PROCESS ALGEBRA

J.C.M. Baeten and W.P. Weijland
Centre for Mathematics and Computer Science, Amsterdam

CAMBRIDGE
UNIVERSITY PRESS

Published by the Press Syndicate of the University of Cambridge
The Pitt Building, Trumpington Street, Cambridge, CB2 1RP
40 West 20th Street, New York, NY10011-4211, USA
10 Stamford Road, Oakleigh, Melbourne 3166, Australia

First published 1990
Reprinted 1995

Printed in Great Britain by The Ipswich Book Company,
Ipswich, Suffolk

Library of Congress cataloguing in publication data available

British Library cataloguing in publication data available

ISBN 0 521 40043 0

Contents

Chapter 1

Preliminaries

1.1 INTRODUCTION

This book provides a systematic introduction to process algebra, suitable for teaching purposes. By the term *process algebra* we mean the study of concurrent communicating processes in an algebraic framework, following the approach of J.A. Bergstra and J.W. Klop (see section 8.1). In the present book we treat concurrency theory (the theory of concurrent communicating processes) in an *axiomatic* way, just as for instance the study of mathematical objects as groups or fields starts with an axiomatization of the intended objects. This axiomatic method is algebraic in the sense that we consider structures (also called "process algebras") as models of some set of (mostly) equational axioms. These structures are equipped with several operators, and so we use the term *algebra* in the sense of model theory. R. Milner, with his Calculus of Communicating Systems (CCS), is generally considered to be the initiator of the field of process algebra. CCS forms the basis for most of the axiom systems presented below.

There is ample motivation for such an axiomatic-algebraic approach to concurrency theory. The main reason is that there is not one definitive notion of *process*. There is a staggering amount of properties which one may or may not attribute to processes, there are dozens of views (*semantics*) which one may have on processes, and there are infinitely many models of processes. So an attempt to organize this field of process theories leads very naturally and almost unavoidably to an axiomatic methodology. A curious consequence is that one has to answer the question "What is a process?" with the seemingly circular answer "A process is something that obeys a certain set of axioms ... for processes". The axiomatic method has proven effective in mathematics and mathematical logic. In our opinion it has its merits in computer science as well, if only for its organizing and unifying power.

Next to the organizing role of this set-up with axiom systems, their models and the study of their relations, we have the obvious *computational* aspect. Even more than in mathematics and mathematical logic, in computer science it is *algebra* that counts. For instance, in a system verification the use of transition diagrams may be very illuminating. For larger systems, however, it may be desirable to have a formalized mathematical language at our disposal in

which specifications, computations and proofs can be given in what is in principle a linear notation. Only then can we hope to succeed in attempts to mechanize formal dealings with the objects of interest. In our case the mathematical language is algebraic, with basic constants, operators to construct larger processes, and equations defining the nature of the processes under consideration. (The format of pure equations will not be enough, though. On occasion, we will use conditional equations and some infinitary proof rules.) To be specific: we will always insist on the use of congruences, rather than mere equivalences in the construction of process algebras, in order to preserve the purely algebraic format.

A further advantage of the use of the axiomatic-algebraic method is that the entire apparatus of mathematical logic and the theory of abstract data types is at our service. For instance, one can study extensions of axiom systems that are homomorphisms of the corresponding process algebras, and one can formulate exact statements as to the relative expressibility of some process operators (definability results).

Of course, the present axiomatizations for concurrency theory do not cover the entire spectrum of interest. Several aspects of processes are as yet not well treated in the algebraic framework. The most notable examples concern the real-time behaviour of processes, and what is called *true concurrency* (non-interleaving semantics). Algebraic theories for these aspects are under development at the moment, however.

In our view, process algebra can be seen as a worthy descendant of "classical" automata theory as it originated three or four decades ago. The crucial difference is that nowadays one is interested not merely in the execution traces (or language) of one automaton, but in the behaviour of systems of communicating automata. As Milner and also Hoare, in his Communicating Sequential Processes (CSP), have discovered, it is no longer sufficient to abstract the behaviour of a process to a language of execution traces. Instead, one has to work with more discriminating process semantics, in which also the timing of choices of a system component is taken into account. Mathematically, this difference is very sharply expressed in the equation $x\cdot(y + z) = x\cdot y + x\cdot z$, where + denotes choice and $\cdot$ is sequential composition; x,y,z are processes. If one is interested in languages of execution traces (trace semantics), this equation holds, but in process algebra it will in general not hold. Nevertheless, process algebra retains the option of adding this equation and studying its effect. In fact, one goal of process algebra is to form a uniform framework in which several different process semantics can be compared and related (see chapter 7). One can call this *comparative concurrency semantics.*

We bring structure in our theory of process algebra by *modularization*, i.e. we start from a minimal theory (containing only the operators $+,\cdot$), and then add new features one at a time. This allows us to study features in isolation, and to combine the modules of the theory in different ways.

The book contains enough material for a one-year graduate course for students of computer science or mathematics. Also, shorter courses can be given. A short course may for example consist of chapters 2,3,4, and optionally chapter 5 or chapter 7.

There are no specific prerequisites, but some exposure to mathematics, specifically algebra and logic, will come in handy.

Every section, except this one, will be concluded with a number of exercises, which the reader can use to check his understanding of the material. Also, some theory is treated in the

exercises, but such theory will not be used later on, so that the exercises can be skipped without problems.

We conclude with a short overview of the contents. In the rest of this chapter, we give a review of the algebraic notions that we will use in the remaining chapters. In chapter 2 we discuss the basic theory, with alternative composition (non-deterministic choice) and sequential composition as operators. We formulate a few simple laws for these operators, and discuss subjects like recursion (processes specified by means of recursive equations). We also take a look at various models for the theory.

In chapter 3 we add parallel composition, and in chapter 4 communication. A large example of how to use the resulting theory in practical applications is a specification of the alternating bit protocol.

In chapter 5 we consider at length the difficult issue of abstraction. We look at notions like fairness, and verify the alternating bit protocol specified in chapter 4.

In chapter 6 we discuss some additional features and operators that can be used in certain applications. Chapter 7 considers different ways to give semantics for theories of concurrency, and chapter 8 looks at the relationship of the theory that is presented in this book with other concurrency theories, specifically CCS and CSP. The book concludes with a bibliography on concurrency.

1.2 TERMS AND EQUATIONS

1.2.1 DEFINITION

We begin with the concept of an **equational specification** (Σ, E). Here, E is a set of **equations** of the form $t_1 = t_2$ where t_1 and t_2 are **terms** and Σ is the **signature**, i.e. the set of constant and function symbols that may appear in the specification. Σ also gives the **arity** of each function symbol (the number of arguments). The equations are often referred to as **axioms**.

1.2.2 EXAMPLE

The equational specification E_1 in table 1 describes the natural numbers, and has a constant symbol 0 and function symbols s (successor), a (addition) and m (multiplication).

We see that signature Σ_1 has a constant symbol 0, and function symbols s, a and m of arity 1, 2 and 2 respectively. A function symbol of arity 1 is called **unary**, and of arity 2 **binary**. A function symbol is sometimes called an **operator** symbol.

a(x, 0) = x
a(x, s(y)) = s(a(x, y))
m(x, 0) = 0
m(x, s(y)) = a(m(x, y), x)

TABLE 1.

1.2.3 NOTE

We talk about function *symbols* and constant *symbols*. We do this in order to distinguish between these purely formal objects and "real" functions (or operators) and constants in "real" algebras, which we will discuss later in 1.3. For now, we play a purely formal game with sets of symbols and equations that have no meaning as yet.

1.2.4 DEFINITION

The four axioms of E_1 also contain **variables** x,y. We will always assume that every signature Σ contains as many variables as we want. We denote variables by x, y, z, ..., possibly subscripted.

1.2.5 DEFINITION

Now we can define inductively the notion of a **term**:

i. variables x, y, ... are terms;

ii. constant symbols c, c', ... are terms;

iii. if F is a function symbol of arity n, and $t_1,...,t_n$ are terms, then $F(t_1,...,t_n)$ is a term.

A term that contains a variable is called an **open** term; a term without variables is called a **closed** or **ground** term.

1.2.6 DEFINITIONS

In an equation with open terms like

$$m(x, s(y)) = a(m(x, y), x) \qquad (1)$$

we may substitute terms from the signature Σ for the variables. For example, we can substitute s(s(0)) for x and 0 for y to obtain

$$m(s(s(0)), s(0)) = p(m(s(s(0)), 0), s(s(0))) \qquad (2).$$

The variables x and y are *bound* to the terms s(s(0)) and 0 respectively. If a variable x is bound to a term, this term must be substituted in every occurrence of x. So separate occurrences of a variable cannot be bound to different terms.

We have just shown that equation (2) can be **derived** from E_1; in symbols

$$E_1 \vdash m(s(s(0)), s(0)) = p(m(s(s(0)), 0), s(s(0))).$$

In general, derivability of an equation either means that it is *in* E, by the rule

i. $s=t \in E$ implies $E \vdash s=t$,

or that it can be obtained from E by means of the following three rules:

ii. **substitution**:

$E \vdash t(x_1,....,x_n) = s(x_1,....,x_n)$ implies $E \vdash t(t_1,....,t_n) = s(t_1,....,t_n)$;

iii. forming **contexts**:

$E \vdash t=s$ implies $E \vdash C[t] = C[s]$,

where C[ ] is a *context*, i.e. a term containing a hole [ ] as in m(x,[ ]). In other words: if $E \vdash t=s$, then in every occurrence of t as a *subterm* in a larger term it can be replaced by s. In this formulation the rule is often referred to as the *replacement* rule.

iv. the **equivalence** properties of =, namely:

symmetry: $E \vdash t=s$ implies $E \vdash s=t$;

reflexivity: $E \vdash t=t$;

transitivity: $E \vdash t=s$ and $E \vdash s=u$ imply $E \vdash t=u$.

1.2.7 EXAMPLE

We prove that $E_1 \vdash a(s(0), s(0)) = s(s(0))$:

1. $E_1 \vdash a(x, s(y)) = s(a(x, y))$ (the second equation of E_1)
2. $E_1 \vdash a(s(0), s(0)) = s(a(s(0), 0))$ (substitution in line 1)
3. $E_1 \vdash a(x, 0) = x$ (the first equation of E_1)
4. $E_1 \vdash a(s(0), 0) = s(0)$ (substitution in line 3)
5. $E_1 \vdash s(a(s(0), 0)) = s(s(0))$ (using the context $s([\ ])$ and line 4)
6. $E_1 \vdash a(s(0), s(0)) = s(s(0))$ (transitivity of =, from 2 and 6)

(In this way, we can make 1+1=2 difficult!)

Usually, we are not so long-winded, and will write down the proof above as follows:

$E_1 \vdash p(s(0), s(0)) = s(p(s(0), 0))$ (from the second equation)

$= s(s(0))$ (from the first equation).

1.2.8 DEFINITION

Another important notion is that of the **occurrence** of a term s in a term t. For instance, the term s(0) occurs twice in the term m(s(s(0)), s(0)). When s occurs in t, we call s a **subterm** of t.

We use the notation ≡ for **syntactical identity**, i.e. $t \equiv s$ when t,s are identical terms. Note that ≡ is different from =. For instance, we have

$E_1 \vdash p(0, 0) = 0$, but not $p(0, 0) \equiv 0$.

1.2.9 EXAMPLE

We add to the signature Σ_1 of example 1.2.2 a binary function symbol e (exponentiation). We extend the set of equations E_1 to E_2 in table 2.

$a(x, 0) = x$
$a(x, s(y)) = s(a(x, y))$
$m(x, 0) = 0$
$m(x, s(y)) = a(m(x, y), x)$
$e(x, 0) = s(0)$
$e(x, s(y)) = m(e(x, y), x)$

TABLE 2.

Using a more suggestive notation, we can write down E_2 as in table 3.

$x + 0 = x$
$x + s(y) = s(x + y)$
$x \cdot 0 = 0$
$x \cdot s(y) = x \cdot y + x$
$x^0 = s(0)$
$x^{s(y)} = x^y \cdot x$

TABLE 3.

1.2.10 CONDITIONAL EQUATIONS

Sometimes we will use implication symbols $\Rightarrow$ in our specifications, interpreted as logical implication. An axiom of the form $G \Rightarrow s=t$ (sometimes written as $\frac{G}{s=t}$), with G a set of equations, is called a **conditional equation** (or **conditional axiom**). If G is an infinite set, we talk about an **infinitary conditional equation**. A specification in which a conditional equation occurs is called a **conditional specification**. What we call a conditional specification, is called a **theory** in mathematical logic. In the literature equational specifications (thus without the symbol $\Rightarrow$) are often called **algebraic** specifications.

In the case of conditional equations, we extend the definition of derivability in 1.2.6 with the following clause:

v. If the conditional equation $G \Rightarrow s=t$ is in E, and, for a certain substitution, all substitution instances of G are derivable from E, then the substitution instance of $s=t$ is derivable from E (using the same substitution).

1.2.11 EXERCISES

1. Show that $E_1 \vdash m(s(s(0)),s(s(0))) = a(s(s(0)),s(s(0)))$.
2. Show by an inductive argument that for every closed term t over Σ_1, either $E_1 \vdash t=0$, or there is a term t' such that $E_1 \vdash t = s(t')$.

It follows that for every closed term t over Σ_1, there is a term t' of the form $s^n(0)$ $(n \geq 0)$ such that $E_1 \vdash t = t'$. (The terms $s^n(0)$ are defined inductively: $s^0(0) \equiv 0$, and $s^{n+1}(0) \equiv s(s^n(0))$.)

3. The same as exercise 2, but now for (Σ_2,E_2).
4. Prove that (a) $E_2 \vdash x + (y + 0) = (x + y) + 0$, and (b) if for a closed term t we have $E_2 \vdash x + (y + t) = (x + y) + t$, then also $E_2 \vdash x + (y + s(t)) = (x + y) + s(t)$.

Notice that this proves the **associativity** $x + (y + z) = (x + y) + z$ for all *closed* terms z (use exercise 3).

5. Prove that $E_2 \vdash x \cdot (y + z) = x \cdot y + x \cdot z$ for all *closed* terms x,y,z. Hint: write z in the form $s^n(0)$ and use exercise 4.
6. Let $E_3 = \{F(F(F(x)))=x, F(F(F(F(F(x)))))=x\}$. Prove that $E_3 \vdash F(x) = x$.

1.3 ALGEBRAS

1.3.1 SEMANTICS

Until now, we have only talked about **syntax**: sets of equations without meaning. Now we discuss the meaning or **semantics** of specifications (Σ,E). We talk about an **algebra** or **model**: an algebra $\mathbb{A}$ consists of a set of elements, A, together with constants in A and functions f from A^n to A (where n is the arity of f). The set A is called the **universe** or **domain** of $\mathbb{A}$.

1.3.2 EXAMPLES

$(\mathbf{N},+,\cdot,s,0)$ is the algebra of the natural numbers ($\mathbf{N} = \{0,1,2,...\}$) with functions $+$, $\cdot$, s and constant 0. $(\mathbf{Z},+,\cdot,s,0)$ is the algebra of the integers ($\mathbf{Z} = \{..-2,-1,0,1,2,...\}$) with the same signature, but with a larger domain. $(\mathbf{Z},+,\cdot,p,s,0)$ is the algebra of integers with the predecessor

operator p, and has a richer signature than (**Z**,+,·,s,0), but apart from the presence of p they are the same.

Another example is the algebra of the Booleans $\mathbb{B}$=(**B**,xor,and,not,0), where **B** = {0,1}, 0 is a constant symbol ("falsity"), and and not are the usual conjunction and negation operators and xor is the "exclusive or" operator, defined by:

xor(x, y)=1 ⇔ x=1 or y=1, but not: x=1 *and* y=1.

It is important to see that (by definition) the domain of an algebra is closed with respect to function applications. For instance, if we would want to define an algebra (**N**,+,·,p,s,0) of natural numbers with predecessor then we cannot just adopt the definition of p from the larger model (**Z**,+,·,p,s,0) since in the latter we have that p(0) = -1 which is not an element in **N**. Therefore, under the same interpretation of p the algebra (**N**,+,·,p,s,0) is not well-defined.

1.3.3 DEFINITION

If Σ is a signature, then we call the algebra $\mathbb{A}$ a **Σ-algebra** when there is a correspondence between the constant symbols in Σ and the "real" constants, and between the formal function symbols in Σ and the "real" functions with the same arity in $\mathbb{A}$. Such a correspondence is called an **interpretation.**

For example, if Σ_1={a,m,s,0} like in example 1.2.2, then we can make (**N**,+,·,s,0) into a Σ_1-algebra by means of the interpretation

a — +
m — ·
s — s
0 — 0.

Notice that there is also another interpretation, viz.

a — ·
m — +
s — s
0 — 0.

1.3.4 DEFINITION

If $\mathbb{A}$ is a Σ-algebra, then the equation $t_1 = t_2$ over (Σ,E) has a meaning in $\mathbb{A}$, when we interpret the constant and function symbols in t_1, t_2 by the corresponding constants and functions in $\mathbb{A}$. Further, the variables x,y,.... in t_1,t_2 are always **universally quantified**, e.g.

$$a(x, s(y)) = s(a(x, y)) \qquad (*)$$

means in (**N**,+,·,s,0) that:

$$\text{for all } n,m \in \mathbf{N} \quad n + s(m) = s(n + m) \qquad (**).$$

If this second statement (**) is actually **true** in $\mathbb{A}$, we write

$$\mathbb{A} \models a(x, s(y)) = s(a(x, y))$$

and say: $\mathbb{A}$ **satisfies** (*), or (*) **holds in** $\mathbb{A}$.

For a conditional equation, this works similarly; the equation

$$x + s(0) = y \Rightarrow y = s(x)$$

means in (**N**,+,·,s,0) that:

$$\text{for all } n,m \in \mathbf{N}, \text{ if } n + 1 = m, \text{ then } m = s(n).$$

If the Σ-algebra $\mathbb{A}$ satisfies all equations $t_1 = t_2$ of E, we use the abbreviation

$$\mathbb{A} \models E.$$

If this is the case, we say that $\mathbb{A}$ is an **algebra for** E, or a **model of** E. We also say that E is a **sound axiomatization** of $\mathbb{A}$.

1.3.5 EXAMPLE

We have $(\mathbf{N},+,\cdot,s,0) \models E_1$ (E_1 as in 1.2.2) under the first interpretation of 1.3.3 (*not* under the other one). One might think that $\mathbb{N} = (\mathbf{N},+,\cdot,s,0)$ is the *only* model for E_1, but that is not the case: this algebra is not uniquely determined. Another algebra that satisfies E_1 is $\mathbb{B} = (\mathbf{B}, \text{xor}, \text{and}, \text{not}, 0)$ from 1.3.2, under the interpretation

a — xor
m — and
s — not
0 — 0.

$\mathbb{B}$ satisfies even more equations than $(\mathbf{N},+,\cdot,s,0)$, because $\mathbb{B} \models s(s(x)) = x$, but $\mathbb{A} \not\models s(s(x)) = x$ ($\not\models$ means: does not satisfy). Thus, in general a specification has more than one model.

1.3.6 INITIAL ALGEBRA

Let us write $\mathsf{Alg}(\Sigma,E)$ for the set of Σ-algebras $\mathbb{A}$ with $\mathbb{A} \models E$. Then there is one special algebra in $\mathsf{Alg}(\Sigma,E)$, the so-called **initial algebra** of (Σ,E) – denoted by $I(\Sigma,E)$ – that satisfies *only* E and nothing more. To be more precise: the domain of $I(\Sigma,E)$ consists of equivalence classes of closed terms over (Σ,E), such that two closed terms s and t are equivalent iff $E \vdash s=t$.

As a consequence, $I(\Sigma,E)$ satisfies only those equations between closed terms that are formally derivable from E, and no others, which is expressed as follows:

For all closed terms t,s $I(\Sigma,E) \models t=s \;\Leftrightarrow\; (\Sigma,E) \vdash t=s$

One can think of the initial algebra as the *set of closed terms* (over Σ) *modulo derivability*.

There may be several algebras in $\mathsf{Alg}(\Sigma,E)$ that satisfy the property in the box (i.e. equations between closed terms hold iff they are derivable from the theory). Such an algebra is called **complete** for the theory (Σ,E), or the theory is called a **complete axiomatization** of the algebra.

1.3.7 EXAMPLE

Let (Σ_1,E_1) be as in 1.2.2 (table 1). Then, the initial algebra $I(\Sigma_1,E_1)$ has a domain with elements as in fig. 1.

Every element is an equivalence class of closed terms, and two terms belong to the same equivalence class exactly when E_1 proves that they are equal. In fact, $I(\Sigma_1,E_1)$ is "the same" as $(\mathbf{N},+,\cdot,s,0)$. To be exact: they are *isomorphic*, meaning that there exists a one-to-one correspondence between the domain elements of both models preserving the function equalities.

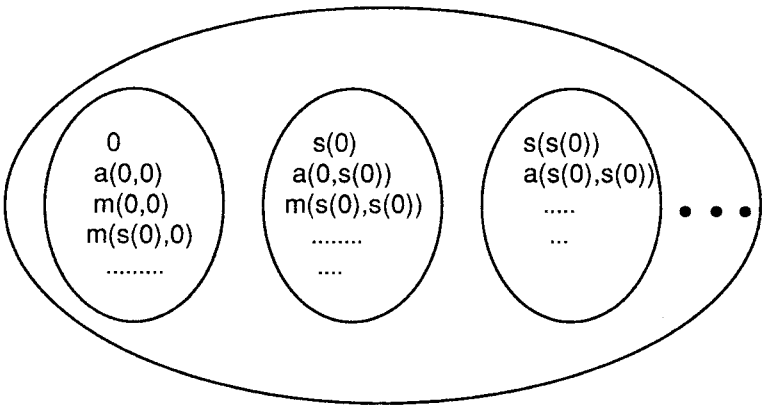

FIGURE 1.

1.3.8 DEFINITION

Suppose we want to alter a model by identifying elements from its domain by means of some equivalence relation (thus decreasing the number of its elements). Then we may wonder whether the resulting structure – with *classes* of elements in its domain, instead of the elements themselves – is again a model. As it turns out, this is not always the case: we need to make sure that in the new model the function values remain independent from the choice of the elements from its equivalence class.

Let $\mathbb{A}$ be a Σ-algebra, then a **congruence** on $\mathbb{A}$ is a binary relation R on the domain A of $\mathbb{A}$ that satisfies the following requirements:

i. R is an equivalence relation on A (i.e. reflexive, symmetric and transitive);

ii. for every n and every n-ary function f of Σ, we have for all $a_1,..,a_n$, $b_1,...b_n \in A$ the following implication:

$R(a_1, b_1)$ & .... & $R(a_n, b_n) \Rightarrow R(f(a_1,...,a_n), f(b_1,...,b_n))$.

This second requirement states that R behaves correctly with respect to functions of Σ. The relation R can be seen as an intended equality relation between objects.

1.3.9 DEFINITION

If R is a congruence on $\mathbb{A}$, then we can obtain a new Σ-algebra by factoring R out on $\mathbb{A}$. The new algebra is called $\mathbb{A}/R$, pronounced $\mathbb{A}$ **modulo** R, and the elements of this algebra are the equivalence classes of R on $\mathbb{A}$. If we denote the equivalence class of a ($\{b \in A : R(a, b)\}$) by [a], then the interpretation of the constants and functions in $\mathbb{A}/R$ is as follows:

a is mapped to [a];

$f([a_1], ..., [a_n]) = [f(a_1,...,a_n)]$.

In order to characterize the initial algebra exactly, we need the following important property of this factoring construction.

1.3.10 THEOREM

The intersection of a family of congruences is again a congruence.

PROOF: see exercises.

1.3.11 PROPOSITION
Now let a signature Σ be given which generates at least one closed term and let $I(\Sigma)$ denote the set of closed terms over Σ. In fact, $I(\Sigma)$ is a Σ-algebra itself, since the set of closed terms contains the constants and is closed under the functions. Now let a set of equations E be given, and define R_E to be the intersection of all congruence relations R on $I(\Sigma)$ with the property that $I(\Sigma)/R \models E$. Then $I(\Sigma)/R_E$ is exactly $I(\Sigma,E)$ (i.e. is isomorphic to it).

1.3.12 TERMINOLOGY
The initial algebra $I(\Sigma,E)$ is also called the **term model** of (Σ,E). For example, the term model of $(\{a,m,s,0\}, E_1)$ (see 1.2.2) is isomorphic to the model of natural numbers $(\mathbf{N},+,\cdot,s,0)$.

In the literature, the expression **abstract data type** is used in many different ways. In one of them, $Alg(\Sigma,E)$ is an abstract data type, but in another it is $I(\Sigma,E)$.

1.3.13 NOTE
It is possible that there is an equation $t = s$ between *open* terms t,s, that holds in an initial algebra $I(\Sigma,E)$, but that does not follow from E. An example of this is the equation $x + y = y + x$, which holds in $(\mathbf{N},+,\cdot,s,0)$, but is not derivable from E_1 (see exercises).

1.3.14 THEOREM
The following is the **completeness theorem** for conditional equational theories:

for all open terms t,s:
$(\Sigma,E) \vdash t=s \;\Leftrightarrow\;$ for all $\mathbb{A} \in Alg(\Sigma,E)$: $\mathbb{A} \models t=s$

A corollary of this theorem is that E_1 has a model in which $x + y = y + x$ does not hold (see note 1.3.13).

1.3.15 EXERCISES
1. Show that $\mathbb{B}$ in 1.3.5 is indeed a model of (Σ_1,E_1).
2. Find a model of (Σ_1,E_1) in which $x + y = y + x$ does not hold.
3. Verify that $\mathbb{B} \models s(m(x, y)) = a(a(s(x), s(y)), m(s(x),s(y)))$ ($\mathbb{B}$ from 1.3.5). Does this formula also hold in $(\mathbf{N},+,\cdot,s,0)$?
4. Prove theorem 1.3.10.

1.4 TERM REWRITING SYSTEMS

1.4.1 MOTIVATION
Let us look again at the equational specification (Σ_1,E_1) of 1.2.2, that has the equations in table 1 (copied below).

$a(x,0) = x$
$a(x,s(y)) = s(a(x,y))$
$m(x,0) = 0$
$m(x,s(y)) = a(m(x,y),x)$

Now we can "calculate" that "2 + 2 = 4":

a(s(s(0)),s(s(0))) =
s(a(s(s(0)),s(0))) =
s(s(a(s(s(0)),0))) =
s(s(s(s(0)))).

The equations in table 1 have a certain *direction* when applied in this derivation: when read from left to right they simplify terms, they *reduce* them. Therefore, we write → instead of =, as in table 4.

a(x,0) → x
a(x,s(y)) → s(a(x,y))
m(x,0) → 0
m(x,s(y)) → a(m(x,y),x)

TABLE 4.

This is an example of a term rewriting system or term reduction system (TRS).

1.4.2 DEFINITION
A **term rewriting system** (Σ,R) consists of a signature Σ and a set of **(rewrite) rules** R. The rules have the form $t_1 \rightarrow t_2$, where t_1 and t_2 are terms over Σ (as defined in 1.2.5). Moreover, we must have:
i. t_1 is not just a variable;
ii. every variable that occurs in t_2, must already occur in t_1 (a rewrite rule may not introduce any variables).

1.4.3 DEFINITION
Just like in 1.2.6, we can *derive* reductions from a TRS. We define a relation → on terms (**one step reduction**) as follows: $t_1 \rightarrow t_2$ holds when this is derivable from the rewrite rules R by means of:
i. substitution;
ii. forming contexts.

Thus, the rewrite steps in example 1.4.1 (where we reduced 2+2 to 4) are all one step reductions.

Then, we define the relation ↠ (**reduction relation**) on terms as follows:

$t_1 \twoheadrightarrow t_2$ holds when there are a number of one-step reductions that lead from t_1 to t_2 (this number may also be 0). To be somewhat more formal: $t_1 \twoheadrightarrow t_2$ holds when this is derivable from the relation → by:
iii. $t_1 \twoheadrightarrow t_2$ if $t_1 \rightarrow t_2$;
iv. (reflexivity) $t \twoheadrightarrow t$;
v. (transitivity) if $t_1 \twoheadrightarrow t_2$ and $t_2 \twoheadrightarrow t_3$ then also $t_1 \twoheadrightarrow t_3$;
So in example 1.4.1 we have a(s(s(0)),s(s(0))) ↠ s(s(s(s(0)))).

1.4.4 DEFINITION
Let (Σ,R) be a TRS.
i. A term t is a **normal form**, or is a term **in normal form**, if there is no term s with $t \rightarrow s$.
ii. A term t **has a normal form** if there is a term s *in* normal form such that $t \twoheadrightarrow s$.

1.4.5 PROPERTIES OF A TRS

It is very desirable for a TRS that every (closed) term has a *unique normal form*. For example, for the TRS in table 4 (in 1.4.1) we can prove (not without difficulty!) that every closed term has a unique normal form of the form $s^n(0)$ (with $n \geq 0$).

Often, such a proof consists of proving that the following two properties are satisfied by a TRS:

i. **strong normalization**: there is no infinite sequence of reductions

$$t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow \ldots.$$

ii. **confluence** (also called the *Church-Rosser* or *diamond* property): if we have for terms t, t_1, t_2 that $t \twoheadrightarrow t_1$ and $t \twoheadrightarrow t_2$, then we can find a term t_3 such that $t_1 \twoheadrightarrow t_3$ and $t_2 \twoheadrightarrow t_3$ (i.e. we can complete the diamond in fig. 2).

Usually, we are only interested in these properties for closed (or ground) terms, and then we talk about **ground normalization** and **ground confluence**.

In order to prove confluence for a TRS, we need to look at all so-called *critical pairs*, i.e. the left-hand sides of two reduction rules that have overlap and thus can be applied to the same term.

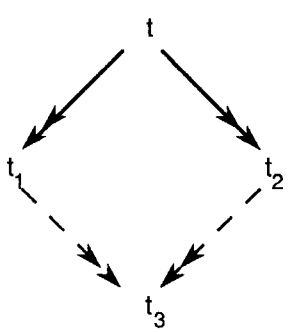

FIGURE 2.

1.4.6 THEOREM

Let (Σ,R) be a TRS. If (Σ,R) is strongly normalizing and confluent, then every term has a unique normal form. If (Σ,R) is ground normalizing and ground confluent, then every closed term has a unique normal form.

1.4.7 NOTE

We can always turn a TRS into an equational specification, by replacing every $\rightarrow$ in the reduction rules by the symbol =. If the TRS is strongly normalizing and confluent, that gives us a decision procedure to find out if two terms are equal in the resulting specification: reduce both terms to normal form; if they are identical, then the terms are equal, if they are not, then the terms are not equal. In this way we obtain an operational (i.e. executable) notion of derivability in a specification. Unfortunately, we cannot give such an operational notion for every specification, since equality is not always decidable.

1.4.8 EXERCISES

1. Prove for the TRS in table 4 that

$$m(a(s(s(0)),s(0)),s(s(0))) \twoheadrightarrow s(s(s(s(s(s(0)))))).$$

2. Prove that the closed normal forms for the TRS in table 4 are the terms of the form $s^n(0)$ (with $n \geq 0$).

3. We consider a TRS with a unary function $\neg$, and binary functions $\wedge$, $\vee$. The reduction rules are given in table 5 (in infix notation, leaving out many brackets).

$\neg\neg x \rightarrow x$
$\neg(x \vee y) \rightarrow \neg x \wedge \neg y$
$\neg(x \wedge y) \rightarrow \neg x \vee \neg y$
$x \vee (y \wedge z) \rightarrow (x \vee y) \wedge (x \vee z)$
$(x \wedge y) \vee z \rightarrow (x \vee z) \wedge (y \vee z)$

TABLE 5.

Reduce the term $\neg(x \wedge \neg y) \wedge z$ to normal form. (Note: the normal forms of this TRS are known in propositional logic as *conjunctive normal forms*.)

4. Prove that the TRS with the one rule

$$f(f(x)) \rightarrow f(g(f(x)))$$

is strongly normalizing.

5. Is the TRS with the one rule

$$f(g(x,y)) \rightarrow g(g(f(f(x)),y),y)$$

strongly normalizing?