0. Rewriting Logic Overview

Camilo Rocha
Escuela Colombiana de Ingeniería

7th International School on Rewriting (ISR2014)
Universidad Técnica Federico Santa María
August 25-29, 2014

Scope

Rewriting logic is a flexible framefork for specifying **concurrent systems** and **proof systems**

semantic framework: concurrent systems and other languages (and their semantics) can be specified in rewriting logic

logical framework: other logics can be specified in rewriting logic

Reachability in this series of talks focuses on rewriting logic as a semantic framework.

Rewrite Theories

A **rewrite theory** $\mathcal{R} = (\Sigma, E, R)$ is the unit of specification

- Σ is a signature defining the **syntax** of the system and of its states
- E is a set of equations defining the system's states as an **algebraic data type**
- R is a set of *rewrite rules* of the form $t \longrightarrow t'$ specifying the system's **local concurrent transitions**

Deduction and Semantics

Given a rewrite theory $\mathcal{R} = (\Sigma, E, R)$, deduction happens at two levels, yielding two different semantics:

- equational deduction with (Σ, E)
  - represented with the congruence $=_E$ on ground terms in T_Σ

$$t =_E u \qquad \text{iff} \qquad (\Sigma, E) \vdash t = u$$

  - intended semantics $\mathcal{T}_{\Sigma/E}$, i.e., the initial model in the class of algebras $\mathbf{Alg}(\Sigma, E)$
- rewrite deduction with R on $\mathcal{T}_{\Sigma/E}$
  - represented with the reachability relation $\rightarrow_{\mathcal{R}}$ on $\mathcal{T}_{\Sigma/E}$

$$t \rightarrow_{\mathcal{R}} t' \qquad \text{iff} \qquad (\Sigma, E, R) \vdash t \rightarrow t'$$

  - intended semantics $\mathcal{T}_{\mathcal{R}}$, i.e., the initial reachability model in the class of reachable models $\mathbf{Mod}(\Sigma, E, R)$

Operational Semantics

Under some conditions, a rewrite theory can be *executed* in rewrite engines such as CafeOBJ, Elan, and Maude:

- the set of equations E can be decomposed into $G \cup B$, with G a set of oriented equations that are **convergent** modulo the structural axioms B (such as A, C, and U axioms)
  - the equational relation $=_E$ can then be implemented by well-founded rewriting by using a "don't" care strategy (reduce at will)
- the set of rewrite rules R is **coherent** w.r.t. G modulo B
  - the rewrite relation $\rightarrow_{\mathcal{R}}$ can then be implemented under the strategy of reducing a state to its canonical form and then applying the enabled rewrite rules (without losing completeness)

Example: the QLOCK protocol

QLOCK is a toy example of a protocol for regulating access of processes to a shared resource; it uses a global queue as follows:

- if a process wants to use the shared resource and its name is not on the queue, then it register its name in the queue
- if a process wants to use the shared resource and its name is the first one on the queue, then this process is granted access to the resource; otherwise, it waits
- if a process finishes using the shared resource, then it removes its name from the queue

QLOCK: a process

A process is specified as a natural number in the equational theory

$$\mathcal{E}_{\mathtt{Nat}} = (\Sigma_{\mathtt{Nat}}, E_{\mathtt{Nat}}),$$

with $E_{\mathtt{Nat}} = \emptyset$ and $\Sigma_{\mathtt{Nat}}$ defined as follows:

```
sort Nat .
op 0 : -> Nat [ctor] .
op s : Nat -> Nat [ctor] .
```

Examples of terms in `Nat`: `0`, `s(0)`, `s(s(0))`, `s(s(s(0)))`, ...

Remark

Note that $\mathbb{N}$, $\mathbb{Q}$, and $\mathbb{R}$ are all algebras of $\mathcal{E}_{\mathtt{Nat}}$, but only $\mathcal{T}_{\Sigma_{\mathtt{Nat}}/E_{\mathtt{Nat}}} \simeq \mathbb{N}$.

QLOCK: the global queue

The global queue is defined as term in the equational theory

$$\mathcal{E}_{\texttt{Queue}} = (\Sigma_{\texttt{Queue}}, E_{\texttt{Queue}})$$

as follows:

```
sort Queue .
op nil : -> Queue [ctor] .
op _@_ : Nat Queue -> Queue [ctor] .
op _;_ : Queue Queue -> Queue .
eq nil ; L2 = L2 .
eq (N @ L1) ; L2 = N @ (L1 ; L2) .
```

QLOCK: collections of processes

A collection of processes is defined as a term in the equational theory

$$\mathcal{E}_{\texttt{Soup}} = (\Sigma_{\texttt{Soup}}, E_{\texttt{Soup}})$$

as follows:

```
sort Soup .
subsort Nat < Soup .
op mt : -> Soup [ctor] .
op __ : Soup Soup -> Soup [ctor assoc comm id: mt] .
```

QLOCK: the global state

A state of the system is defined as a term in the equational theory

$$\mathcal{E}_{\texttt{State}} = (\Sigma_{\texttt{State}}, E_{\texttt{State}})$$

as follows:

```
sort State .
op _|_|_|_ : Soup Soup Soup Queue -> State [ctor] .
```

QLOCK: concurrent transitions

The concurrent transitions of the protocol are defined as rewrite rules in the theory

$$\mathcal{R}_{\texttt{QLOCK}} = (\Sigma_{\texttt{State}}, E_{\texttt{State}}, R_{\texttt{QLOCK}})$$

with $R_{\texttt{QLOCK}}$ defined as follows:

```
vars N N' : Nat .   var  Q : Queue .   vars Si Sw Sc : Soup .

rl [to-waiting] :
   Si N | Sw | Sc | Q  => Si | Sw N | Sc | Q ; (N @ nil ) .

rl [to-critical] :
   Si | Sw N | Sc | N @ Q   => Si | Sw | Sc N | N @ Q .

rl [to-inactive] :
   Si | Sw | Sc N | N' @ Q  => Si N | Sw | Sc | Q .
```

Execution and Formal Analysis

The rewrite theory $\mathcal{R}_{\texttt{QLOCK}}$ is convergent and coherent, then:

- the system can be executed from a given initial state
- the system can be model-checked using, for example, Maude's LTL model-checker or Maude's LTRL (linear temporal logic of rewriting) model checker
- properties of the system's states can be established by using Maude's Inductive Theorem Prover (ITP)

Execution and Formal Analysis

The rewrite theory $\mathcal{R}_{\texttt{QLOCK}}$ is convergent and coherent, then:

- the system can be executed from a given initial state
- the system can be model-checked using, for example, Maude's LTL model-checker or Maude's LTRL (linear temporal logic of rewriting) model checker
- properties of the system's states can be established by using Maude's Inductive Theorem Prover (ITP)

However,

- Can you prove a mutual exclusion property for `QLOCK` (e.g., at most one process has access to the shared resource at any particular moment of execution) ?