- "Algebraic specifications" for distributed systems
- Data types defined by algebraic equational specification $(S, \leq, \Sigma, E)$
- Concurrent transitions modeles by rewrite rules

$$l: \ [t]_E \longrightarrow [t']_E \ \textbf{if} \ cond$$

Rewriting Logic

- "Algebraic specifications" for distributed systems
- Data types defined by algebraic equational specification $(S, \leq, \Sigma, E)$
- Concurrent transitions modeles by rewrite rules

$$l: \ [t]_E \longrightarrow [t']_E \ \textbf{if} \ cond$$

- Rules not terminating or confluent

Rewriting Logic

- "Algebraic specifications" for distributed systems
- Data types defined by algebraic equational specification $(S, \leq, \Sigma, E)$
- Concurrent transitions modeles by rewrite rules

$$l : \; [t]_E \longrightarrow [t']_E \; \textbf{if} \; cond$$

- Rules not terminating or confluent
- Expressive and general model for distributed systems

Rewriting Logic

- "Algebraic specifications" for distributed systems
- Data types defined by algebraic equational specification $(S, \leq, \Sigma, E)$
- Concurrent transitions modeles by rewrite rules

$$l : \ [t]_E \longrightarrow [t']_E \ \textbf{if} \ cond$$

- Rules not terminating or confluent
- Expressive and general model for distributed systems
- Natural model for distributed objects
  - distributed state: multiset of objects and messages

Maude

Maude: high-performance tool for rewriting logic

- order-sorted and membership equational specification
- deduction modulo A/C/AC

Maude

Maude: high-performance tool for rewriting logic

- order-sorted and membership equational specification
- deduction modulo A/C/AC
- equational reduction
- simulation of one behavior
- reachability analysis by explicit-state breadth-first search
- LTL model checking

Beginner's FM Course in Oslo: Content (I)

- Basic algebraic specification in Maude:
  - order-sorted signatures, terms, equations, memberships,
  - lists, multisets, binary trees
  - quick-sort, merge-sort

Beginner's FM Course in Oslo: Content (I)

- Basic algebraic specification in Maude:
  - order-sorted signatures, terms, equations, memberships,
  - lists, multisets, binary trees
  - quick-sort, merge-sort
- Classic term rewriting theory
  - confluence
  - termination (incl. theory of simplification orderings)

Beginner's FM Course in Oslo: Content (I)

- Basic algebraic specification in Maude:
  - order-sorted signatures, terms, equations, memberships,
  - lists, multisets, binary trees
  - quick-sort, merge-sort
- Classic term rewriting theory
  - confluence
  - termination (incl. theory of simplification orderings)
- Equational logic
  - deduction rules
  - (un)decidability results
  - inductive theorems

Specification in Maude: Lists

Lists in Maude:

```
sorts List NeList .    subsorts Int < NeList < List .

op nil : -> List [ctor] .
op __ : List List -> List [assoc id: nil ctor] .
op __ : NeList NeList -> NeList [assoc id: nil ctor] .
```

Lists in Maude:

```
sorts List NeList .    subsorts Int < NeList < List .

op nil : -> List [ctor] .
op __ : List List -> List [assoc id: nil ctor] .
op __ : NeList NeList -> NeList [assoc id: nil ctor] .

op length : List -> Nat .      ops first last : NeList -> Int .
op rest : NeList -> List .     op reverse : List -> List .

vars I J K : Int .   vars L L' : List .   vars NEL NEL' : NeList

eq length(nil) = 0 .                 eq reverse(nil) = nil .
eq length(I L) = 1 + length(L) .    eq reverse(L I) = I reverse(L)

eq first(I L) = I .                              eq rest(I L) = L .

eq last(L I) = I .
```

Specification in Maude: Merge-Sort

```
op mergeSort : List -> List .
op merge : List List -> List [comm] .

eq mergeSort(nil) = nil .
eq mergeSort(I) = I .

ceq mergeSort(NEL NEL') = merge(mergeSort(NEL), mergeSort(NEL'))
if length(NEL) == length(NEL')  or length(NEL) == s length(NEL')

eq merge(nil, L) = L .
ceq merge(I L, J L') = I merge(L, J L')  if I <= J .
```

Beginner's FM Course in Oslo: Content (II)

Dynamic systems

- rewriting logic
- simple examples: soccer game, NFL, lives of people, coffee bean game, . . .
- object-oriented specification of distributed systems
  - dining philosophers, . . .

Beginner's FM Course in Oslo: Content (II)

Dynamic systems

- rewriting logic
- simple examples: soccer game, NFL, lives of people, coffee bean game, ...
- object-oriented specification of distributed systems
  - dining philosophers, ...
- simulation and reachability analysis
  - search for deadlocks in dining philosophers, etc.

Beginner's FM Course in Oslo: Content (III)

- Communication protocols: alternating bit, sliding windows,
  - search in sliding windows takes time

Beginner's FM Course in Oslo: Content (III)

- Communication protocols: alternating bit, sliding windows,
  - search in sliding windows takes time
- Two-phase commit protocol for distrubuted transactions

Beginner's FM Course in Oslo: Content (III)

- Communication protocols: alternating bit, sliding windows,
  - search in sliding windows takes time
- Two-phase commit protocol for distrubuted transactions
- NSPK authentication protocol
  - model of NSPK for multiple runs + Dolev-Yao intruders
  - search for bad state

Beginner's FM Course in Oslo: Content (III)

- Communication protocols: alternating bit, sliding windows,
  - search in sliding windows takes time
- Two-phase commit protocol for distrubuted transactions
- NSPK authentication protocol
  - model of NSPK for multiple runs + Dolev-Yao intruders
  - search for bad state
- Linear temporal logic properties and model checking

Example: NSPK

Message 1.	$A \rightarrow B$	:	$A.B.\{N_a.A\}_{PK(B)}$
Message 2.	$B \rightarrow A$	:	$B.A.\{N_a.N_b\}_{PK(A)}$
Message 3.	$A \rightarrow B$	:	$A.B.\{N_b\}_{PK(B)}$

```
rl [read-2-send-3] :
   (msg (encrypt (NONCE ; NONCE') with pubKey(A)) from B to A)
   < A : Initiator | initSessions : initiated(B, NONCE) IS >
  =>
   < A : Initiator | initSessions : trustedConnection(B) IS >
   msg (encrypt NONCE' with pubKey(B)) from A to B .
```

Example: NSPK: Search for Bad State

```
Maude> (search [1]
          < "Scrooge" : Initiator |
                          initSessions :
                             notInitiated("Beagle Boys"), ... >
          < "Bank" : Responder | respSessions : emptySession,
                          nonceCtr : 1 >
          < "Beagle Boys" : Intruder |
                          initSessions :
                             notInitiated("Bank"), ... >
        =>*
          C:Configuration
          < "Bank" : Responder |
                           respSessions :
                                trustedConnection("Scrooge")
                                RS:RespSessions > .)
```

Discussion (I)

- Algebraic specification/TRS + modeling/analysis of distributed systems

Discussion (I)

- Algebraic specification/TRS + modeling/analysis of distributed systems
- "Fun" functional and object-oriented programming/modeling
  - simple and intuitive formalism for distributed systems

Discussion (I)

- Algebraic specification/TRS + modeling/analysis of distributed systems
- "Fun" functional and object-oriented programming/modeling
  - simple and intuitive formalism for distributed systems
- Mature and high-performance tool support

Discussion (I)

- Algebraic specification/TRS + modeling/analysis of distributed systems
- "Fun" functional and object-oriented programming/modeling
  - simple and intuitive formalism for distributed systems
- Mature and high-performance tool support
- Easy to model wide range of distributed systems

Discussion (I)

- Algebraic specification/TRS + modeling/analysis of distributed systems
- "Fun" functional and object-oriented programming/modeling
  - simple and intuitive formalism for distributed systems
- Mature and high-performance tool support
- Easy to model wide range of distributed systems
- (NSKP++) motivates use of formal methods

Discussion (II)

- Well motivated introduction of key FM elements
  - deduction systems: equational and rewriting logic
  - verification of program properties:
    - termination and confluence
    - inductive properties
  - formal modeling
  - model checking
  - state space explosion
  - (LTL) properties of distributed systems

Discussion (II)

- Well motivated introduction of key FM elements
  - deduction systems: equational and rewriting logic
  - verification of program properties:
    - termination and confluence
    - inductive properties
  - formal modeling
  - model checking
  - state space explosion
  - (LTL) properties of distributed systems
- Extensions: probabilistic and real-time systems

Discussion (II)

- Well motivated introduction of key FM elements
  - deduction systems: equational and rewriting logic
  - verification of program properties:
    - termination and confluence
    - inductive properties
  - formal modeling
  - model checking
  - state space explosion
  - (LTL) properties of distributed systems
- Extensions: probabilistic and real-time systems
- "Hot" research topic
- Used in industry:
  - new browser security flaws (Microsoft Research)
  - bugs in embedded automotive software (Japan)
  - ...

Discussion (III)

- Course book exists

Discussion (III)

- Course book exists
- Positive student feedback in Oslo
- Different groups at University of Oslo use Maude
- Ex-TA has start-up company developing Maude product

Discussion (III)

- Course book exists
- Positive student feedback in Oslo
- Different groups at University of Oslo use Maude
- Ex-TA has start-up company developing Maude product
- Main complaint: Full Maude (Maude's OO extension) not robust and does not give good error messages