Formal Specification and Analysis of Concurrent Systems in Rewriting Logic

Kyungmin Bae

Korea Logic Day, 12 January 2024

Department of Computer Science and Engineering
POSTECH (Pohang University of Science and Technology)

Rewriting Logic

José Meseguer

Joseph Goguen

- Fragment of first-order logic with two relations: equality $=$ and evolution $\rightarrow$
- Equations $(\forall X)\ t = t'$ and rules $(\forall X)\ t \rightarrow t'$
- Axioms: reflexivity, equality, congruence, replacement, transitivity

Rewriting Logic

- Semantic framework
  - Concurrency models: actors, process calculi, Petri nets, ⋯
  - Programming languages: C, Java, JavaScript, Scheme, Haskell, ⋯
  - Modeling languages: Verilog, ABEL, AADL, Ptolemy II, PLEXIL, ⋯
- Modeling framework
  - Concurrent object-oriented systems, real-time embedded systems, ⋯
  - Scheduling protocols, network protocols, security protocols, ⋯
  - Hardware designs, systems biology, ⋯
- Formal analysis framework
  - Reachability analysis, model checking, theorem proving, ⋯
  - Real-time systems, probabilistic systems, hybrid systems, ⋯

History

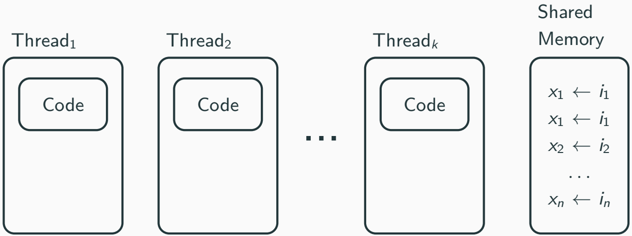
- Clear (Edinburgh) 1970s
- OBJ family (Stanford, Oxford) 1980s
- Maude (SRI, UIUC), ELAN (France), CafeOBJ (Japan) present

Rewriting Logic Specification: Informal Description

- State: algebraic data structures
  - recursive data types and functions
  - lists, sets, multi-sets, $\cdots$
- (Concurrent) transition: evolution of patterns
  - rewrite rule $t \to t'$
  - pattern t (concurrently) evolves to pattern t'

Example: Very Simple Parallel Language (2)

- Configuration



- Defined as logical term of the form:

$$\{[1, Code_1] \mid [2, Code_2] \mid \cdots \mid [k, Code_k], \quad [x_1, i_1]\ [x_2, i_2] \cdots [x_n, i, n]\}$$

Example: Very Simple Parallel Language (3)

- Semantics of program defined as rewrite rules R:

$$
\begin{aligned}
& \{[I, \mathit{skip} \,;\, P] \mid \mathit{THREADS}, \mathit{MEM}\} \\
\to\ & \{[I, P] \mid \mathit{THREADS}, \mathit{MEM}\}
\end{aligned}
$$

$$
\begin{aligned}
& \{[I, (V = E) \,;\, P] \mid \mathit{THREADS}, \mathit{MEM}\} \\
\to\ & \{[I, (V = E) \,;\, P] \mid \mathit{THREADS}, \mathit{update}(\mathit{MEM}, [V, \mathit{eval}(E)])\}
\end{aligned}
$$

$$
\begin{aligned}
& \{[I, \text{if } (T)\ \{P\} \,;\, P'] \mid \mathit{THREADS}, \mathit{MEM}\} \\
\to\ & \{[I, \left(\mathit{eval}(T)\ ?\ P\ :\ \mathit{skip}\right) \,;\, P'] \mid \mathit{THREADS}, \mathit{MEM}\}
\end{aligned}
$$

$$
\begin{aligned}
& \{[I, \text{while } (T)\ \{P\} \,;\, P'] \mid \mathit{THREADS}, \mathit{MEM}\} \\
\to\ & \{[I, \left(\mathit{eval}(T)\ ?\ (P \,;\, \text{while } (T)\ \{P\})\ :\ \mathit{skip}\right) \,;\, P'] \mid \mathit{THREADS}, \mathit{MEM}\}
\end{aligned}
$$

- Aux functions (*eval*, *update*, _?_ : _) are defined by equations E

Example: Very Simple Parallel Language (4)

- Dekker's algorithm
  - only one thread can enter its critical section.
  - hard to guarantee its correctness by testing

Thread 1

```
c1 = 1;
while (c2 == 1) {
  if (turn == 2) {
    c1 = 0;
    while (turn == 2) { /* busy wait */ }
    c1 = 1;
  }
}
... /* critical section */
turn = 2;
c1 = 0;
...
```

Thread 2

```
c2 = 1;
while (c1 == 1) {
  if (turn == 1) {
    c2 = 0;
    while (turn == 1) { /* busy wait */ }
    c2 = 1;
  }
}
... /* critical section */
turn = 1;
c2 = 0;
...
```

Example: Very Simple Parallel Language (5)

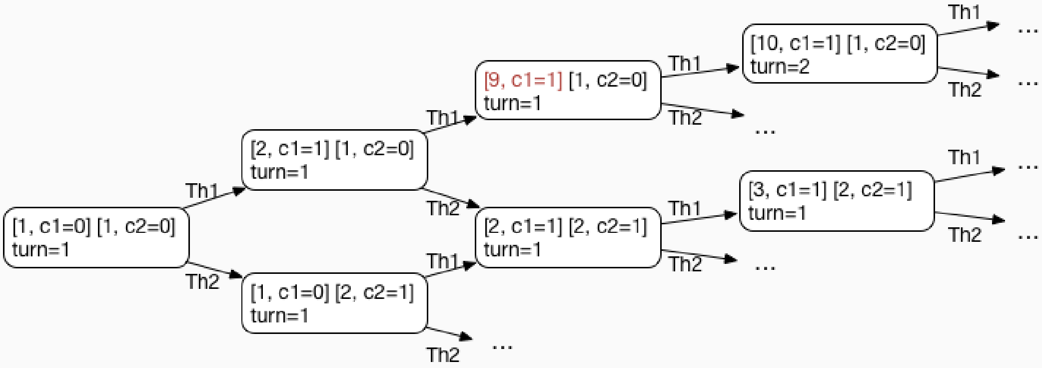
Dekker's Algorithm

```
{
 [1, c1 = 1; while (c2 == 1) { if (turn == 2) { c1 = 0; while (turn == 2) { skip } ;
    c1 = 1 } } ; crit ; turn = 2 ; c1 = 0]
|
 [2, c2 = 1; while (c1 == 1) { if (turn == 1) { c2 = 0; while (turn == 1) { skip } ;
    c2 = 1 } } ; crit ; turn = 1 ; c2 = 0]
 ,
 [c1,0] [c2,0] [turn,1]
}
```

- Can execute specification using rewrite rules
- Can enumerate all possible configurations by interleaving

Example: Very Simple Parallel Language (6)

- Graph by examining all possible interleaving



Rewrite Theories

- Concurrent system specifications in rewriting logic
- Rewrite theory $\mathcal{R} = (\Sigma, E, R)$:

  Σ: signature for logical terms $t \in T_\Sigma$

  E: equations defining equalities $t = t'$

  R: rewrite rules specifying labeled transitions $l : t \longrightarrow t'$

- $\mathcal{R}$ specifies a concurrent system
  - states: elements of algebraic data structure by equational theory (Σ, E)
  - concurrent transitions are specified by the rules R.

Rewriting Logic: Rules of Deduction

- Reflexivity

$$\frac{}{\mathcal{R} \vdash t \longrightarrow t}$$

- Transitivity

$$\frac{\mathcal{R} \vdash t_1 \longrightarrow t_2 \qquad \mathcal{R} \vdash t_2 \longrightarrow t_3}{\mathcal{R} \vdash t_1 \longrightarrow t_3}$$

- Equality

$$\frac{E \vdash u' = u \qquad \mathcal{R} \vdash u \longrightarrow v \qquad E \vdash v = v'}{\mathcal{R} \vdash u' \longrightarrow v'}$$

- Congruence

$$\frac{\mathcal{R} \vdash t_1 \longrightarrow t_1' \qquad \cdots \qquad \mathcal{R} \vdash t_n \longrightarrow t_n'}{\mathcal{R} \vdash f(t_1, \ldots, t_n) \longrightarrow f(t_1', \ldots, t_n')}$$

- Replacement

$$\frac{t(x_1, \ldots, x_n) \to u(x_1, \ldots, x_n) \in R \qquad \mathcal{R} \vdash p_1 \longrightarrow p_1' \quad \cdots \quad \mathcal{R} \vdash p_n \longrightarrow p_n'}{\mathcal{R} \vdash t[p_1/x_1, \ldots, p_n/x_n] \longrightarrow u[p_1'/x_1, \ldots, p_n'/x_n]}$$

Models of Rewrite Theories

Definition (Reachability Model)

A Σ-reachability model is a pair $\mathcal{A}_{\rightarrow} = (\mathcal{A}, \rightarrow^{\mathcal{A}})$, where

- $\mathcal{A}$ is a Σ-algebra, and
- $\rightarrow^{\mathcal{A}}$ is a reflexive, transitive, and congruence relation on $u(\mathcal{A})$.

- $\mathcal{A}_{\rightarrow}$ is an ordinary first-order structure with a binary relation $\rightarrow$.
- $\mathcal{A}_{\rightarrow} = (\mathcal{A}, \rightarrow^{\mathcal{A}})$ satisfies $\mathcal{R} = (\Sigma, E, R)$ iff: $\mathcal{A} \models E$ and $\overline{\alpha}(t) \rightarrow^{\mathcal{A}} \overline{\alpha}(t')$ for each rule $t \rightarrow t' \in R$ and assignment $\alpha : X \rightarrow u(\mathcal{A})$

Soundness and Completeness of Rewriting Logic

Theorem

Given a rewrite theory $\mathcal{R} = (\Sigma, E, R)$ *for any terms* $t, t' \in T_{\Sigma(X)}$*:*

$$\mathcal{R} \models t \longrightarrow t' \iff \mathcal{R} \vdash t \longrightarrow t'$$

Computability Conditions (1)

- Equations E (oriented) are ground confluent and terminating

$$\begin{tikzcd} t \arrow[r, "E^*"] \arrow[d, "E^*"'] & t_1 \arrow[d, dashed, "E^!"] \\ t_2 \arrow[r, dashed, "E^!"'] & t' \end{tikzcd}$$

- Any term has E-normal form
- Equality between two terms becomes decidable

Computability Conditions (2)

- Rules R are ground coherent with respect to E

$$\begin{tikzcd}
t \arrow[r, "R^1"] \arrow[dd, "E^!"'] & t' \arrow[dr, dashed, "E^!"] & \\
 & & w \\
u \arrow[r, dashed, "R^1"'] & u' \arrow[ur, dashed, "E^!"'] &
\end{tikzcd}$$

- Rule application of E-normal form is complete

- One-step rewrites become decidable

Maude

- Language and tool for rewriting logic
- Rewriting-based declarative programming language
- High-performance analysis tool
- Available at `http://maude.cs.illinois.edu`

Some Applications

- Distrubted systems, protocols, and algorithms
  - IETF multicast protocols, wireless sensor network algorithms, ⋯
  - Cloud transaction systems: Apache Cassandra, Google's Megastore, ZooKeeper, ⋯
- Programming languages
  - C, Java, JVM, Scheme, Ethereum smart contracts, ⋯
  - Verilog, NASA Plan Execution Language (Plexil), AADL, Ptolemy II, ⋯
- Security
  - Found address/status bar spoof attacks in Internet Explorer
  - Security protocol verificaion tools: Maude-NPA, Tamarin, ⋯
- Neuroscience, biological reactions (e.g., Pathway Logic at SRI), ⋯

Some Recent Progress

- More powerful formal analysis
  - A new inductive theorem prover (NuITP)
  - Infinite-state model checking
- Combination of rewriting logic with other formalisms
  - Satisfiability modulo theories (SMT)
  - Interactive theorem proving: Lean, Coq, $\cdots$
- New applications
  - Cyber-physical systems: discrete + continuous
  - Quantum-resistant security protocol

Thank you!