Maude: A Formal Tool and MetaTool for Wide Spectrum Formal Methods

Jose Meseguer[1], Carolyn Talcott[2]

[1]Computer Science Laboratory
SRI International, Menlo Park, CA

[2]Computer Science Department
Stanford University, Palo Alto, CA

Maude's Formal Methodology

Our formal methodology consists of the following *increasingly stronger methods*, to which a system design is subjected:

1. *Formal specification*, results in a first *formal model* of the system.

2. *Execution of the specification* for simulation and debugging purposes.

3. *Model-checking analysis* to find more errors. initial state.

4. *Narrowing analysis*— symbolic model-checking using logical variables.

5. *Formal Proof* — for highly critical properties.

Introducing Maude

An implementation of Rewriting Logic that supports

- Natural representation of many logical systems via mappings of logics
- Concurrent and distributed computation in a natural way
  - Systems can be synchronous, asynchronous, or hybrid, and can communicate through point-to-point, multicast, broadcast, or combinations
- Reflective capabilities such as meta-tools, dynamic adaption and composition

Rewriting Logic

- Static structure of the system described as an algebraic data type: `f(t1,...,tn)`
- Dynamics of a system is described by rewrite rules:
  `rl [l] :  t => t'`
- Concurrent transitions!

```
<a:Node | nbs:{b,c}, seqNos: (a,5) u MAP, ATTS1>
                                  <b:Node | nbs:{a,c}, ATTS2>
       <c:Node | nbs:{a,b}, seqNos : MAP', ATTS3>
   msg 3 To c From b Src d
f(t1,t2,t3,t4)
```

```
<a:Node | nbs:{b,c}, seqNo: (a,6) u MAP, ATTS1'>
 multimsg 6 To {b,c} From a Src a
                                  <b:Node | nbs:{a,c}, ATTS2'>
f(t1',t2',t3',t4')
       <c:Node | nbs:{a,b}, seqNos:(d,3) u MAP'', ATTS3'>
```

Overview of Tools and Case Studies

- an inductive theorem prover,
- a Church-Rosser checker,
- Full Maude, including module algebra and object-oriented modules,
- a proof assistant for OCC, the Open Calculus of Constructions,
- $HOL \rightarrow Nuprl$, (w Cornell and Bell Labs)
- $LinLogic \rightarrow RWLogic$,
- $PTS \rightarrow RWLogic$
- $TileLogic \rightarrow RWLogic$ (w U. Pisa)

Applications Experience

- Reliable Broadcast Protocol Design (w UCSC)
- Active Net algorithms (w UPenn)
- Cryptographic Protocol Analysis (w Denker (SRI))
- Semantics of CAPSL (Millen and Denker (SRI))
- Composable Communication Services
- Interoperation of Heterogeneous Software Architectures

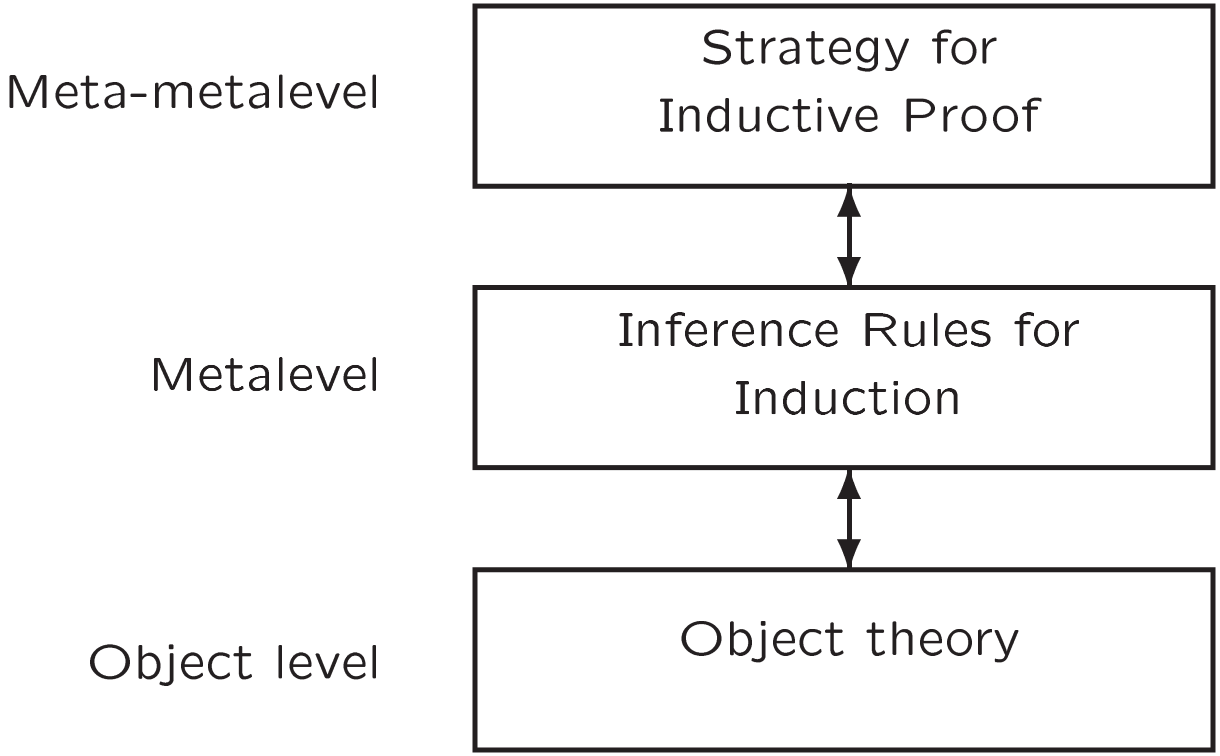
An Inductive Theorem Prover I

The Maude inductive prover is built using Maude's reflective capability at three levels.

- The specifications we are dealing with are equational theories T having an initial algebra semantics.
- The rules of inference for induction are expressed as a rewrite theory at the meta-level
- Proving an inductive theorem for T corresponds to applying the inference rules with some *strategy* expressed at the meta-meta-level.

The *three levels* make the design entirely *modular*, so, for example, the strategy can be changed without any change to the inference rules.

An Inductive Theorem Prover II



The Maude Church-Rosser Checker

The Church-Rosser Checker has a reflective design similar to that of the inductive theorem prover.

It checks equational specifications for satisfaction of the Church-Rosser property and outputs proof obligations that can be used to modify the specification or to verify the CR property.

The tool is currently being extended in two ways:

- To support Knuth-Bendix completion of membership equational logic specifications
- To support coherence completion of rewrite theories

Implementing Full Maude in Core Maude

The methodology for using Maude as a meta tool for developing formal specification tools has been applied in the development of Maude itself.

The basic elements of rewriting logic are implemented in *Core Maude*. *Full Maude* extends this with

- a special syntax for *object-oriented specifications*, and
- a rich *module algebra* of parameterized modules and module composition in the Clear/OBJ style.

Full Maude is formally specified in Core Maude and this formal specification *is* the implementation.

In Maude such executable specifications are easy to build, maintain, and extend, and have competetive performance.

Execution Environments

The same reflective methods used to develop full Maude have been and are being used to develop execution environments for other formal languages

- The Wright architecture description language
    - and its mapping to CSP
- Machine Readable CSP
- The CAPSL security protocal specification language
- (Modular) SOS
- Action Notation
- Actor Specification Diagrams

A Translator from HOL to Nuprl I

HOL has a rich library of theories that, if translated to other systems, could save a lot of effort to the users of those systems.

- Howe defined a map from the HOL logic into the logic of Nuprl, and implemented such a mapping.
- With the help of D. Howe and R. Constable, M.-O. Stehr and J. Meseguer have formally specified it in Maude.
- The result is an *executable formal specification* of the mapping that has translated large HOL libraries Nuprl.

A Translator from HOL to Nuprl II

Correctness of the Translation and Mapping of Proofs

- Using a somewhat more abstract version of the mapping, Stehr and Meseguer have proved a strong correctness result, namely, that it is a map of *entailment systems*.

- This result is of a proof-theoretic nature and hence complements the semantical argument given by Howe.

- Allen and Naumov at Cornell, and Stehr and Meseguer at SRI are jointly studying how this proof-theoretic argument can be used to translate HOL proofs into Nurpl proofs.

Pure Type Systems in Rewriting Logic I

Pure type systems (PTS) contain such important systems as:

- the simply typed and polymorphic lambda calculi,
- a system λP close to the logical framework LF,
- their combination, the calculus of constructions CC.

PTS systems have been specified in Maude giving a representation that is natural and close to mathematical practice.

This provides Maude with an ideal basis for representing higher-order logics.

OCC — The Open Calculus of Constructions

The *open calculus of constructions* (OCC) is an extension of the calculus of constructions that combines advantages of the rewriting logic (first-order, initial semantics, executable) and higher-order pure type systems.

- Similar to membership equational logic, OCC supports *conditional equations* and *conditional assertions* together with an operational semantics based on *conditional rewriting modulo equations*.

- The formal executable specification of OCC in Maude uses the *reflective capabilities of Maude*, yielding orders of magnitude speedups over Lego and Coq in the evaluation of functional expressions.

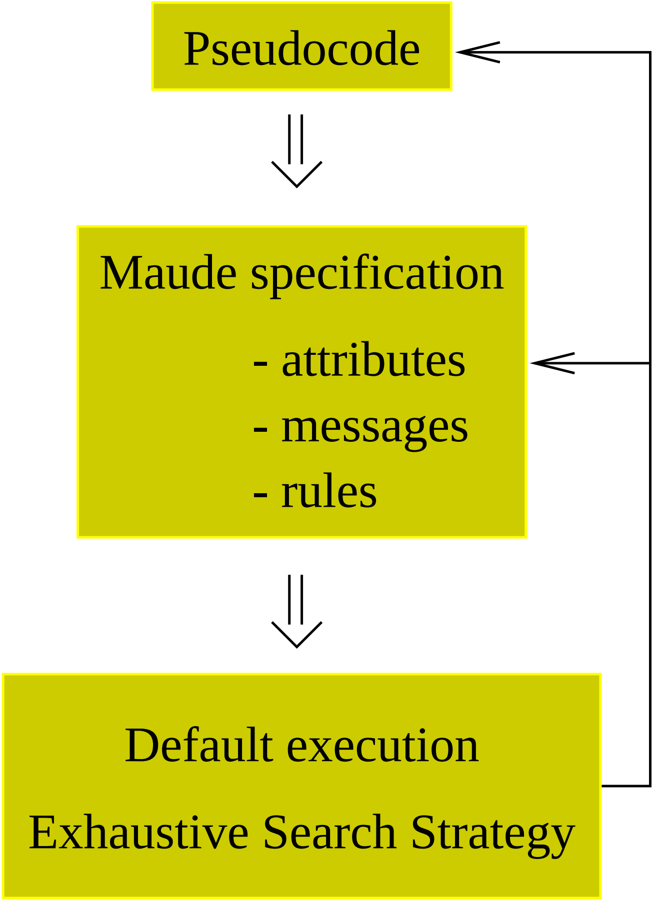
Reliable Broadcast Protocol Case study

A new protocol for Reliable Broadcast in a network with dynamic topology is being developed at UCSC.

The protocol aims to ensure that all nodes satisfying certain connectedness criteria receive broadcast information.

Modelling the protocol in Maude has been part of the protocol design process with substantial benefit.

Overview of Method



RBP

Pseudocode

N^i = set of direct neighbors

ST^i_j = state (active/passive) of node i wrt to source j

SQ^i_j = sequence number

...

Start (i=j):

source node j checks if ST^j_j = passive. If yes, then

- send latest msg for source j to every k in N^j
- ST^j_j := active
- SQ^j_j := SQ^j_j + 1
- ST^j_{jk} := active for all k in N^j
- s^j_j := j

Maude specification

attributes

```
class Node | nbs : OidSet,
             states : Map[Oid,Status],
             seqNos : Map[Oid,Int],
             ...
op active : -> Status .
...
```

messages

```
msg msg_To_From_Src_ :
    Number Oid Oid Oid -> Msg .
msg multimsg_To_From_Src_ :
    Number OidSet Oid Oid -> Msg .
```

conditional rule

```
crl [Start] :
 < J:Node | nbs : OIDSET,
            states : (J,passive) u MAP,
            seqNos : (J,N) u MAP'
            ... >
 => < J:Node | states : (J,active) u Map,
               seqNos : (J, N+1) u MAP',
               ... >
    multimsg N+1 To QIDSET From J Src J
 if OIDSET =/= mt .
```

RBP – Value Added

Original pseudocode informal specification

- had mistakes and inconsistencies,
- did not consider some cases,
- did not make explicit some key assumptions,
- was not executable.

Maude specification

- found deadlocks and unexpected behaviors,
- made explicit all key assumptions,
- is complete and executable,
- allows systematic exploration of all behaviors, and
- is amenable to formal verification.

ReCapping

The Maude Methodology provides

- Maximum benefit via early insertion of formal modeling
- Simplicity and intuitive appeal of Rewriting Logic
- Executability of Rewriting Logic: simulation, testing, debugging
- Formal analysis: e.g. the highly concurrent and nondeterministic behavior of communication protocols
- Formal meta-tool capabilities for building and interoperating formal tools.

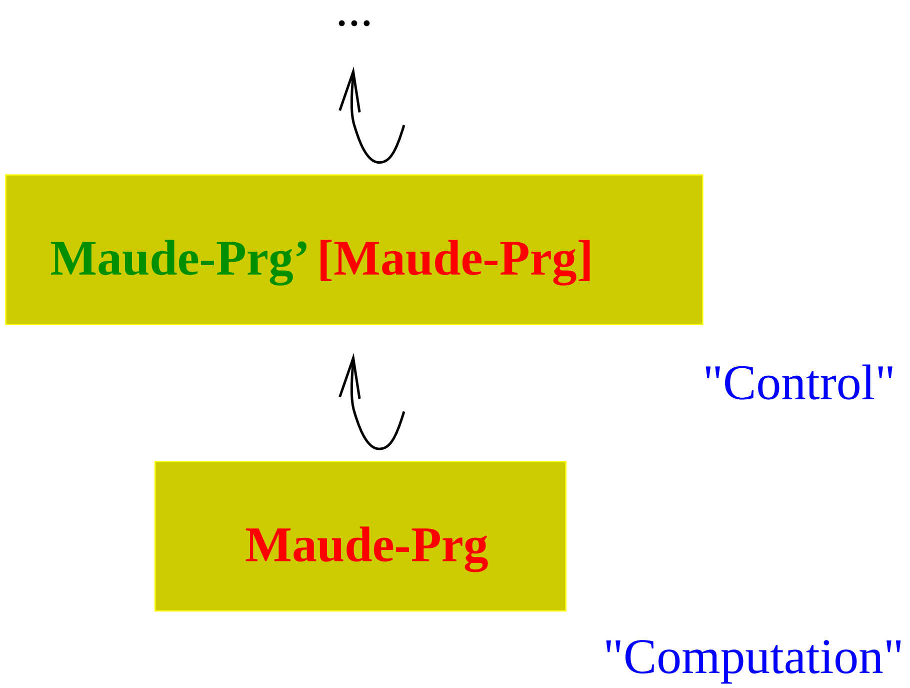
Analysis Techniques

Maude supports a wide spectrum of analysis capabilities:

- Executable specification and testing of specifications
- Model checking and search
- Symbolic simulation (narrowing)
- Theorem proving

How is this done?

Reflection



E.g., `Maude-Prg'`
- manipulates execution order of rules in `Maude-Prg`
- defines strategies
- implements model-checker
- implements theorem-prover

Maude as a Meta-language

Maude provides flexible meta-language capabilities to represent the syntax of a given logic or language.

In particular, Maude has a *parser-generator* (`meta-parse`) that yields an efficient parser for any context-free grammar.

A context-free grammar can be specified within Maude as a *signature* of typed operations in a Maude module.

Similarly, Maude has a *pretty-printer generator* (`meta-pretty-print`) that, given a context-free grammar, generates a pretty-printer for it.