The Maude 2.0 System

M. Clavel F. Durán S. Eker P. Lincoln

N. Martí-Oliet J. Meseguer C. Talcott

Sort System

- *Kinds* replace the *Error Sorts* of Maude 1 and can be used in operator and variable declarations.

```
  op f : [Nat] -> Nat .   *** error recovery.
```

- Operators can be declared at the kind level to indicate partiality.

```
  op gamma : [Real] -> [Real] .
  op gamma : Real ~> Real .
```

- Rewriting can occur at kind level for error recovery.
- Canonical name for a kind is its maximal sort(s), separated by commas and enclosed in brackets.
- Union sorts abolished.

Variables

- Each sort or kind t is considered to have a countable set of variables $v : t$ for any Maude identifier v, which do not have to be declared.
- A variable declaration `var v : t .` is now considered to declare v as an *alias* for the variable $v : t$.

```
parse in BOOL : X:Bool .
red in BOOL : X:Bool == X:[Bool] .   *** false
```

Conditions

- Rather than a single equality, conditions now allow a list of *condition fragments*, separated by /\.

- 4 types of fragments; last type is for rule conditions only:

⟨term⟩ = ⟨term⟩	equality test
⟨term⟩ : ⟨sort⟩	sort test
⟨pattern⟩ := ⟨term⟩	assignment by matching
⟨term⟩ => ⟨pattern⟩	rewrite proof search

- Patterns may have unbound variables that are bound by matching; regular terms are reduced upto strategy.

- Failure of a fragment causes backtracking.

Statement Attributes

- Statements can now take a list of attributes.
- Label attribute can be given to any statement, not just rules.
- Can be used with *trace select* feature to select statements for tracing.
- Metadata attribute can attach arbitrary string of meta-data to a statement for meta-processing.

```
eq X * X - Y * Y = (X + Y) * (X - Y)
  [label diff-sqrs metadata "lemma"] .
```

Ctor and Ditto Attributes

- Ctor attribute allows operators to be declared as constructors — needed for ITP and used in term coloring.

- Ditto copies all attributes other than ctor from a previous subsort overloaded declaration of the operator.

```
fmod CTOR-&-DITTO-TEST is
  sorts NzNat Nat Int .
  subsorts NzNat < Nat < Int .
  op 0 : -> Nat [ctor] .
  op s_ : Nat -> NzNat [ctor prec 14] .
  op s_ : Int -> Int [ditto] .   *** non-ctor
  op -_ : NzNat -> Int [ctor prec 14] .
  op -_ : Int -> Int [ditto] .   *** non-ctor
endfm
```

Iter Attribute

- Allows efficient storage, i/o and sort computations for huge towers of unary operator symbols.
- Main application is the efficient implementation of natural numbers using the successor notation.

```
fmod ITER-TEST is
  sorts Even Odd Nat .
  subsorts Even Odd < Nat .
  op 0 : -> Even .
  op s_ : Even -> Odd [iter] .
  op s_ : Odd -> Even [iter] .
endfm

red s_^123456789(0) .
red s_^1234567890(0) .
```

Format Attribute

- Allows control of white-space, color and style for pretty-printing operators.
- Format words are given for each white-space position.

s	space	r	red
t	tab	g	green
+	increment indent counter	b	blue
-	decrement indent counter	y	yellow
i	indent by indent counter	m	magenta
n	new line	c	cyan
d	default spacing	u	underline
o	original style	!	bright

Format Attribute (continued)

```
op while _ do _ od : Bool Statement -> Statement .
  ^     ^ ^  ^ ^  ^
```

```
op let _ := _ : Variable Expression -> Statement .
  ^   ^ ^  ^ ^
```

```
op while _ do _ od : Bool Statement -> Statement
  [format (nir! o r! o++ --nir! o)] .
```

```
op let _ := _ : Variable Expression -> Statement
  [format (nir! o d d d)] .
```

Natural Numbers

- Nats are constructed using successor operator and the *iter* attribute.

```
fmod NAT is
  sorts Zero NzNat Nat .
  subsort Zero NzNat < Nat .
  op 0 : -> Zero [ctor] .
  op s_ : Nat -> NzNat
    [ctor iter special (...)] .
  ...
endfm
```

- Decimal i/o by default.
- Built-in operators very efficient (use GNU GMP).
- Gcd, lcm, mod exp, bitwise ops.

Natural Numbers (continued)

```
red in NAT : gcd(18, gcd(X:Nat, 12)) .

fmod COMBINATORIAL is protecting NAT .
  op _! : Nat -> NzNat .
  op C : Nat Nat -> Nat .

vars N M : Nat .
  eq 0 ! = s 0 .
  eq (s N) ! = s N * N ! .
  eq C(N, M) = N ! quo (M ! * sd(N, M) !) .
endfm

red 1000 ! .
red C(1000, 100) .
```

Integers

- Ints are constructed from Nats using unary minus operator.

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

- Adds binary minus, not and abs.
- Most Nat ops extended.

Rational Numbers

- Rats are constructed from Nats and Ints using division operator.

```
fmod RAT is protecting INT .
  sorts NzRat Rat .
  subsorts NzInt < NzRat Int < Rat .
  op _/_ : NzInt NzNat -> NzRat
    [ctor prec 31 gather (E e) special (...)] .
  op _/_ : NzRat NzRat -> NzRat [ditto] .
  op _/_ : Rat NzRat -> Rat [ditto] .
  ...
endfm
```

- Supports usual arithmetic, comparison, abs, ceiling, floor, frac, trunc.
- Implemented by Maude equations rather than by built-ins.

Floating Point Numbers

- Floats are treated as a large set of constants - no algebraic structure.
- Underlying representation is hardware double precision floating point numbers; IEEE 754 on recent hardware.
- +/- Infinity supported.
- All usual floating points ops supported; known inconsistencies at corner cases patched.

Character Strings

- Strings are treated as a countable set of constants - no algebraic structure.
- Uses Nats for length, positions.
- Underlying representation is the C++ STL extension *ropes*.
- Heavyweight strings that support sharing and persistence efficiently for functional languages.
- Supports concatenation, length, comparison, substrings, find/rfind.

Conversion Functions

- FiniteFloat to Rat is exact.
- Rat to Float is nearest available Float; may be +/- Infinity.
- Rat to and from String is exact; choice of representation base.
- String to Float is nearest available Float; may be +/- Infinity.
- Float to String has enough significant digits that converting back finds the original Float.
- Exact decimal expansion of a Float is available for fancier i/o processing - may be huge.

LTL Model Checker

- Linear Temporal Logic manipulations (simplifications and negative normal form) are done by Maude code.
- The satisfaction of (possibly parameterized) propositions are defined in Maude.
- On-the-fly model checking is done by a built-in operation

```
  op _|=_ : State Formula ~> ModelCheckResult
    [special (...)] .
```

- Implementation uses state-of-the-art Buchi automaton construction algorithm and standard double depth first search of the synchronous product for a counterexample.
- LTL Satisfiability solving and tautology checking also provided.

New Meta Level

- Simpler metaterm representation:

  ```
  1.0 + X:Float
  ```

  is meta-represented as

  ```
  '_+_['1.0.FiniteFloat, 'X:Float]
  ```

- Many more descent functions such as metaXapply() (with contexts) and metaSearch().
- Descent functions return sorts of terms.
- Ascent functions to inspect modules in database.
- Much more sophisticated caching at module and operator level.

Search Command

- Breadth first search with cycle detection.
- All nodes in search graph reduce w.r.t. equations.
- Optional backtracking to find more solutions.
- Path to a given solution and current search graph can be printed.

```
search <term> <search-type> <pattern>
  such that <condition> .
```

=>	exactly one rewrite
=> *	zero or more rewrites
=> +	one or more rewrites
=>!	until no more rules apply

Profiling

- Keep track of how many times each eq/mb/rl is applied.
- For a condition, also keep track for each fragment of
  - Initial attempts.
  - Backtrack attempts.
  - Successes.
  - Failures.

```
set profile on .
...
show profile .
```

Term Coloring

- Color (possibly intermediate) results based on reduced flag and constructor status to flag problems.

```
set print color on .
```

reduced, ctor	not colored
reduced, non-ctor, colored below	blue
reduced, non-ctor, no colored below	red
unreduced, no reduced above	green
unreduced, reduced directly above	magenta
unreduced, reduced not directly above	cyan

- Red and magenta denote likely origin of problem, blue and cyan denote secondary damage.

Miscellaneous Features

- Frozen attribute prevents selected arguments of an operator from being rewritten by rules.

- Position fair rewriting make bottom-up passes over the term, applying a certain number of rule rewrites at each position. Since rule rewriting is non-destructive, term graph is *virtual* unless forced by tracing or debugging.

- Break to debugger when select operators rewrite or statements with given labels execute.

- Integrated compiler for sublanguage — GNU g++ used as backend. Speed up is typically a factor of 5-9.

Optimizations

- Rules can now use *greedy matching* under some circumstances.
- Left-to-right sharing allows reuse of matched subterms in rhs or condition. Matched subterms themselves can be in earlier condition fragments.
- New discrimination nets allow order-sorted partial subsumption analysis.
- Substitution slot coloring used to minimize size of substitutions by slot reuse - a win for large rhs.