Rewriting Logic

Specification and Verification of Programs

Fredrik Rømming

IN5110/9110, Fall 2021

UNIVERSITY
OF OSLO

Overview

- For deeper insight: IN2100, IN5100/9100 (Peter Ölveczky)
- Rewriting logic
  - Can naturally express both non-deterministic computation and logical deduction with great generality.
- Maude: language and system for rewriting logic

Theoretical Computer Science 96 (1992) 73–155
Elsevier
73

Conditional rewriting logic as a unified model of concurrency

José Meseguer*

SRI International, Menlo Park, CA 94025, USA, and Center for the Study of Language and Information, Stanford University, Stanford, CA 94305, USA

Equational Logic

Syntax: first-order terms + equality

Equational specification (Σ, E):

- Σ: Algebraic signature (functions, variables, constants)
- E: Set of equations of terms

Equational Logic

$E \vdash s = t$ if $(s = t) \in E$ or can be deduced by:

Reflexivity

$$t = t$$

Symmetry

$$\frac{t = t'}{t' = t}$$

Transitivity

$$\frac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3}$$

Congruence

$$\frac{t_i = t_i', i = 1 \dots n}{f\left(t_1, \dots, t_n\right) = f\left(t_1', \dots, t_n'\right)}$$

Substitutivity. For each substitution σ

$$\frac{t = t'}{\sigma(t) = \sigma\left(t'\right)}$$

Rewriting Logic

Equational Logic + Rewrite rules

Rewriting Logic specification $\mathcal{R} = (\Sigma, E, L, R)$:

- Σ, E: Equational specification
- L: Set of labels
- R: Set of rewrite rules $l : t \longrightarrow t'$, where $l \in L$, and t, t' are terms

Rewriting Logic

$\mathcal{R} \vdash s \longrightarrow t$ if $(s \longrightarrow t) \in R$ or can be deduced by:

Reflexivity

$$t \longrightarrow t$$

Equality

$$\frac{u \longrightarrow u'}{t \longrightarrow t'} \text{ if } E \vdash t = u \text{ and } E \vdash t' = u'$$

Transitivity

$$\frac{t_1 \longrightarrow t_2 \quad t_2 \longrightarrow t_3}{t_1 \longrightarrow t_3}$$

Congruence

$$\frac{t_i \longrightarrow t_i', i = 1 \ldots n}{f\,(t_1, \ldots, t_n) \longrightarrow f\,\left(t_1', \ldots, t_n'\right)}$$

Substitutivity. For each substitution σ

$$\frac{t \longrightarrow t'}{\sigma(t) \longrightarrow \sigma\,(t')}$$

Metalogical notes

$$t \overset{*}{\rightsquigarrow}_E u \quad \text{iff} \quad (\Sigma, \emptyset, \{l\}, \text{rules}(E)) \vdash t \longrightarrow u$$

where:

- $t \overset{*}{\rightsquigarrow}_E u$ means that t can be reduced to u by 0 or more applications of the equations in E to t.
- rules(E): transforms each equation $t_1 = t_2$ in E to a rewrite rule $l : t_1 \longrightarrow t_2$.

Heavily related to Universal algebra and Category theory

Why Rewriting Logic?

Software modules have algebraic structure

- Data form sets
- Operations on data $\simeq$ functions on sets

Intuitive formal system specification:

- Data types modeled by equational specifications
- Dynamic behaviors modeled by rewrite rules

Data Types

Elements	Functions
$\mathbb{N}$	$+, <, *, \ldots$
$\mathbb{Z}$	$+, -, \ldots$
lists of numbers	add, first, concat, remove element, sort, ...
stacks	pop, push, top, empty?, ..
multisets	add, remove, in?, ...
strings	substring, concat, ..
binary trees	size, inorder, preorder, isSearchTree, ..
graphs	hasCycle?, newEdge, ..
$\ldots$	$\ldots$

Maude syntax example

The data type $(\mathbb{N}, +)$:

```
fmod NAT-ADD is
  sort Nat .
  op 0 :  -> Nat [ctor] .
  op s :  Nat -> Nat [ctor] .
  op _+_ :  Nat Nat -> Nat .

  vars M N : Nat .

  eq 0 + M = M .
  eq s(M) + N = s(M + N) .
endfm
```

- elements (ground terms) defined by constructor functions
- other functions defined (recursively) by equations
- equations applied from left to right to simplify expressions
- equations must be (ground) confluent (Church-Rosser) and terminating
- Maude computes normal form of expressions

Reduction in Maude

Elements of sort `Nat` are `s(0)`, `s(s(0))`, `s(s(s(0)))`,...

1. Start Maude
2. Read file into Maude:

   ```
   Maude> in nat-add.maude
   ```

3. Execute Maude:

   ```
   Maude> red s(0) + s(0) .
   result Nat:  s(s(0))
   ```

4. End session with q (or quit)

Dynamic systems

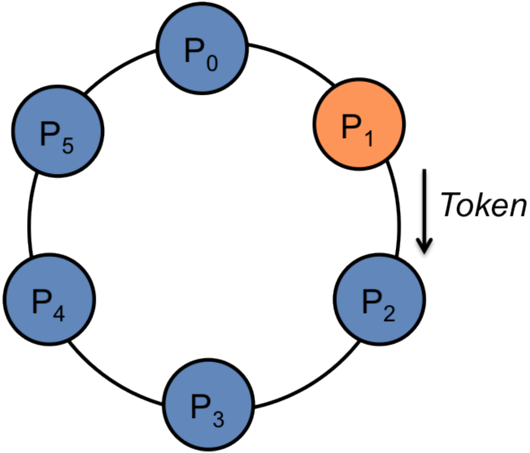
states that change

- States modeled by (E-equivalence classes of) terms
- State change modeled by labeled rewrite rules:

  rl [l]: t => t' .

  crl [l]: t => t' if cond .
- Dynamic systems may not be terminating or deterministic

Example: Token Ring distributed mutual exclusion



Maude rewriting example I

```
mod TOKEN-RING-MUTEX is
  sorts Name Node MutexState .
  op node:_state:_next:_ :  Name MutexState Name -> Node [ctor] .
  ops outsideCS waitCS insideCS : -> MutexState [ctor] .

  sorts Msg MsgContent .
  op msg_from_to_ :  MsgContent Name Name -> Msg [ctor] .
  op token :  -> MsgContent [ctor] .

  sort State .  subsort Node Msg < State .
  op none :  -> State [ctor] .
  op __ :  State State -> State [ctor assoc comm id:  none] .
```

Maude rewriting example II

```
vars N N2 N3 : Name .

rl [needCS] :
  node: N state: outsideCS next: N2
=>
  node: N state: waitCS next: N2 .

rl [receiveAndPassOnToken] :
  (msg token from N3 to N)
  node: N state: outsideCS next: N2
=>
  (node: N state: outsideCS next: N2)
  (msg token from N to N2) .
```

Maude rewriting example III

```
  rl [receiveAndKeepToken] :
    (msg token from N3 to N)
    node:  N state:  waitCS next:  N2
  =>
    node:  N state:  insideCS next:  N2 .

  rl [exitCS] :
    node:  N state:  insideCS next:  N2
  =>
    (node:  N state:  outsideCS next:  N2)
    (msg token from N to N2) .
endm
```

Maude rewriting example IV

```
mod TEST-MUTEX is including TOKEN-RING-MUTEX .
  ops a b c d e :  -> Name [ctor] .

  op init :  -> State .
  eq init =
    (msg token from d to a)
    (node:  a state:  outsideCS next:  b)
    (node:  b state:  outsideCS next:  c)
    (node:  c state:  outsideCS next:  d)
    (node:  d state:  outsideCS next:  e)
    (node:  e state:  outsideCS next:  a) .
endm
```

Simulation

```
Maude> frew [30] init .

result (sort not calculated):
  (node:  a state:  waitCS next:  b)
  (node:  b state:  waitCS next:  c)
  (node:  c state:  waitCS next:  d)
  (node:  d state:  insideCS next:  e)
   node:  e state:  waitCS next:  a
```

Search

=>* (reachable in 0 or more steps)

=>! ("final/deadlocked state

```
Maude> search [1] init =>*
      REST:State
      (node:  N:Name state:  insideCS next:  N2:Name)
      (node:  N3:Name state:  insideCS next:  N4:Name) .

No solution.

Maude> search [1] init =>!  STATE:State .

No solution.
```

Beyond reachability

X must happen in all behaviors (from initial state), e.g.:

- Node n must have executed in CS

More complex path behaviors, e.g.:

- Each node must execute in CS infinitely often
- Fairness: a node cannot execute forever outside CS without entering the wait state

Beyond reachability

X must happen in all behaviors (from initial state), e.g.:

- Node n must have executed in CS

More complex path behaviors, e.g.:

- Each node must execute in CS infinitely often
- Fairness: a node cannot execute forever outside CS without entering the wait state

Transition systems $\simeq$ abstract rewriting systems

Rewrite Theory $\longleftrightarrow$ Kripke Structure

Definition (Kripke Structure) Given a set AP of atomic propositions, a Kripke structure is a triple $(S, \rightarrow, L)$ s.t.:

- S is a set (of states)
- $\rightarrow \subseteq S \times S$ is a left-total binary relation (the transition relation)
- L is a labeling function $L : S \rightarrow 2^{AP}$ assigning to each state the atomic propositions holding in that state.

Rewrite Theory $\longleftrightarrow$ Kripke Structure

Definition (Kripke Structure) Given a set AP of atomic propositions, a Kripke structure is a triple $(S, \rightarrow, L)$ s.t.:

- S is a set (of states)
- $\rightarrow \subseteq S \times S$ is a left-total binary relation (the transition relation)
- L is a labeling function $L : S \rightarrow 2^{AP}$ assigning to each state the atomic propositions holding in that state.

A rewrite theory $\mathcal{R} = (\Sigma, E, R)$ with designated state sort `State` and labeling function L defines a Kripke structure $\left(T_{\Sigma, E_{\texttt{State}}}, \longrightarrow^{\bullet}, L\right)$, where:

- $T_{\Sigma, E_{\texttt{State}}}$ is the set of (E-equivalence classes of) ground terms of sort State
- $\longrightarrow^{\bullet}$ is the one-step sequential rewrite relation on the states extended with transitions `t` $\longrightarrow^{\bullet}$ `t` for deadlocked states
- L is the labeling function (in Maude: `op _|=_ : State -> Prop`)

LTL Model checking in Maude

- `is including MODEL-CHECKER`
- `sort State`
  - States must have sort `State`
- `sort Prop`
  - (Parametric) atomic propositions are terms of sort `Prop`
- `op _|=_ :  State -> Prop`
  - Define |= so `t |= p` is true when `p` holds in state `t`
  - No need to define false cases
- `sort Formula`
  - LTL formula is a term of sort `Formula`
  - Atomic propositions
  - Booleans `True`, `False`, `~`, `/\`, `\/`, `->`, ...
  - Temporal operators `[]`, `<>`, `O`, `U`

Model checking example I

```
load model-checker

mod MODEL-CHECK-MUTEX is including MODEL-CHECKER .
  protecting TEST-MUTEX .
  ops exInCS outside waiting :  Name -> Prop [ctor] .
  var MS : MutexState .  var REST : State .  vars N N2 :  Name .

  eq REST (node:  N state:  insideCS next:  N2) |= exInCS(N) = true .
  eq REST (node:  N state:  outsideCS next:  N2) |= outside(N) = true .
  eq REST (node:  N state:  MS next:  N2) |= waiting(N) = MS == waitCS .

  op fair :  Name -> Formula .
  eq fair(N) = (<> [] outside(N)) -> ([] <> waiting(N)) .
  op allFair :  -> Formula .
  eq allFair = fair(a) /\ fair(b) /\ fair(c) /\ fair(d) /\ fair(e) .
endm
```

Maude LTL example

```
Maude> red modelCheck(initState, formula) .
```

Examples:

```
Maude> red modelCheck(init, <> exInCS(b)) .

result ModelCheckResult:  counterexample(...)

Maude> red modelCheck(init,
       allFair -> (([] <> exInCS(c)) /\ ([] <> exInCS(e)))) .

result Bool:  true
```

Benefits of this approach to LTL

Consequences of elegant syntax and expressive formalization:

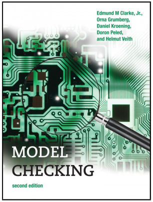
- Can define temporal logic formulas recursively
- State space collapsed by equations, equational equivalence classes

Maude LTL implementation overview

`model-checker.maude`

Maude Manual:

- "Maude uses an on-the-fly LTL model-checking procedure of the style described in [24]... reduce the satisfaction problem to the emptiness problem of the language accepted by the synchronous product of two Büchi automata..."
- "For efficiency purposes we need to make $B_{\neg\phi}$ as small as possible,



Beyond LTL: TLR and LTLR

Sometimes hard to express desired properties using only a state-based logic.

E.g. fairness requirements combine state-based properties (the enabledness of an action) with action-based properties (an action is "taken").

Action pattern: a rule label l with a partial substitution σ of the variables in the rule, and optionally a context ("position" or "part of the state").

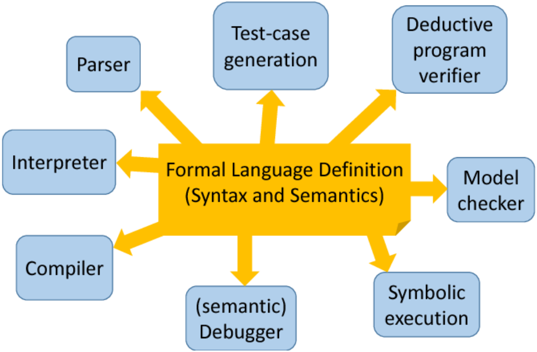
Temporal Logic of Rewriting (TLR): extends state-based atomic propositions with action patterns.

Linear Temporal Logic of Rewriting (LTLR): LTL where the atomic propositions can be both state propositions and action patterns.

$\quad \Diamond\Box$ "message m from o is in the state"
$\rightarrow \Box\Diamond$ ("apply rule l_1 with $\circ \mapsto o$" $\vee \ldots \vee$ "apply rule l_k with $\circ \mapsto o$ ").

Beyond Maude: $\mathbb{K}$

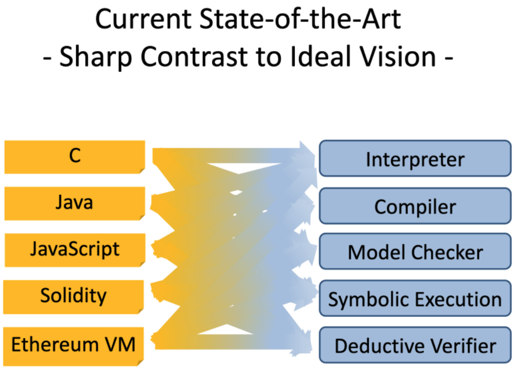
$\mathbb{K}$: Framework for defining PL syntax and semantics as Rewrite theory to automatically generate PL tools.



(Taken from Grigore Rosu's slides for invited talk at ICTAC'21)

Beyond Maude: $\mathbb{K}$

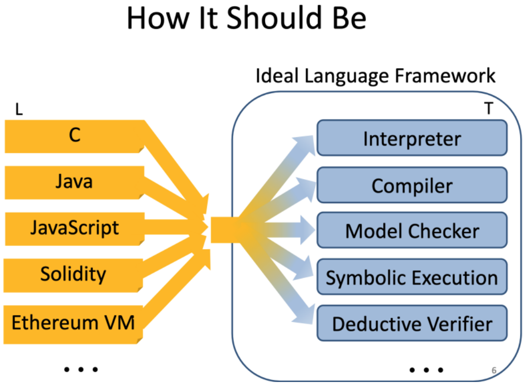
$\mathbb{K}$: Framework for defining PL syntax and semantics as Rewrite theory to automatically generate PL tools.



(Taken from Grigore Rosu's slides for invited talk at ICTAC'21)

Beyond Maude: $\mathbb{K}$

$\mathbb{K}$: Framework for defining PL syntax and semantics as Rewrite theory to automatically generate PL tools.



(Taken from Grigore Rosu's slides for invited talk at ICTAC'21)

Beyond Maude: $\mathbb{K}$

$\mathbb{K}$: Framework for defining PL syntax and semantics as Rewrite theory to automatically generate PL tools.

K Scales

Several large languages were recently defined in K:

- JavaScript ES5: by Park etal [PLDI'15]
  - Passes existing conformance test suite (2872 programs)
  - Found (confirmed) bugs in Chrome, IE, Firefox, Safari
- Java 1.4: by Bogdanas etal [POPL'15]
- x86: by Dasgupta etal [PLDI'19]
- C11: Ellison etal [POPL'12, PLDI'15]
  - 192 different types of undefined behavior
  - 10,000+ program tests (gcc torture tests, obfuscated C, ...)
  - Commercialized by startup (Runtime Verification, Inc.)

\+ EVM[CSF'18], Solidity, IELE[FM'19], WASM, Michelson,

(Taken from Grigore Rosu's slides for invited talk at ICTAC'21)

Beyond Maude: $\mathbb{K}$

https://kframework.org/

Conclusion

Rewriting Logic:

- Intuitive and expressive language to model distributed systems
- State space collapsed by equations (equational equivalence classes)
- Easy to define complex temporal logic formulae (recursively), including parametrized formulae
- Introduces TLR and LTLR state and action base temporal logic.
- Expressivity enables K (Turing completeness)

More?

- IN2100 and IN5100/9100
- Maude Manual:

  `http://maude.lcc.uma.es/maude31-manual-html/maude-manual.html`
- Webpage:

  `http://maude.cs.illinois.edu/w/index.php/The_Maude_System`
- Our upcoming shiny new paper in LNCS