1

Process algebra

1.1 Definition

This book is about *process algebra*. The term 'process algebra' refers to a loosely defined field of study, but it also has a more precise, technical meaning. The latter is considered first, as a basis to delineate the field of process algebra.

Consider the word 'process'. It refers to *behavior* of a *system*. A system is anything showing behavior, in particular the execution of a software system, the actions of a machine, or even the actions of a human being. Behavior is the total of events or actions that a system can perform, the order in which they can be executed and maybe other aspects of this execution such as timing or probabilities. Always, the focus is on certain aspects of behavior, disregarding other aspects, so an abstraction or idealization of the 'real' behavior is considered. Rather, it can be said that there is an *observation* of behavior, and an action is the chosen unit of observation. Usually, the actions are thought to be discrete: occurrence is at some moment in time, and different actions can be distinguished in time. This is why a process is sometimes also called a *discrete event system*.

The term 'algebra' refers to the fact that the approach taken to reason about behavior is algebraic and axiomatic. That is, operations on processes are defined, and their equational laws are investigated. In other words, methods and techniques of universal algebra are used (see e.g., (MacLane & Birkhoff, 1967)). To allow for a comparison, consider the definition of a *group* in universal algebra.

Definition 1.1.1 (Group) A *group* is a structure $(G, *, ^{-1}, u)$, with G the universe of elements, binary operator $*$ on G, unary operator $^{-1}$, and constant $u \in G$. For any $a, b, c \in G$, the following laws, or axioms, hold:

- $a * (b * c) = (a * b) * c$;

- $u * a = a = a * u$;
- $a * a^{-1} = a^{-1} * a = u$.

So, a group is any mathematical structure consisting of a single universe of elements, with operators on this universe of elements that satisfy the group axioms. Stated differently, a group is any *model* of the *equational theory of groups*. Likewise, it is possible to define operations on the universe of processes. A *process algebra* is then any mathematical structure satisfying the axioms given for the defined operators, and a process is then an element of the universe of this process algebra. The axioms allow *calculations* with processes, often referred to as *equational reasoning*.

Process algebra thus has its roots in universal algebra. The field of study nowadays referred to as process algebra, however, often, goes beyond the strict bounds of universal algebra. Sometimes the restriction to a single universe of elements is relaxed and different types of elements, different sorts, are used, and sometimes binding operators are considered. Also this book goes sometimes beyond the bounds of universal algebra.

The simplest model of system behavior is to see behavior as an input/output function. A value or input is given at the beginning of a process, and at some moment there is a(nother) value as outcome or output. This behavioral model was used to advantage as the simplest model of the behavior of a computer program in computer science, from the start of the subject in the middle of the twentieth century. It was instrumental in the development of (finite-state) *automata theory*. In automata theory, a process is modeled as an automaton. An automaton has a number of *states* and a number of *transitions*, going from state to state. A transition denotes the execution of an (elementary) action, the basic unit of behavior. Besides, an automaton has an initial state (sometimes, more than one) and a number of final states. A behavior is a run, i.e., an execution path of actions that lead from the initial state to a final state. Given this basic behavioral abstraction, an important aspect is when to consider two automata equal, expressed by a notion of equivalence, the *semantic equivalence*. On automata, the basic notion of semantic equivalence is language equivalence: an automaton is characterized by the set of runs, and two automata are equal when they have the same set of runs. An algebra that allows equational reasoning about automata is the algebra of regular expressions (see e.g., (Linz, 2001)).

Later on, the automata model was found to be lacking in certain situations. Basically, what is missing is the notion of *interaction*: during the execution from initial state to final state, a system may interact with another system. This is needed in order to describe parallel or distributed systems, or so-called

reactive systems. When dealing with models of and reasoning about interacting systems, the phrase *concurrency theory* is used. Concurrency theory is the theory of interacting, parallel and/or distributed systems. Process algebra is usually considered to be an approach to concurrency theory, so a process algebra will usually (but not necessarily) have *parallel composition* as a basic operator. In this context, automata are mostly called *transition systems*. The notion of equivalence studied is usually not language equivalence. Prominent among the equivalences studied is the notion of *bisimilarity*, which considers two transition systems equal if and only if they can mimic each other's behavior in any state they may reach.

Thus, a usable definition of *the field of process algebra* is the field that studies the behavior of parallel or distributed systems by algebraic means. It offers means to describe or *specify* such systems, and thus it has means to talk about parallel composition. Besides this, it can usually also talk about alternative composition (choice between alternatives) and sequential composition (sequencing). Moreover, it is possible to reason about such systems using algebra, i.e., equational reasoning. By means of this equational reasoning, *verification* becomes possible, i.e., it can be established that a system satisfies a certain property. Often, the study of transition systems, ways to define them, and equivalences on them are also considered part of process algebra, even when no equational theory is present.

1.2 Calculation

Systems with distributed or parallel, interacting components abound in modern life: mobile phones, personal computers interacting across networks (like the web), and machines with embedded software interacting with the environment or users are but a few examples. In our mind, or with the use of natural language, it is very difficult to describe these systems exactly, and to keep track of all possible executions. A formalism to describe such systems precisely, allowing reasoning about such systems, is very useful. Process algebra is such a formalism.

It is already very useful to have a formalism to describe, to specify interacting systems, e.g., to have a compact term specifying a communication protocol. It is even more useful to be able to reason about interacting systems, to verify properties of such systems. Such verification is possible on transition systems: there are automated methods, called *model checking* (see e.g., (Clarke *et al.*, 2000)), that traverse all states of a transition system and check that a certain property is true in each state. The drawback is that transition systems grow very large very quickly, often even becoming infinite. For instance, a system

having 10 interacting components, each of which has 10 states, has a total number of 10 000 000 000 states. It is said that model-checking techniques suffer from the *state-explosion problem*. At the other end, reasoning can take place in logic, using a form of deduction. Also here, progress is made, and many *theorem-proving tools* exist (see e.g., (Bundy, 1999)). The drawback here is that finding a proof needs user assistance, as the general problem is undecidable, and this necessitates a lot of knowledge about the system.

Equational reasoning on the basis of an algebraic theory takes the middle ground, in an attempt to combine the strengths of both model checking and theorem proving. Usually, the next step in the procedure is clear. In that sense, it is more rewriting than equational reasoning. Consequently, automation, which is the main strength of model checking, can be done straightforwardly. On the other hand, representations are compact and allow the presence of parameters, so that an infinite set of instances can be verified at the same time, which are strong points of theorem proving.

As an example, Chapter 8 presents a complete verification of a simple communication protocol: it is verified that the external behavior of the protocol coincides with the behavior of a one-place buffer. This is the desired result, because it proves that every message sent arrives at the receiving end.

1.3 History

Process algebra started in the 1970s. At that point, the only part of concurrency theory that existed was the theory of Petri nets, conceived by Petri starting from his thesis in 1962 (Petri, 1962). In 1970, three main styles of formal reasoning about computer programs could be distinguished, focusing on giving semantics (meaning) to programming languages.

(i) *Operational semantics*: A computer program is modeled as an execution of an abstract machine. A state of such a machine is a valuation of variables; a transition between states is an elementary program instruction. The pioneer of this field is McCarthy (McCarthy, 1963).

(ii) *Denotational semantics*: In a denotational semantics, which is typically more abstract than an operational semantics, computer programs are usually modeled by a function transforming input into output. The most well-known pioneers are Scott and Strachey (Scott & Strachey, 1971).

(iii) *Axiomatic semantics*: An axiomatic semantics emphasizes proof methods proving programs correct. Central notions are program assertions, proof triples consisting of precondition, program statement,

and postcondition, and invariants. Pioneers are Floyd (Floyd, 1967) and Hoare (Hoare, 1969).

Then, the question was raised how to give semantics to programs containing a parallel-composition operator. It was found that this is difficult using the methods of denotational, operational, or axiomatic semantics as they existed at that time, although several attempts were made. (Later on, it became clear how to extend the different types of semantics to parallel programming, see e.g., (Owicki & Gries, 1976) or (Plotkin, 1976).) Process algebra developed as an answer to this question.

There are two paradigm shifts that need to be made before a theory of parallel programs in terms of a process algebra can be developed. First of all, the idea of a behavior as an input/output function needs to be abandoned. The relation between input and output is more complicated and may involve *nondeterminism*. This is because the interactions a process has between input and output may influence the outcome, disrupting functional behavior. A program can still be modeled as an automaton, but the notion of language equivalence is no longer appropriate. Secondly, the notion of *global* variables needs to be overcome. Using global variables, a state of a modeling automaton is given as a valuation of the program variables, that is, a state is determined by the values of the variables. The independent execution of parallel processes makes it difficult or impossible to determine the values of global variables at any given moment. It turns out to be simpler to let each process have its own local variables, and to denote exchange of information explicitly via *message passing*.

Bekič

One of the first people studying the semantics of parallel programs was Hans Bekič. He was born in 1936, and died due to a mountain accident in 1982. In the early seventies, he worked at the IBM lab in Vienna, Austria. The lab was well-known in the sixties and seventies for its work on the definition and semantics of programming languages, and Bekič played a part in this, working on the denotational semantics of ALGOL and PL/I. Growing out of his work on PL/I, the problem arose how to give a denotational semantics for parallel composition. Bekič tackled this problem in (Bekič, 1971). This internal report, and indeed all the work of Bekič, is made accessible through the book edited by Cliff Jones (Bekič, 1984). The following remarks are based on this book.

In (Bekič, 1971), Bekič addresses the semantics of what he calls ‘quasi-parallel execution of processes’. From the introduction:

> Our plan to develop an *algebra of processes* may be viewed as a *high-level* approach: we are interested in how to compose complex processes from simpler (still arbitrarily complex) ones.

Bekič uses global variables, so a state is a valuation of variables, and a program determines an action, which gives in a state (non-deterministically) either *null* if and only if it is an end-state, or an elementary step, giving a new state and rest-action. Further, Bekič has operators for alternative composition, sequential composition, and (quasi-)parallel composition. He gives a law for quasi-parallel composition, called the 'unspecified merging' of the elementary steps of two processes. That law is definitely a precursor of what later would be called the expansion law of process algebra. It also makes explicit that Bekič has made the first paradigm shift: the next step in a merge is not determined, so the idea of a program as a function has been abandoned.

Concluding, Bekič contributed a number of basic ingredients to the emergence of process algebra, but he does not yet provide a coherent comprehensive theory.

CCS

The central person in the history of process algebra without a doubt is Robin Milner. A.J.R.G. Milner, born in 1934, developed his process theory CCS, the *Calculus of Communicating Systems*, over the years 1973 to 1980, culminating in the publication of the book (Milner, 1980) in 1980.

Milner's oldest publications concerning the semantics of parallel composition are (Milner, 1973; Milner, 1975), formulated within the framework of denotational semantics, using so-called transducers. He considers the problems caused by non-terminating programs, with side effects, and non-determinism. He uses operators for sequential composition, for alternative composition, and for parallel composition. He refers to (Bekič, 1971) as related work.

Next, in terms of the development of CCS, are the articles (Milner, 1979) and (Milne & Milner, 1979). In that work, Milner introduces *flow graphs*, with ports, where a named port synchronizes with the port with its co-name. Operators are parallel composition, restriction (to prevent certain specified actions), and relabeling (for renaming ports). Some laws are stated for these operators.

The two papers that put in place most of CCS as it is known to date, (Milner, 1978a) and (Milner, 1978b), conceptually built upon this work, but appeared in 1978. The operators prefixing and alternative composition are added and provided with laws. Synchronization trees are used as a model. The prefix τ occurs as a *communication trace*, i.e., what remains of a synchronization of a name and a co-name. Such a remains is typically unobservable, and later,

τ developed into what is now usually called the silent step. The paradigm of message passing, the second paradigm shift, is taken over from (Hoare, 1978). Interleaving is introduced as the observation of a single observer of a communicating system, and the expansion law is stated. Sequential composition is not a basic operator, but a derived one, using communication, abstraction, and restriction.

The paper (Hennessy & Milner, 1980), with Matthew Hennessy, formulates basic CCS, with two important semantic equivalence relations, observational equivalence and strong equivalence, defined inductively. Also, so-called Hennessy-Milner logic is introduced, which provides a logical characterization of process equivalence. Next, the book (Milner, 1980) was published, which is by now a standard process algebra reference. For the first time in history, the book presents a complete process algebra, with a set of equations and a semantic model. In fact, Milner talks about *process calculus* everywhere in his work, emphasizing the calculational aspect. He presents the equational laws as truths about his chosen semantic domain, transition systems, rather than considering the laws as primary, and investigating the range of models that they have. The book (Milner, 1980) was later updated in (Milner, 1989).

CSP

A very important contributor to the development of process algebra is Tony Hoare. C.A.R. Hoare, born in 1934, published his influential paper (Hoare, 1978) as a technical report in 1976. The important step is that he does away completely with global variables, and adopts the message-passing paradigm of communication, thus realizing the second paradigm shift. The language CSP, *Communicating Sequential Processes*, described in (Hoare, 1978) has synchronous communication and is a guarded-command language (based on (Dijkstra, 1975)). No model or semantics is provided. This paper inspired Milner to treat message passing in CCS in the same way.

A model for CSP was elaborated in (Hoare, 1980). This is a model based on trace theory, i.e., on the sequences of actions a process can perform. Later on, it was found that this model was lacking, for instance because deadlock behavior is not preserved. For this reason, a new model based on so-called failure pairs was presented in (Brookes *et al.*, 1984), for the language that was then called TCSP, *Theoretical* CSP. Later, TCSP was called CSP again. In the language, due to the less discriminating semantics when compared to the equivalence adopted by Milner and the presence of two alternative composition operators, it is possible to do without a silent step like τ altogether. The book (Hoare, 1985) gives a good overview of CSP.

Between CCS and CSP, there is some debate concerning the nature of alternative composition. Some say the $+$ of CCS is difficult to understand (exemplified by the philosophical discussion on 'the weather of Milner'), and CSP proposes to distinguish between internal and external non-determinism, using two separate operators; see also (Hennessy, 1988a).

Some other process theories

Around 1980, concurrency theory and in particular process theory is a vibrant field with a lot of activity world wide. There is research on Petri nets, partially ordered traces, and temporal logic, among others. Other process theories are trace theory and the invariants calculus. In particular, there is the metric approach by De Bakker and Zucker (De Bakker & Zucker, 1982a; De Bakker & Zucker, 1982b). It has a notion of distance between processes: processes that do not differ in behavior before the n-th step have a distance of at most 2^{-n}. This turns the domain of processes into a metric space, that can be completed. Recursive equations allow to specify unbounded process behavior. In the metric approach by De Bakker and Zucker, solutions to an important class of recursive equations, so-called *guarded* recursive equations, exist by application of Banach's fixed point theorem. This result later influenced the development of process algebra, in particular the development of ACP.

ACP

Jan Bergstra and Jan Willem Klop started to work in 1982 on a question of De Bakker's as to what can be said about solutions of *unguarded* recursive equations. As a result, they wrote the paper (Bergstra & Klop, 1982). In this paper, the phrase 'process algebra' is used for the first time, with exactly the two meanings given in the first part of this chapter. The paper defines a process algebra with alternative, sequential, and parallel composition, but without communication. A model was established based on projective sequences, meaning that a process is given by a sequence of approximations by finite terms, and in this model, it is established that all recursive equations, both guarded and unguarded, have a solution. In adapted form, this paper was later published as (Bergstra & Klop, 1992). In (Bergstra & Klop, 1984a), this process algebra, called PA, for Process Algebra, was extended with communication to yield the theory ACP, the *Algebra of Communicating Processes*. Textbooks on ACP are (Baeten & Weijland, 1990; Fokkink, 2000).

Comparing the three most well-known process algebras to date, CCS, CSP, and ACP, it can be concluded that there is a considerable amount of work and

applications realized in all three of them. In that sense, there seem to be no fundamental differences between the theories with respect to the range of applications. Historically, CCS was the first with a complete theory. Compared to the other two, CSP has the least distinguishing equational theory. More than the other two, ACP emphasizes the algebraic aspect: there is an equational theory with a range of semantic models. Also, ACP has the most general communication scheme: in CCS, communication is combined with abstraction, and also CSP has a restricted communication scheme.

Further developments

The development of CCS, CSP, and ACP was followed by the development of other process algebras, such as SCCS (Milner, 1983), CIRCAL (Milne, 1983), MEIJE (Austry & Boudol, 1984), and the process algebra of Hennessy (Hennessy, 1988a). Moreover, many process algebras were extended with extra features, such as timing or probabilities. A number of these extensions are also addressed in this book.

Over the years, many process algebras have been developed, each making its own set of choices in the different possibilities. The reader may wonder whether this is something to be lamented. In (Baeten *et al.*, 1991), it is argued that this is actually a good thing, as long as there is a good exchange of information between the different research groups, as each different process algebra has its own set of advantages and disadvantages. The theoretical framework developed in this book is generic, in the sense that most features found in other process algebras can be defined in it. Throughout the book, it is indicated how this can be achieved.

This book

This book follows the ACP approach in its emphasis on algebra. The main difference with the theory set out in (Bergstra & Klop, 1984a; Baeten & Weijland, 1990) is that successful termination is integrated in the theory almost from the beginning. As set out in (Baeten, 2003), this leads to some other changes in the theory. The basic theory starts with a prefixing operator as in CCS and CSP, and adds the sequential-composition operator, which is a basic operator in ACP, in a later chapter.

This book arose as a complete and thorough revision of the book (Baeten & Weijland, 1990). Although many changes have occurred, the approach and methodology remain the same. Also some parts of the text have remained almost unchanged. The book has been updated in many places to reflect the

latest developments, making it the most complete and in-depth account of the state-of-the-art in process algebra at the time of writing.

Bibliographical remark

The historic overview of this section first appeared in (Baeten, 2005).