1
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†

\[
\begin{align*}
\text{1.} & \quad x + 0 \approx x, \\
\text{2.} & \quad x + s(y) \approx s(x + y).
\end{align*}
\]

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

\[
s(0) + s(s(0)) \approx s(s(0) + s(0)) \approx s(s(0)) + 0 \approx s(s(0)).
\]

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**,† Throughout this book, 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.
1 Motivating Examples

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 \(\rightarrow\) instead of \(\Rightarrow\)). 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, \(((X + X) \ast Y) + 1\) 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 \(\ast\). For the partial derivative with respect to \(X\), we introduce the additional (unary) function symbol \(D_X\). The following rules are (some of the) well-known rules for computing the derivative:

\[
\begin{align*}
(R1) \quad D_X(X) & \rightarrow 1, \\
(R2) \quad D_X(Y) & \rightarrow 0, \\
(R3) \quad D_X(u + v) & \rightarrow D_X(u) + D_X(v), \\
(R4) \quad D_X(u \ast v) & \rightarrow (u \ast D_X(v)) + (D_X(u) \ast v).
\end{align*}
\]

In terms like \(D_X(u + v)\), the symbols \(u\) and \(v\) are variables, with the intended meaning that they can be replaced by arbitrary expressions. 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 \(D_X(X \ast X)\), the rules (R1)–(R4) lead to the possible reductions depicted in Fig. 1.1. We can use this example to illustrate two of the most important properties of term rewriting systems:

\(\dagger\) These variables should not be confused with the indeterminates \(X, Y\) of the arithmetic expressions, which are constant symbols.
Motivating Examples

\[ D_X(X \ast X) \]

\[ \xrightarrow{R4} \]

\[ (X \ast D_X(X)) + (D_X(X) \ast X) \]

\[ \xrightarrow{R1} \quad \xrightarrow{R1} \]

\[ (X \ast 1) + (D_X(X) \ast X) \quad (X \ast D_X(X)) + (1 \ast X) \]

\[ \xrightarrow{R1} \quad \xrightarrow{R1} \]

\[ (X \ast 1) + (1 \ast X) \]

Fig. 1.1. Symbolic differentiation of the expression \( D_X(X \ast X) \).

**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 because rule (R4) leads to a considerable increase in the size of the expression.

An example of a non-terminating rule is

\[ u + v \rightarrow v + u, \]

which expresses commutativity of addition. The sequence \( (X \ast 1) + (1 \ast X) \rightarrow (1 \ast X) + (X \ast 1) \rightarrow (X \ast 1) + (1 \ast X) \rightarrow \ldots \) 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 find a common term \( s \) that can be reached both from \( t_1 \) and from \( t_2 \) by rule application?

In Fig. 1.1 this is the case, and more generally, one can prove (but how?) that (R1)–(R4) are confluent. This shows that the symbolic differentiation of a given expression always leads to the same deri-
1 Motivating Examples

A term rewriting system is confluence (i.e. the term to which no more rules apply), independent of the strategy for applying rules.

If we add the simplification rule

\[(R5) \quad u + 0 \rightarrow u\]

to (R1)–(R4), we lose the confluence property (see Fig. 1.2).

\[
\begin{align*}
D_X(x + 0) & \\
\xrightarrow{R5} & D_X(x) \\
\xrightarrow{R3} & D_X(x) + D_X(0) \\
\xrightarrow{R1} & 1 \\
\xrightarrow{R1} & 1 + D_X(0)
\end{align*}
\]

Fig. 1.2. $D_X(x)$ 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 $D_X(0) \rightarrow 0$. 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

\[
\begin{align*}
(G1) \quad (x \circ y) \circ z & \approx x \circ (y \circ z), \\
(G2) \quad e \circ x & \approx x, \\
(G3) \quad i(x) \circ x & \approx e,
\end{align*}
\]
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 $x \circ i(x)$:

$$
eq \underset{G_3}{\approx} i(x \circ i(x)) \circ (x \circ i(x))$$
$$\underset{G_2}{\approx} i(x \circ i(x)) \circ (x \circ (e \circ i(x)))$$
$$\underset{G_3}{\approx} i(x \circ i(x)) \circ (x \circ ((i(x) \circ x) \circ i(x)))$$
$$\underset{G_1}{\approx} i(x \circ i(x)) \circ ((x \circ (i(x) \circ x)) \circ i(x))$$
$$\underset{G_1}{\approx} i(x \circ i(x)) \circ ((x \circ i(x)) \circ x) \circ i(x))$$
$$\underset{G_1}{\approx} i(x \circ i(x)) \circ ((x \circ i(x)) \circ (x \circ i(x)))$$
$$\underset{G_3}{\approx} e \circ (x \circ i(x))$$
$$\underset{G_2}{\approx} x \circ i(x).$$

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{align*}
(RG1) & \quad (x \circ y) \circ z \rightarrow x \circ (y \circ z), \\
(RG2) & \quad e \circ x \rightarrow x, \\
(RG3) & \quad i(x) \circ x \rightarrow e.
\end{align*}

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 $\hat{s}$ and $t$ to a normal form $\hat{t}$. Then we check whether $\hat{s}$ and $\hat{t}$ 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
1 Motivating Examples

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 (see Exercise 7.12 on page 184).
Abstract Reduction Systems

This chapter is concerned with the abstract treatment of reduction, where reduction is synonymous with the traversal of some directed graph, the stepwise execution of some computation, the gradual transformation of some object (e.g. a term), or any similar step by step activity. Mathematically this means we are simply talking about binary relations. An abstract reduction system is a pair \((A, \rightarrow)\), where the reduction \(\rightarrow\) is a binary relation on the set \(A\), i.e. \(\rightarrow \subseteq A \times A\). Instead of \((a, b) \in \rightarrow\) we write \(a \rightarrow b\).

The term “reduction” has been chosen because in many applications something decreases with each reduction step, but cannot decrease forever. Yet this need not be the case, as witnessed by the reduction \(0 \rightarrow 1 \rightarrow 2 \rightarrow \cdots\).

Unless noted otherwise, all our discussions take place in the context of some arbitrary but fixed abstract reduction system \((A, \rightarrow)\).

2.1 Equivalence and reduction

We can view reduction in two ways: the first is as a directed computation, which, starting from some point \(a_0\), tries to reach a normal form by following the reduction \(a_0 \rightarrow a_1 \rightarrow \cdots\). This corresponds to the idea of program evaluation. Or we may consider \(\rightarrow\) merely as a description of \(\leftrightarrow\), where \(a \leftrightarrow b\) means that there is a path between \(a\) and \(b\) where the arrows can be traversed in both directions, for example, as in \(a_0 \leftarrow a_1 \rightarrow a_2 \leftrightarrow a_3\). This corresponds to the idea of identities which can be used in both directions. The key question here is to decide if two elements \(a\) and \(b\) are equivalent, i.e. if \(a \leftrightarrow b\) holds. Settling this question by an undirected search along both \(\rightarrow\) and \(\leftarrow\) is bound to be expensive. Wouldn’t it be nice if we could decide equivalence by reducing both \(a\) and \(b\) to their normal forms and testing if the normal forms are identical? As explained in the first chapter, this idea is only going to work if reduction terminates and normal forms are unique.
Formally, we talk about *termination* and *confluence* of reduction, and the study of these two notions is one of the central themes of this book.

### 2.1.1 Basic definitions

In the sequel, we define a great many symbols, not all of which will be put to immediate use. Therefore you may treat these definitions as a table of relevant notions which can be consulted when necessary.

Given two relations \( R \subseteq A \times B \) and \( S \subseteq B \times C \), their **composition** is defined by

\[
R \circ S := \{(x, z) \in A \times C \mid \exists y \in B. (x, y) \in R \land (y, z) \in S\}
\]

**Definition 2.1.1** We are particularly interested in composing a reduction with itself and define the following notions:

\[
\begin{align*}
\xrightarrow{0} &:= \{(x, x) \mid x \in A\} \quad \text{identity} \\
\xrightarrow{i+1} &:= \xrightarrow{i} \circ \xrightarrow{1} \quad \text{\((i+1)\)-fold composition, } i \geq 0 \\
\xrightarrow{i} &:= \bigcup_{i \geq 0} \xrightarrow{i} \quad \text{transitive closure} \\
\xrightarrow{\ast} &:= \xrightarrow{i} \cup \xrightarrow{0} \quad \text{reflexive transitive closure} \\
\xrightarrow{\equiv} &:= \xrightarrow{\ast} \cup \xrightarrow{\leftarrow} \quad \text{reflexive closure} \\
\xrightarrow{\downarrow} &:= \{(y, x) \mid x \rightarrow y\} \quad \text{inverse} \\
\xrightarrow{\leftarrow} &:= \xrightarrow{\downarrow} \quad \text{inverse} \\
\xrightarrow{\leftrightarrow} &:= \xrightarrow{\downarrow} \cup \xrightarrow{\leftarrow} \quad \text{symmetric closure} \\
\xrightarrow{\leftrightarrow}^{\ast} &:= (\leftrightarrow)^{\ast} \quad \text{reflexive transitive symmetric closure} \\
\xrightarrow{\ast} &:= (\leftrightarrow)^{\ast} \quad \text{reflexive transitive symmetric closure}
\end{align*}
\]

Some remarks are in order:

1. Notations like \( \xrightarrow{\ast} \) and \( \xrightarrow{\leftarrow} \) only work for arrow-like symbols. In the case of arbitrary relations \( R \subseteq A \times A \) we write \( R^n \), \( R^{-1} \) etc.

2. Some of the constructions can also be expressed nicely in terms of paths:
   \[x \xrightarrow{n} y\text{ if there is a path of length } n \text{ from } x \text{ to } y,
   x \xrightarrow{\downarrow} y\text{ if there is some finite path from } x \text{ to } y,
   x \xrightarrow{\leftrightarrow} y\text{ if there is some finite nonempty path from } x \text{ to } y.
\]

3. The word **closure** has a precise meaning: the \( P \) closure of \( R \) is the least set with property \( P \) which contains \( R \). For example, \( \xrightarrow{\ast} \), the reflexive transitive closure of \( \xrightarrow{\leftrightarrow} \), is the least reflexive and transitive relation which contains \( \xrightarrow{\rightarrow} \). Note that for arbitrary \( P \) and \( R \), the \( P \) closure of \( R \) need not exist, but in the above cases they always do because reflexivity, transitivity and symmetry are closed under arbitrary intersections. In
2.1 Equivalence and reduction

such cases the $P$ closure of $R$ can be defined directly as the intersection of all sets with property $P$ which contain $R$.

4. It is easy to show that $\leftrightarrow$ is the least equivalence relation containing $\rightarrow$.

Let us add some terminology to this notation:

1. $x$ is **reducible** iff there is a $y$ such that $x \rightarrow y$.
2. $x$ is in **normal form** (irreducible) iff it is not reducible.
3. $y$ is a **normal form of** $x$ iff $x \rightarrow^* y$ and $y$ is in normal form. If $x$ has a uniquely determined normal form, the latter is denoted by $x \downarrow$.
4. $y$ is a **direct successor** of $x$ iff $x \rightarrow y$.
5. $y$ is a **successor** of $x$ iff $x \rightarrow^* y$.
6. $x$ and $y$ are **joinable** iff there is a $z$ such that $x \rightarrow^* z \rightarrow^* y$, in which case we write $x \downarrow y$.

**Example 2.1.2**

1. Let $A := \mathbb{N} - \{0, 1\}$ and $\rightarrow := \{(m, n) \mid m > n \text{ and } n \text{ divides } m\}$. Then
   
   (a) $m$ is in normal form iff $m$ is prime.
   
   (b) $p$ is a normal form of $m$ iff $p$ is a prime factor of $m$.
   
   (c) $m \downarrow n$ iff $m$ and $n$ are not relatively prime.
   
   (d) $\leftrightarrow = \rightarrow$ because $>$ and “divides” are already transitive.
   
   (e) $\leftrightarrow = A \times A$.

2. Let $A := \{a, b\}^*$ (the set of words over the alphabet $\{a, b\}$) and $\rightarrow := \{\langle ubav, uabv \rangle \mid u, v \in A\}$. Then
   
   (a) $w$ is in normal form iff $w$ is sorted, i.e. of the form $a^*b^*$.
   
   (b) Every $w$ has a unique normal form $w\downarrow$, the result of sorting $w$.
   
   (c) $w_1 \downarrow w_2$ iff $w_1 \rightarrow \leftrightarrow w_2$ iff $w_1$ and $w_2$ contain the same number of $a$s and $b$s.

Finally we come to some of the central notions of this book.

**Definition 2.1.3**  A reduction $\rightarrow$ is called

**Church-Rosser**† iff $x \leftrightarrow y \Rightarrow x \downarrow y$ (see Fig. 2.1).

**confluent** iff $y_1 \leftrightarrow x \rightarrow y_2 \Rightarrow y_1 \downarrow y_2$ (see Fig. 2.1).

**terminating** iff there is no infinite descending chain $a_0 \rightarrow a_1 \rightarrow \cdots$

**normalizing** iff every element has a normal form.

**convergent** iff it is both confluent and terminating.

Both reductions in Example 2.1.2 terminate, but only the second one is Church-Rosser and confluent.

† Alonzo Church and J. Barkley Rosser proved that the $\lambda$-calculus has this property [51].
2 Abstract Reduction Systems

![Diagram](https://via.placeholder.com/150)

Fig. 2.1. Church-Rosser property, confluence and semi-confluence.

Remarks:

1. The diagrams in Fig. 2.1 have a precise meaning and are used throughout the book in this manner: solid arrows represent universal and dashed arrows existential quantification; the whole diagram is an implication of the form $\forall \bar{x}. P(\bar{x}) \Rightarrow \exists \bar{y}. Q(\bar{x}, \bar{y})$. For example, the confluence diagram becomes $\forall x, y_1, y_2. y_1 \not\rightarrow x \rightarrow y_2 \Rightarrow \exists z. y_1 \not\rightarrow z \leftarrow y_2$.

2. Because $x \downarrow y$ implies $x \leftrightarrow y$, the Church-Rosser property can also be phrased as an equivalence: $x \leftrightarrow y \Leftrightarrow x \downarrow y$.

3. Any terminating relation is normalizing, but the converse is not true, as the example in Fig 2.2 shows.

![Diagram](https://via.placeholder.com/150)

Fig. 2.2. Confluent, normalizing and acyclic but not terminating.

Thus we have come back to our initial motivation: the Church-Rosser property is exactly what we were looking for, namely the ability to test equivalence by the search for a common successor. We will now see how it relates to termination and confluence.

2.1.2 Basic results

It turns out that the Church-Rosser property and confluence coincide. The fact that any Church-Rosser relation is confluent is almost immediate, and the reverse implication has a beautiful diagrammatic proof which is shown in Fig. 2.3. It is based on the observation that any equivalence $x \leftrightarrow y$ can be