Rewriting Logic and Its Applications in Biology

Part 1: Rewriting Logic and Maude

Temur Kutsia

RISC, Johannes Kepler University of Linz, Austria
kutsia@risc.uni-linz.ac.at

May 29, 2008

What Are These Lectures About?

What Are These Lectures About?

- Describe (yet another) approach to modeling (certain) biological processes.

What Are These Lectures About?

- Describe (yet another) approach to modeling (certain) biological processes.
- The approach is called Pathway Logic.

What Are These Lectures About?

- Describe (yet another) approach to modeling (certain) biological processes.
- The approach is called Pathway Logic.
- The processes it models include signal transduction, metabolism, inter-cellular signalling, neuron systems.

What Are These Lectures About?

- Describe (yet another) approach to modeling (certain) biological processes.
- The approach is called Pathway Logic.
- The processes it models include signal transduction, metabolism, inter-cellular signalling, neuron systems.
- Pathway logic models are executable in the Maude programming language.

What Are These Lectures About?

- Describe (yet another) approach to modeling (certain) biological processes.
- The approach is called Pathway Logic.
- The processes it models include signal transduction, metabolism, inter-cellular signalling, neuron systems.
- Pathway logic models are executable in the Maude programming language.
- Maude is based on a simple but powerful logic called Rewriting Logic.

What Are These Lectures About?

- Describe (yet another) approach to modeling (certain) biological processes.
- The approach is called Pathway Logic.
- The processes it models include signal transduction, metabolism, inter-cellular signalling, neuron systems.
- Pathway logic models are executable in the Maude programming language.
- Maude is based on a simple but powerful logic called Rewriting Logic.
- Today: Rewriting Logic and Maude.

What Are These Lectures About?

- Describe (yet another) approach to modeling (certain) biological processes.
- The approach is called Pathway Logic.
- The processes it models include signal transduction, metabolism, inter-cellular signalling, neuron systems.
- Pathway logic models are executable in the Maude programming language.
- Maude is based on a simple but powerful logic called Rewriting Logic.
- Today: Rewriting Logic and Maude.
- Next week: Pathway Logic.

Contents

How Do We Rewrite?

Example

- Given the rewrite rules:

$$\begin{aligned} (1) \qquad & 0 + N \to N \\ (2) \qquad & s(M) + N \to s(M + N) \end{aligned}$$

- How do we rewrite $s(0) + s(s(0))$?

How Do We Rewrite?

Example

- Given the rewrite rules:

$$\begin{array}{ll} (1) & 0 + N \to N \\ (2) & s(M) + N \to s(M + N) \end{array}$$

- How do we rewrite $s(0) + s(s(0))$?

  One rewriting step:

$$s(0) + s(s(0)) \longrightarrow \text{(by the rule (2) with } M = 0, N = s(s(0))$$
$$s(0 + s(s(0))).$$

How Do We Rewrite?

Example

- Given the rewrite rules:

$$\begin{array}{ll} (1) & 0 + N \to N \\ (2) & s(M) + N \to s(M + N) \end{array}$$

- How do we rewrite $s(0) + s(s(0))$?
  Rewrite until impossible (reduce):

$$s(0) + s(s(0)) \longrightarrow \text{(by the rule (2) with } M = 0, N = s(s(0))$$

How Do We Rewrite?

Example

- Given the rewrite rules:

$$\begin{array}{ll} (1) & 0 + N \to N \\ (2) & s(M) + N \to s(M + N) \end{array}$$

- How do we rewrite $s(0) + s(s(0))$?

  Rewrite until impossible (reduce):

$$s(0) + s(s(0)) \longrightarrow \text{(by the rule (2) with } M = 0, N = s(s(0))$$
$$s(0 + s(s(0))) \longrightarrow \text{(by the rule (1) with } N = s(s(0))$$

How Do We Rewrite?

Example

- Given the rewrite rules:

$$\begin{aligned} (1) \qquad & 0 + N \to N \\ (2) \qquad & s(M) + N \to s(M + N) \end{aligned}$$

- How do we rewrite $s(0) + s(s(0))$?
  Rewrite until impossible (reduce):

$$\begin{aligned} & s(0) + s(s(0)) \longrightarrow \text{(by the rule (2) with } M = 0, N = s(s(0)) \\ & s(0 + s(s(0))) \longrightarrow \text{(by the rule (1) with } N = s(s(0)) \\ & \quad s(s(s(0))). \end{aligned}$$

Rewriting Logic

What is Rewriting Logic?

- A logic of actions whose models are concurrent systems.
- A logic for executable specification and analysis of software systems.
- A logic to specify other logics or languages.
- An extension of equational logic with local rewrite rules to express
  - concurrent change over time,
  - inference rules.

Rewriting Logic. Some Formal Notions

- Equational specification:
  - Syntax: signature, terms, equations
  - Semantics
- Matching, rewriting
- Rewriting logic (parametrized by an equational specification):
  - Syntax
  - Semantics
  - Inference system

Rewriting Logic. Equational Specification

Many-sorted signature: a pair (S, Σ) where

- S is a set of sorts.
- $\Sigma = \{\Sigma_{\overline{s},s} \mid \overline{s} \in S^*, s \in S\}$ is an $S^* \times S$-sorted family of function symbols.

Rewriting Logic. Equational specification

Example (Many-Sorted Signature)

Let

$$S = \{Nat, Bool\}$$
$$\Sigma = \{\Sigma_{Nat}, \Sigma_{Bool}, \Sigma_{Nat,Nat}, \Sigma_{Nat,Nat,Nat}, \Sigma_{Bool,Bool}, \Sigma_{Bool,Bool,Bool}, \Sigma_{Nat,Nat,Bool}\}$$

where

$$\begin{aligned} \Sigma_{Nat} &= \{0\} & \Sigma_{Bool} &= \{T, F\} \\ \Sigma_{Nat,Nat} &= \{s\} & \Sigma_{Nat,Nat,Nat} &= \{+\} \\ \Sigma_{Bool,Bool} &= \{\neg\} & \Sigma_{Bool,Bool,Bool} &= \{\vee\} \\ \Sigma_{Nat,Nat,Bool} &= \{\leq\} \end{aligned}$$

then (S, Σ) is a many-sorted signature.

Rewriting Logic. Equational specification

S-sorted family of terms

$$\mathcal{T}_\Sigma(X) = \{\mathcal{T}_{\Sigma,s}(X) \mid s \in S\}$$

over a many-sorted signature (S, Σ) and an S-sorted family $X = \{X_s \mid s \in S\}$ of (pairwise disjoint) sets of variables:

1. $X_s \subseteq \mathcal{T}_{\Sigma,s}(X)$ for each $s \in S$: variables are terms.
2. If $f \in \Sigma_{s_1,\ldots,s_n,s}$, $n \geq 0$ and $t_i \in \mathcal{T}_{\Sigma,s_i}(X)$ for each $1 \leq i \leq n$, then $f(t_1, \ldots, t_n) \in \mathcal{T}_{\Sigma,s}(X)$: A function symbol applied to terms of the appropriate sorts produces a new term.

Rewriting Logic. Equational specification

Example (Terms)

- (S, Σ) – many-sorted signature from the previous example.
- $X = \{X_{Nat}, X_{Bool}\}$ – family of Variables.
- $x \in X_{Nat}, A \in X_{Bool}$.

Some examples of S-sorted terms:

$$
\begin{array}{rcl}
x & \in & \mathcal{T}_{\Sigma,Nat}(X) \\
s(x) & \in & \mathcal{T}_{\Sigma,Nat}(X) \\
+(0, s(0)) & \in & \mathcal{T}_{\Sigma,Nat}(X) \\
\leq(s(0), x) & \in & \mathcal{T}_{\Sigma,Bool}(X)
\end{array}
\qquad
\begin{array}{rcl}
0 & \in & \mathcal{T}_{\Sigma,Nat}(X) \\
\neg F & \in & \mathcal{T}_{\Sigma,Bool}(X) \\
\vee(T, A) & \in & \mathcal{T}_{\Sigma,Bool}(X) \\
\vee(F, \leq(0, s(0)) & \in & \mathcal{T}_{\Sigma,Bool}(X)
\end{array}
$$

Often infix notation is preferred: $F \vee 0 \leq s(0)$ instead of $\vee(F, \leq(0, s(0))$.

Rewriting Logic. Equational specification

- Notation: $x : s$ means that x is a variable of the sort s.
- Σ-equation:

$$(x_1 : s_1, \ldots x_n : s_n)\ l = r,$$

  where $l, r \in \mathcal{T}_{\Sigma,s}(\{x_1 : s_1, \ldots x_n : s_n\})$.
- Conditional Σ-equation:

$$(x_1 : s_1, \ldots x_n : s_n)\ l = r \text{ if } u_1 = v_1, \ldots, u_m = v_m$$

  where $(x_1 : s_1, \ldots x_n : s_n)\ l = r$, $(x_1 : s_1, \ldots x_n : s_n)\ u_i = v_i$ are Σ-equations.
- Many-sorted specification: (S, Σ, E), where E is a set of conditional Σ-equations.

Rewriting Logic. Equational Specification

- Semantics of many-sorted specification is given by algebras.
- A many-sorted (S, Σ)-algebra consists of a carrier set A_s for each $s \in S$ and a function $F_f^{\overline{s},s} : A_{\overline{s}} \longrightarrow A_s$ for each $f \in \Sigma_{\overline{s},s}$.
- A meaning of a term and satisfaction of a (conditional) equation in an algebra can be defined by induction.
- The semantics of a many-sorted specification (S, Σ, E) is the set of (S, Σ)-algebras that satisfy all (conditional) equations in E.

Rewriting Logic. Equational Specification

- The semantics can be used to answer concrete questions, e.g. whether two terms have the same meaning.
- However, it is more convenient to use syntactic means to answer such questions. It would also provide more opportunities for efficient mechanization.
- Under certain conditions equational deduction can be mechanized by matching and rewriting.
- For rewriting, the equations are oriented from left to right.

Rewriting Logic. Matching and rewriting

Towards matching and rewriting:

- Substitution: A sort-preserving map $\sigma : X \longrightarrow \mathcal{T}_\Sigma(Y)$, where X and Y are S-sorted families of variables for (S, Σ). Substitutions can be uniquely extended to homomorphisms over terms.
- A term t matches a term r with a substitution σ if $\sigma(t) \equiv r$ (syntactically equal).

Rewriting Logic. Matching and Rewriting

Rewriting with unconditional equations:

- Requirement on oriented equations: All variables in the right hand side also appear in the left hand side.
- Under this assumption, a term t rewrites to a term t' using such an equation $(\cdots)\ l = r$ if
  - there is a subterm q in t such that $q \equiv \sigma(l)$,
  - t' is obtained from t by replacing q with the term $\sigma(r)$

Example (Rewriting)

A term $(0 + s(0)) + y$ rewrites to $s(0) + y$ by the equation $(x : Nat)\ 0 + x = x$.

Rewriting Logic. Matching and Rewriting

Confluence and termination:

- A set of equations is confluent if the result of rewriting a term is unique: For all t, t_1, t_2 if $t \rightarrow_E^* t_1$ and $t \rightarrow_E^* t_2$, then there exists a term t' such that $t_1 \rightarrow_E^* t'$ and $t_2 \rightarrow_E^* t'$.
- A set of equations E is terminating if there is no infinite sequence of rewriting steps $t_0 \rightarrow_E t_1 \rightarrow_E t_2 \cdots$.
- If E is confluent and terminating, any term t can be reduced to a unique normal form $t\downarrow_E$.
- Efficient mechanization: To check semantic equality of two terms, it is enough to check equality between their respective normal forms.

Example (Confluence and Termination)

$\{(x : Nat)\ 0 + x = 0,\ (x : Nat, y : Nat)\ s(x) + y = s(x + y)\}$ is confluent and terminating (left-to-right rewriting).

Rewriting Logic. Matching and Rewriting

Rewriting with conditional equations:

- Requirement on oriented equations: All variables in the right hand side and in the condition also appear in the left hand side.
- Under this assumption and confluence and termination of E a term t rewrites to a term t' using such an equation $(\cdots)\ l = r\ \textit{if}\ u_1 = v_1, \ldots, u_n = v_n$ in E if
  - there is a subterm q in t such that $q \equiv \sigma(l)$,
  - $\sigma(u_i) \downarrow_E \equiv \sigma(v_i) \downarrow_E$ for all $1 \leq i \leq n$,
  - t' is obtained from t by replacing q with the term $\sigma(r)$.

Rewriting Logic. Equational Specification

Order-sorted signature:

- Obtained from a many-sorted signature by adds a partial ordering $\leq$ to the set of sorts.
- $s_1 \leq s_2$ is interpreted by the subset inclusion $A_{s_1} \subseteq A_{s_2}$ between the corresponding carrier sets.
- Operations can be overloaded.
- Certain restrictions are introduced to guarantee that each term has the least sort and that equational deduction behaves well.
- Oriented equations should be sort-decreasing.
- Subsorts help to avoid partial functions.

Rewriting Logic. Equational Specification

Example (Ordered Sorts)

The successor function on natural numbers can be used to construct nonzero natural numbers.

Subsorts help to avoid partial functions.

We can define a subsort $NzNat < Nat$, introduce $0 \in \Sigma_{Nat}$, $s \in \Sigma_{Nat,NzNat}$, $div \in \Sigma_{Nat,NzNat,Nat}$ and two equations for it:

$$(x : Nat, y : NzNat)\ x\ div\ y = 0\ \textit{if}\ y > x$$
$$(x : Nat, y : NzNat)\ x\ div\ y = s((x - y)\ div\ y)\ \textit{if}\ y \leq x$$

where $>, \leq, -$ are defined elsewhere.

Rewriting Logic

Syntax of Rewriting Logic:

- Signature: an equational specification (Ω, E). RWL is parametrized by the choice its underlying equational logic. For instance, it can be many-sorted or order-sorted equational specification (S, Σ, E), or a more expressive membership equational logic (K, S, Σ, E).
- The signature of RWL makes explicit E in order to emphasize that rewriting will operate on congruence classes modulo E.
- Sentences of RWL are sequents (called rewrites)

$$[t]_E \longrightarrow [t']_E,$$

where t and t' are terms and $[t]_E$, $[t']_E$ are the corresponding congruence classes modulo E.

Rewriting Logic

Syntax of Rewriting Logic:

- A RWL specification $\mathcal{R}$ is a tuple $\mathcal{R} = (\Omega, E, L, R)$ where (Ω, E) is a signature, L is a set of labels, and R is a set of labeled rewrite rules:

$$r : [t]_E \longrightarrow [t']_E \text{ if } [u_1]_E \longrightarrow [v_1']_E \wedge \cdots \wedge [u_1]_E \longrightarrow [v_1']_E,$$

  where t and t' are terms and $[t]_E$, $[t']_E$, etc. are the corresponding congruence classes of terms in $\mathcal{T}_{\Omega,E}(X)$ modulo E.

Rewriting Logic

Inference rules of Rewriting Logic (E is omitted from congruence classes for simplicity):

1. Reflexivity. For each $[t] \in \mathcal{T}_{\Sigma,E}(X)$,

$$\frac{}{[t] \longrightarrow [t]}$$

2. Congruence. For each $f \in \Sigma_n$,

$$\frac{[t_1] \longrightarrow [t'_1] \cdots [t_n] \longrightarrow [t'_n]}{[f(t_1, \ldots, t_n)] \longrightarrow [f(t'_1, \ldots, t'_n)]}$$

Rewriting Logic

Inference rules of Rewriting Logic:

3. Replacement. For each rewrite rule

$$r : [t(\overline{x})] \longrightarrow [t'(\overline{x})] \; \textit{if} \; [u_1(\overline{x})] \longrightarrow [v_1(\overline{x})] \wedge \cdots \wedge [u_k(\overline{x})] \longrightarrow [v_k(\overline{x})]$$

in R with $\overline{x}$ abbreviating $x_1, \ldots x_n$,

$$\frac{\begin{array}{c}[w_1] \longrightarrow [w_1'] \cdots [w_n] \longrightarrow [w_n'] \\ [\sigma(u_1(\overline{x}))] \longrightarrow [\sigma(v_1(\overline{x}))] \cdots [\sigma(u_k(\overline{x}))] \longrightarrow [\sigma(v_k(\overline{x}))]\end{array}}{[\sigma(t(\overline{x}))] \longrightarrow [\sigma'(t'(\overline{x}))]}$$

where $\sigma = \{x_1 \mapsto w_1, \ldots, x_n \mapsto w_n\}$ and
$\sigma' = \{x_1 \mapsto w_1', \ldots, x_n \mapsto w_n'\}$

Rewriting Logic

Inference rules of Rewriting Logic:

4. Transitivity.

$$\frac{[t_1] \longrightarrow [t_2] \qquad [t_2] \longrightarrow [t_3]}{[t_1] \longrightarrow [t_3]}$$

Rewriting Logic

The inference system can be proved sound and complete with respect to RWL semantics. Not discussed here.

Rewriting Logic

Summarizing computational and logical viewpoints for RWL:

State	$\leftrightarrow$	*Term*	$\leftrightarrow$	*Proposition*
Transition	$\leftrightarrow$	*Rewriting*	$\leftrightarrow$	*Deduction*
Distributed structure	$\leftrightarrow$	*Algorithmic structure*	$\leftrightarrow$	*Propositional structure*

Maude

- Maude is a language and environment based on rewriting logic.
- See: http://maude.cs.uiuc.edu/
- Features:
  - Executability — position /rule/object fair rewriting
  - High performance engine — {ACI} matching
  - Modularity and parameterization
  - Builtins — booleans, number hierarchy, strings
  - Reflection – using descent and ascent functions
  - Search and model-checking

References

Materials used to prepare these lectures:


Papers on Maude and Rewriting Logic.
http://maude.cs.uiuc.edu/papers/.


Pathway Logic web page.
http://pl.csl.sri.com/.


Maude Primer
http://maude.cs.uiuc.edu/primer/.


M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada.
A Maude Tutorial.
SRI International, 2000.

References


S. Eker, M. Knapp, K. Laderoute, P. Lincoln, and C. Talcott.
Pathway Logic: Executable models of biological networks.
ENTCS, 71, 2002.


J. Meseguer.
Bio-Pathway Logic. Slides.


J. Meseguer.
Conditional Rewriting Logic as a unified model of concurrency.
TCS, 96(1):73–155, 1992.


C. Talcott.
Pathway Logic tutorial. Parts 1 and 2. Slides.