Introduction to Term Rewriting: Techniques and Applications

Departamento de Sistemas Informáticos y Computación (DSIC)
Universidad Politécnica de Valencia (UPV)

Salvador Lucas
http://www.dsic.upv.es/~slucas

Summer School on Foundations of Information Technologies – FIT 2009
Equations, that is, expressions of the form

\[ s = t \]

are very common in mathematics, logic, artificial intelligence, and programming.

Equations are frequently used for:

1. writing programs as *sets of equations*,
2. defining *interpreters* of programming languages,
3. *requirements specification* in software engineering,
4. reasoning about program and system *properties* and implementing its (automated) *verification*,
5. *modeling* computational systems.
Introduction to Term Rewriting: Techniques and Applications

Equational reasoning

The following equations:

\[ x + 0 = x \]
\[ (x + y) + z = x + (y + z) \]
\[ x + (-x) = 0 \]
\[ x + y = y + x \]

express the axioms of Abelian groups (zero element, inverse element, associativity, and commutativity).

Variables \(x, y, z\) are universally quantified here.

Given this set of equations \(E\), we are interested in answering questions like: is \(0 + x = x\) (more precisely: \((\forall x) 0 + x = x\)) a consequence of \(E\)?

Remark

The application of rewriting techniques to this kind of problems provide a powerful framework for equational reasoning.
Equational programming

The following set of equations can be used to describe a sorting function over lists of natural numbers (in Peano's notation: $0 = 0$, $1 = s(0)$, $2 = s(s(0))$, ...)

\[
\begin{align*}
\max(0, X) & = X & \min(0, X) & = 0 \\
\max(X, 0) & = X & \min(X, 0) & = 0 \\
\max(s(X), s(Y)) & = s(\max(X, Y)) & \min(s(X), s(Y)) & = s(\min(X, Y)) \\
\end{align*}
\]

\[
\begin{align*}
\text{sort}(\text{nil}) & = \text{nil} \\
\text{sort}(\text{cons}(X, L)) & = \text{insert}(X, \text{sort}(L)) \\
\text{insert}(X, \text{nil}) & = \text{cons}(X, \text{nil}) \\
\text{insert}(X, \text{cons}(Y, L)) & = \text{cons}(\min(X, Y), \text{insert}(\max(X, Y), L))
\end{align*}
\]
Equational programming

We can use these equations to arrange the elements of the list $[2,0,1]$ in increasing ordering: in Peano's notation, this list is encoded as follows:

$$cons(s(s(0)), cons(0, cons(s(0), nil)))$$

The evaluation of

$$sort(cons(s(s(0)), cons(0, cons(s(0), nil))))$$

would 'yield' the ordered list $[0, 1, 2]$, i.e.,

$$cons(0, cons(s(0), cons(s(s(0)), nil)))$$

Remark

From a logical point of view, we would rather say that the expression

$$sort(cons(s(s(0)), cons(0, cons(s(0), nil))))$$

is equivalent to

$$cons(0, cons(s(0), cons(s(s(0)), nil)))$$

in the equational theory generated by the previous set of equations.
Rewrite rules and equations

Equations are often thought or used in a *given* direction. This is natural in *equational programming*, where the focus is in *computation*, i.e., a *state-transformation* perspective is usually assumed.

For instance, we can impose a *left-to-right* orientation to the previous equations to obtain *rewrite rules* as follows:

\[
\begin{align*}
\text{max}(0, X) & \rightarrow X \\
\text{max}(X, 0) & \rightarrow X \\
\text{max}(s(X), s(Y)) & \rightarrow s(\text{max}(X, Y)) \\
\text{min}(0, X) & \rightarrow 0 \\
\text{min}(X, 0) & \rightarrow 0 \\
\text{min}(s(X), s(Y)) & \rightarrow s(\text{min}(X, Y)) \\
\text{sort}(\text{nil}) & \rightarrow \text{nil} \\
\text{sort}(\text{cons}(X, L)) & \rightarrow \text{insert}(X, \text{sort}(L)) \\
\text{insert}(X, \text{nil}) & \rightarrow \text{cons}(X, \text{nil}) \\
\text{insert}(X, \text{cons}(Y, L)) & \rightarrow \text{cons}(\text{min}(X, Y), \text{insert}(\text{max}(X, Y), L))
\end{align*}
\]
When using Rewrite Rules instead of equations, we obtain a Term Rewriting System (TRS).

The intended computational mechanism is Term Rewriting which replaces, within a given expression, instances of the left-hand side of a rule by the corresponding instance of the right-hand side.

Remark

The evaluation of an expression like $3*(2+2)$ involves the application of rewriting techniques: subexpressions like $2+2$ are rewritten into equivalent but usually simpler ones (like 4) until no further simplification is possible. A set of equations is (silently) used to specify equivalent expressions (for instance, $2+2=4$ and $3*4=12$).
Rewriting computations

**Example** With the following TRS:

\[
\begin{align*}
\text{max}(0, X) & \rightarrow X \\
\text{max}(X, 0) & \rightarrow X \\
\text{max}(s(X), s(Y)) & \rightarrow s(\text{max}(X, Y))
\end{align*}
\]

we can compute the *maximum* among three numbers:

\[
\begin{align*}
\text{max}(s(s(s(0))), \text{max}(s(0), s(s(0)))) & \rightarrow (3) \text{max}(s(s(s(0))), s(s(0))) \\
\rightarrow (1) \text{max}(s(s(s(0))), s(s(0))) & \rightarrow (3) s(s(s(0))) \\
\rightarrow (3) s(s(s(0))) & \rightarrow (3) s(s(s(0))) \\
\rightarrow (2) s(s(s(0))) & \rightarrow (2) s(s(s(0)))
\end{align*}
\]
Rewriting and programming languages

In *programming languages* whose operational principle is (some variant of) term rewriting:

- *Programs* are *Term Rewriting Systems*.
- The *computational mechanism* is *Term Rewriting* (which is *Turing-complete*).
- A *program execution* consists of the application of *rewriting steps* to a given *initial expression* until no further step is possible (this corresponds to a *normalization semantics*).

**Examples**

Functional languages (*Haskell*, *ML*),
Algebraic languages (*Maude*),
Equational languages
The following set of rules provides a partial representation of the connectivity of the (old) WWV’05 web site (held in Valencia):

\[
\begin{align*}
\text{wwv05}(U) & \rightarrow \text{submission}(U) \\
\text{wwv05}(U) & \rightarrow \text{org}(U) \\
\text{wwv05}(U) & \rightarrow \text{valencia}(U) \\
\text{wwv05}(U) & \rightarrow \text{travelling}(U) \\
\text{submission}(U) & \rightarrow \text{entcs}(U) \\
\text{sbmlink}(U) & \rightarrow \text{login}(U) \\
\text{sbmlink}(U) & \rightarrow \text{register}(U) \\
\text{speakers}(U) & \rightarrow \text{krishnamurthi}(U) \\
\text{org}(U) & \rightarrow \text{ballis}(U) \\
\text{login}(U) & \rightarrow \text{vlogin}(U) \\
\text{blogin}(U) & \rightarrow \text{submit}(U) \\
\text{wwv05}(U) & \rightarrow \text{speakers}(U) \\
\text{wwv05}(U) & \rightarrow \text{cfp} \\
\text{wwv05}(U) & \rightarrow \text{accomodation}(U) \\
\text{submisson}(U) & \rightarrow \text{sbmlink}(U) \\
\text{submisson}(U) & \rightarrow \text{entcswwv05}(U) \\
\text{sbmlink}(U) & \rightarrow \text{forgotten}(U) \\
\text{speakers}(U) & \rightarrow \text{finkelstein}(U) \\
\text{org}(U) & \rightarrow \text{alpuente}(U) \\
\text{org}(U) & \rightarrow \text{escobar}(U) \\
\text{vlogin}(\text{slucas}) & \rightarrow \text{blogin}(\text{slucas})
\end{align*}
\]
Modeling computational systems

One can use this rewrite system to ask questions about the *reachability* of some WWW pages for some users (depending on their registration status).

Maude> search wwv05(slucas) =>+ submit(slucas) .
search in WebWWV05safe : wwv05(slucas) =>+ submit(slucas) .

Solution 1 (state 21)
No more solutions.
states: 22  rewrites: 21 in 0ms cpu (1ms real) (~ rewrites/second)

Other users *are not* allowed to visit some pages:

Maude> search wwv05(smith) =>+ submit(smith) .
search in WebWWV05safe : wwv05(smith) =>+ submit(smith) .

No solution.
states: 20  rewrites: 19 in 0ms cpu (0ms real) (~ rewrites/second)
Rewriting and programming languages

By using the concepts, results and methods of *Term Rewriting* we can investigate:

1. the *deterministic* character of computations (*confluence*),
2. how to prove that no *infinite* computation is possible (*termination*),
3. how to ensure that only *finite* computational sequences are followed, even when infinite sequences exist (*normalizing strategies*),
4. how to follow *optimal* computation paths with respect to some feasible notion of *efficiency* (*optimal strategies*),
5. methods and strategies to compute appropriate *canonical forms* (*normalizing, root-normalizing and infinitary normalizing strategies*),
6. properties of *combinations* of programs (*modularity*).
7. ...
Summary of the course

The aim of this course is providing a taste of rewriting and having a deeper contact with some essential topics like confluence and termination.

The course is structured as follows:

1. Introduction
2. Rewrite Systems and Term Rewriting
3. Termination of Rewriting
   1. The Recursive Path Ordering
   2. Termination with Polynomial Orderings
4. Confluence
References

F. Baader and T. Nipkow.
Term Rewriting and All That.

N. Dershowitz and J.-P. Jouannaud.
Rewrite Systems.

E. Ohlebusch.
Advanced Topics in Term Rewriting.

TeReSe, editor,