Motivating Examples

Equational reasoning is concerned with a rather restricted class of first-order languages: the only predicate symbol is equality. It is, however, at the heart of many problems in mathematics and computer science, which explains why developing specialized methods and tools for this type of reasoning is very popular and important. For example, in mathematics one often defines classes of algebras (such as groups, rings, etc.) by giving defining identities (which state associativity of the group operation, etc.). In this context, it is important to know which other identities can be derived from the defining ones. In algebraic specification, new operations are defined from given ones by stating characteristic identities that must hold for the defined operations. As a special case we have functional programs where functions are defined by recursion equations.

For example, assume that we want to define addition of natural numbers using the constant 0 and the successor function s. This can be done with the identities[1]

$$\begin{aligned} x + 0 &\approx x, \\ x + s(y) &\approx s(x + y). \end{aligned}$$

By applying these identities, we can calculate the sum of 1 (encoded as $s(0)$) and 2 (encoded as $s(s(0))$):

$$s(0) + s(s(0)) \approx s(s(0) + s(0)) \approx s(s(s(0)) + 0) \approx s(s(s(0))).$$

In this calculation, we have interpreted the identities as rewrite rules that tell us how a subterm of a given term can be replaced by another term.

This brings us to one of the key notions of this book, namely **term rewriting systems**. What do we mean by **terms**? They are built from **variables**, **constant symbols**, and **function symbols**. In the above example, "+" is a binary function symbol, "s" is a unary function symbol, 0 is a constant symbol, and x, y are variables. Examples of terms over these symbols are 0, x, $s(s(0))$, $x + s(0)$, $s(s(s(0)) + 0)$. In our example calculation, we have used the identities only from left to right, but in general, identities can be applied in both directions.

In the following, we give two examples that illustrate some of the key issues arising in connection with identities and rewrite systems, and which will be treated in detail in this book. In the first example, the rewrite rules are intended to be used only in one direction (which is expressed by writing $\to$ instead of $\approx$). This is an instance of rewriting as a computation mechanism. In the second, we consider the identities defining groups, which are intended to be used in both directions. This is an instance of rewriting as a deduction mechanism.

Symbolic Differentiation

We consider symbolic differentiation of arithmetic expressions that are built with the operations $+$, $*$, the indeterminates X, Y, and the numbers $0, 1$. For example, $((X + X) * Y) + 1$ is an admissible expression. These expressions can be viewed as terms that are built from the constant symbols 0, 1, X, and Y, and the binary function symbols $+$ and $*$. For the partial derivative with respect to X, we introduce the additional (unary) function symbol D_X. The following rules are (some of the)

[1] We use $\approx$ for identities to make a clear distinction between the object level sign for identity and our use of $=$ for equality on the meta-level.

well-known rules for computing the derivative:

$$\begin{array}{lrcl} \text{(R1)} & D_X(X) & \to & 1, \\ \text{(R2)} & D_X(Y) & \to & 0, \\ \text{(R3)} & D_X(u+v) & \to & D_X(u) + D_X(v), \\ \text{(R4)} & D_X(u*v) & \to & (u * D_X(v)) + (D_X(u) * v). \end{array}$$

The symbols u and v are variables in terms like $D_X(u + v)$, with the intended meaning that they can be replaced by arbitrary expressions.[2] Thus, rule (R3) can be applied to terms having the same pattern as the left-hand side, i.e., a D_X followed by a +-expression.

Starting with the term $D_X(X * X)$, the rules (R1)–(R4) lead to the possible reductions depicted in Fig. 1. We can use this example to illustrate two of the most

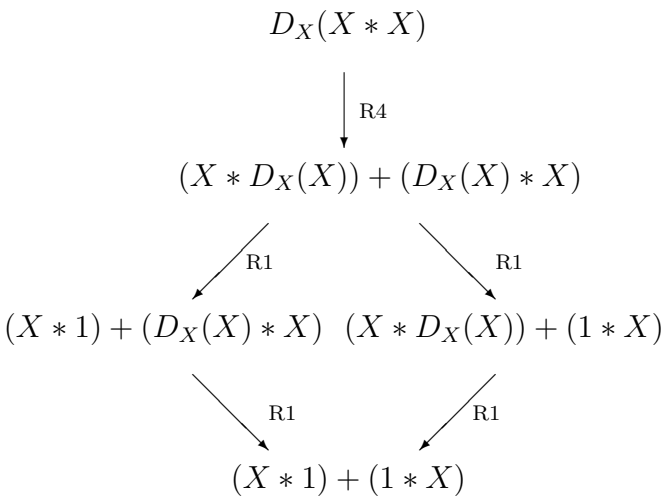

Figure 1: Symbolic differentiation of the expression $D_X(X * X)$.

important properties of term rewriting systems:

Termination: Is it always the case that after finitely many rule applications we reach an expression to which no more rules apply? Such an expression is then called a **normal form**.

For the rules (R1)–(R4) this is the case. It is, however, not completely trivial to show this. In fact, rule (R4) leads to a considerable increase in the size of the expression.

An example of a non-terminating rule is

$$u + v \to v + u,$$

which expresses commutativity of addition. The sequence $(X * 1) + (1 * X) \to (1 * X) + (X * 1) \to (X * 1) + (1 * X) \to \ldots$ is an example for an infinite chain of applications of this rule. Of course, non-termination need not always be caused by a single rule; it could also result from the interaction of several rules.

Confluence: If there are different ways of applying rules to a given term t, leading to different derived terms t_1 and t_2, can t_1 and t_2 be joined, i.e., can we always

[2] These variables should not be confused with the indeterminates X, Y of the arithmetic expressions, which are constant symbols.

find a common term s that can be reached both from t_1 and from t_2 by rule application?

In Fig. 1 this is the case, and more generally, one can show (but how?) that (R1)–(R4) are confluent. This shows that the symbolic differentiation of a given expression always leads to the same derivative (i.e., the term to which no more rules apply), independent of the strategy for applying rules.

If we add the simplification rule

$$\text{(R5)} \quad u + 0 \to u$$

to (R1)–(R4), we lose the confluence property (see Fig. 2).

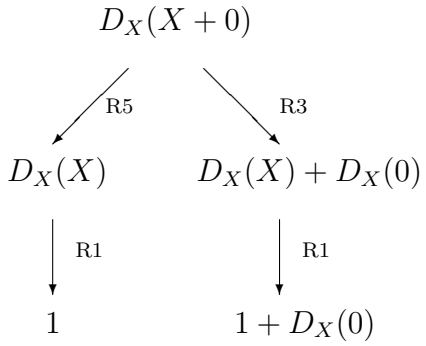

Figure 2: $D_X(X)$ and $D_X(X) + D_X(0)$ cannot be joined.

In our example, non-confluence of (R1)–(R5) can be overcome by adding the rule $D_X(0) \to 0$. More generally, one can ask whether this is always possible, i.e., can we always make a non-confluent system confluent by adding implied rules (**completion** of term rewriting systems).

Because of their special form, the rules (R1)–(R4) constitute a functional program (on the left-hand side, the defined function D_X occurs only at the very outside). Termination of the rules means that D_X is a total function. Confluence of the rules means that the result of a computation is independent of the evaluation strategy. Confluence of (R1)–(R4) is not a lucky coincidence. We will prove that all term rewriting systems that constitute functional programs are confluent.

Group Theory

Let $\circ$ be a binary function symbol, i be a unary function symbol, e be a constant symbol, and x, y, z be variable symbols. The class of all groups is defined by the identities

$$\begin{array}{lrcl} \text{(G1)} & (x \circ y) \circ z & \approx & x \circ (y \circ z), \\ \text{(G2)} & e \circ x & \approx & x, \\ \text{(G3)} & i(x) \circ x & \approx & e, \end{array}$$

i.e., a set G equipped with a binary operation $\circ$, a unary operation i, and containing an element e, is a group iff the operations satisfy the identities (G1)–(G3). Identity (G3) states only that for every group element g, the element $i(g)$ is a left-inverse of g with respect to the left-unit e. The identities (G1)–(G3) can be used to show that this left-inverse is also a right-inverse. In fact, using these identities, the term e can be transformed into the term $x \circ i(x)$:

$$
\begin{aligned}
e &\overset{G3}{\approx} i(x \circ i(x)) \circ (x \circ i(x)) \\
&\overset{G2}{\approx} i(x \circ i(x)) \circ (x \circ (e \circ i(x))) \\
&\overset{G3}{\approx} i(x \circ i(x)) \circ (x \circ ((i(x) \circ x) \circ i(x))) \\
&\overset{G1}{\approx} i(x \circ i(x)) \circ ((x \circ (i(x) \circ x)) \circ i(x)) \\
&\overset{G1}{\approx} i(x \circ i(x)) \circ (((x \circ i(x)) \circ x) \circ i(x)) \\
&\overset{G1}{\approx} i(x \circ i(x)) \circ ((x \circ i(x)) \circ (x \circ i(x))) \\
&\overset{G1}{\approx} (i(x \circ i(x)) \circ (x \circ i(x))) \circ (x \circ i(x)) \\
&\overset{G3}{\approx} e \circ (x \circ i(x)) \\
&\overset{G2}{\approx} x \circ i(x).
\end{aligned}
$$

This example illustrates that it is nontrivial to find such derivations, i.e., to solve the so-called **word problem** for sets of identities: given a set of identities E and two terms s and t, is it possible to transform the term s into the term t, using the identities in E as rewrite rules that can be applied in both directions?

One possible way of approaching this problem is to consider the identities as uni-directional rewrite rules:

$$
\begin{array}{lrcl}
\text{(RG1)} & (x \circ y) \circ z & \to & x \circ (y \circ z), \\
\text{(RG2)} & e \circ x & \to & x, \\
\text{(RG3)} & i(x) \circ x & \to & e.
\end{array}
$$

The basic idea is that the identities are only applied in the direction that "simplifies" a given term. One is now looking for normal forms, i.e., terms to which no more rules apply. In order to decide whether the terms s and t are equivalent (i.e., can be transformed into each other by applying identities in both directions), we use the uni-directional rewrite rules to reduce s to a normal form $\widehat{s}$ and t to a normal form $\widehat{t}$. Then we check whether $\widehat{s}$ and $\widehat{t}$ are syntactically equal. There are, however, two problems that must be overcome before this method for deciding the word problem can be applied:

- Equivalent terms can have distinct normal forms. In our example, both $x \circ i(x)$ and e are normal forms with respect to (RG1)–(RG3), and we have shown that they are equivalent. However, the above method for deciding the word problem would fail because it would find that the normal forms of $x \circ i(x)$ and e are distinct.

- Normal forms need not exist: the process of reducing a term may lead to an infinite chain of rule applications.

We will see that termination and confluence are the important properties that ensure existence and uniqueness of normal forms. If a given set of identities leads to a non-confluent rewrite system, we do not have to give up. We can again apply the idea of completion to extend the rewrite system to a confluent one. In the case of groups, a confluent and terminating extension of (RG1)–(RG3) exists.