Book Introduction by the Authors

Invited by

Luca Aceto

luca.aceto@gmail.com
President of EATCS
Reykjavik University, Reykjavik, Iceland

BOOK ANNOUNCEMENT:
Modeling and Analysis of Communicating Systems

Jan Friso Groote[1] and Mohammad Reza Mousavi[2]

[1]Department of Computer Science, Eindhoven University of Technology, The Netherlands

[2]Center for Research on Embedded Systems, Halmstad University, Sweden

1 Motivation

1.1 Subject Matter

Computer systems are becoming increasingly complex parallel, distributed, and communicating devices. Unfortunately, human beings are notoriously weak in thinking of and reasoning about concurrency scenarios [5]. Therefore, mathematical modeling and analysis techniques are required to comprehend and design them. This situation is similar to classical engineering disciplines, where mathematical comprehension is already for a long time the essential tool to engineer artifacts.

In “Modeling and Analysis of Communicating Systems" [8], we put forward a method for rigorous modeling and mathematical analysis of communicating and concurrent systems. The primary focus of the book is to present formalisms, methods and tools that are helpful in understanding and designing the complex concurrent systems that surround us.

This book takes process algebras and in particular the notion of an atomic action as a starting point. This basic formalism has been enriched with various datatypes, including functions, sets, lists, reals, and quantifiers. In order to formulate requirements, the modal μ-calculus is used, and is extended with data and time, which as such is second to none when it comes to expressiveness.

Most of the theory in the book is presented as classical theory that can be studied, understood and used by humans. But if it comes to the study and analysis of actual behavior, this is generally so complex, that manually analyzing them is not very effective. Therefore, the book is supported by an extensive and powerful toolset, available from `http://www.mcrl2.org`.

On the toolset webpage, it can be seen that the method has been used to under-

stand, design and improve a whole range of systems, ranging from the software of the Alma telescopes in Chili, the immense sensor control systems of the particle accelerator of CERN in Switzerland, to the software in the cruise class solar car Stella, which won the 2013 solar competition in Australia.

1.2 Distinguishing Features and Alternatives

This is certainly not the only textbook about modeling and analysis of concurrent systems. We mention a few of the best alternatives to our book and briefly comment on how our book differs from them.

The textbooks by Baeten et al. [2], Aceto et al. [1], Roscoe [12] and Fokkink [7] all provide excellent introductions to the theoretical foundations of the modeling approach treated in our book. The recent book by Baier and Katoen [3], on the other hand, provides an in-depth yet very accessible treatment of an analysis technique advocated in this book, namely model checking.

The above-mentioned alternatives take a more theoretical approach in the presentation of similar formalisms and/or analysis techniques. However, our emphasis is to prepare to use the formalism in modeling and reasoning of actual systems. The courses offered based on the present book have always been accompanied with practical projects in which the students apply the learned techniques in modeling (somewhat simplified) industrial systems in the area of their respective program.

1.3 Organization of This Article

In this short article, we give an overview of the three parts of the book: modeling, analysis and semantics in sections 2, 3, and 4 below, respectively. We conclude in section 5, by reporting about our method and experience in using this book in various courses.

2 Modeling

2.1 Behavioral Modeling

The first part of the book is about rigorous modeling. The modeling approach advocated in this book is "behavioral modeling", which is introduced in Chapter 2. Behavioral modeling focuses on specifying the interaction of the component or the system under study at its interface. This approach is inspired by the seminal work of pioneers such as Petri [11], Bekič [4], Milner [10], and Hoare [9]. In

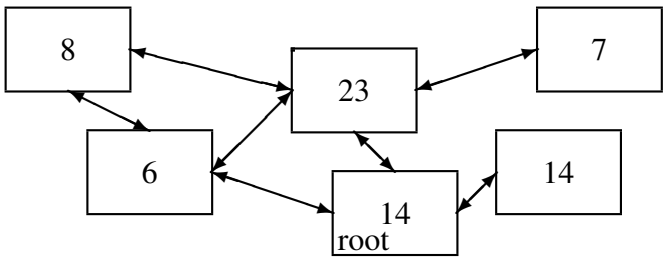

Figure 1: A set of distributed processes

order to reduce such specifications or compare them with their implementations, different notions of behavioral equivalence are introduced in the same chapter.

Subsequently, an extensible language of abstract data types is introduced in Chapter 3, which allows for parameterizing the interactions and specifying the content of the messages being communicated at any desired level of abstraction. The language of abstract data type offers built-in data types such as Booleans, integers, and rational numbers with their typical operations, as well as sort constructors, e.g., for structured (enumerated) data types, lists, sets, bags and function sorts. One can also define arbitrary algebraic data types by specifying the constructors and the operators on the sort.

Chapters 4 and 5 treat the process language. To give an idea about the kind of specifications allowed in the process language and its associated data type language, consider the distributed summing protocol depicted in figure 1. It consists of an ad-hoc network in which each node contains a number. The nodes must communicate in such a way that all numbers in the nodes are added up after which the root reports this sum.

The processes of the network interact via matching actions st, $\overline{st}$ (for *start*), ans, $\overline{ans}$ (for *answer*) and the total sum is communicated using a $\overline{rep}$ (for *report*) action. We think of the overbarred action as a sending activity, and plain (not overbarred) action as a receiving action. The process X is described by means of six parameters:

- i: the ID number of the process.
- t: the *total sum* computed so far by the process. Initially, it contains the value that is the contribution of process i to the total sum.
- N: a list of *neighbors* to which the process still needs to send an $\overline{st}$ message. Initially, this list contains exactly all neighbors. We write $rem(j, N)$ to remove neighbor j from list N.
- p: the index of the initiator, or *parent*, of the process. Variable p is also called the parent link of i.

$$\begin{aligned}
&\textbf{proc}\, X(i{:}\mathbb{N}^+, t{:}\mathbb{N}, N{:}\mathit{List}(\mathbb{N}), p, w, s{:}\mathbb{N}) = \\
&\qquad (s{\approx}0) \rightarrow \textstyle\sum_{j:\mathbb{N}} \mathit{st}(i,j){\cdot}X(i,t,\mathit{rem}(j,N),j,\#(N){-}1,1){+}\\
&\qquad \textstyle\sum_{j:\mathbb{N}}(j{\in}N \wedge s{\approx}1) \rightarrow \overline{\mathit{st}}(j,i){\cdot}X(i,t,\mathit{rem}(j,N),p,w,s){+}\\
&\qquad \textstyle\sum_{j,m:\mathbb{N}}(s{\approx}1) \rightarrow \mathit{ans}(i,j,m){\cdot}X(i,t+m,N,p,w{-}1,s){+}\\
&\qquad \textstyle\sum_{j:\mathbb{N}}(s{\approx}1) \rightarrow \mathit{st}(i,j){\cdot}X(i,t,N,p,w{-}1,s){+}\\
&\qquad (i{\approx}1 \wedge N{\approx}[] \wedge w{\approx}0 \wedge s{\approx}1) \rightarrow \overline{\mathit{rep}}(t){\cdot}X(i,t,N,p,w,2){+}\\
&\qquad (i{\not\approx}1 \wedge N{\approx}[] \wedge w{\approx}0 \wedge s{\approx}1) \rightarrow \overline{\mathit{ans}}(p,i,t){\cdot}X(i,t,N,p,w,2);
\end{aligned}$$

Table 1: The behavior of a parameterized node X in the distributed summing protocol

- w: The number of *st* and *ans* messages that the process is still *waiting* for.
- s: the *state* the process is in. The process can be in three states, denoted by 0, 1, and 2. If s equals 0, the process is in its initial state. If s equals 1, the process is active. If s equals 2, the process has finished.

The specification of a parameterized node X in the distributed summing protocol is given in table 1. This specification has the shape of a recursive equation, with a single variable at the left and the definition of a 'body' at the right. In the first line of the definition of X, process i is in its initial state ($s{\approx}0$) and an *st* message is received from some process j, upon which j is stored as the parent and s switches from 0 to 1, indicating that process i has become active. Since it makes no sense to send start messages to one's parent, j is removed from N. The counter w is initialized to the number of neighbors of i, not counting process j. In line 2, a message is sent to a neighbor j, which is thereupon removed from N. In line 3, a sum is received from some process j via an *ans* message containing the value m, which is added to t, the total sum computed by process i so far. The counter w is decreased. In line 4 an *st* message is received from neighbor j. The message is ignored, except that the counter w is decreased. In line 5 a $\overline{\mathit{rep}}(t)$ is sent (in case $i = 1$), when process 1 is active, there are no more *ans* or *st* messages to be received (formalized by the condition $w = 0$), and an $\overline{\mathit{st}}$ message has been sent to all neighbors (formalized by the condition $N{\approx}[]$). The status variable s becomes 2, indicating that process 1 is no longer active. Line 6 is as line 5 but for processes $i \neq 1$. Now an $\overline{\mathit{ans}}$ message is sent to parent p, containing the total sum t computed by process i.

The system comprises the parallel composition of n copies of the process X with distinct identifiers. The composition of these processes first enforces synchronization on send and receive actions (resulting in the same actions postfixed with an asterisk, e.g., $\mathit{ans}^*(i,j,m)$ and $\mathit{st}^*(i,j)$) and subsequently hides the result of the synchronization. We refer to Chapters 7 and 13 of the book for the com-

plete specification of this example as well as several other distributed and parallel systems and protocols.

2.2 Correctness Properties

A logical language based on an extension of the modal μ-calculus is defined in Chapter 6 for specifying correctness properties of systems. This extension allows a.o. for data in the modal formulae, both as arguments of actions and as parameters/arguments of fixed point variables. This extension allows for very succinct encoding of existing temporal logics such as LTL, CTL and CTL* [6].

Here are three typical examples of modal μ-calculus formulae that can be verified on the distributed summing protocol (before hiding the result of the synchronizations among components):

$$\nu X.\forall t{:}\mathbb{N}.[!\overline{rep(t)}]X \wedge \langle true\rangle true$$

$$[true^{\star}\cdot(\exists t{:}\mathbb{N}.\overline{rep(t)})\cdot true^{\star}\cdot(\exists t{:}\mathbb{N}.\overline{rep(t)})]false$$

$$\begin{array}{ll} \nu Y(n{:}\mathbb{N}{:=}0). & \forall i,j,t{:}\mathbb{N}.[ans^*(i,j,t)]Y(t)\wedge \\ & \forall i,j,t{:}\mathbb{N}.[!ans^*(i,j,t)]Y(n)\wedge \\ & \forall t{:}\mathbb{N}.[\overline{rep}(t)](t \geq n) \end{array}$$

The first formula given above, states that the protocol will not reach a deadlock state before sending an outcome of the sum through the action $\overline{rep(t)}$. The second formula specifies that the outcome of the sum can be announced at most once. Finally, the third formula states that the reported sum is not less than the partial sums announced through $ans^*(i, j, t)$ actions.

2.3 Timed Behavior

The underlying language also supports timed actions and timed processes. However, to make the presentation more accessible time and timed specification are treated separately in Chapter 8. If the axioms of timed systems are different from their untimed variants, this is briefly indicated in the earlier chapters. In addition to the extension of the specification language with time, the timed extensions of various notions of behavioral equivalence as well as the timing extensions to the modal μ-calculus with data are presented in this chapter.

The theory offers a full set of axioms to manipulate timed processes, and methods to determine whether timed modal formulas are valid on processes. Although, there are a number of experimental tools that allow verification of timed systems, these are not yet available in the standard distribution of the toolset.

3 Analysis

3.1 Axioms and transformation rules

The book offers numerous process algebraic axioms and equalities on modal formulae. But these alone are not sufficient to verify actual processes. Hence, a wealth of methods and techniques have been provided to prove behavioral specifications equal, and to show that specifications satisfy modal requirements.

3.2 Linear Processes Equations

The workhorse for all verifications is the elimination of parallelism from processes by transforming processes into linear process equations. The book sketches how this is done using the so-called Greibach Normal Form. The principles of these methods are explained in Chapters 9 and 10.

When processes are linearized, they are far easier to manipulate. Notions such as invariants, which are tricky to define for arbitrary processes, can very naturally be defined on linear processes.

Linear processes are so effective that they form the heart of the toolset. Every process is first linearized, before any other operation can be applied to it. Before generating a state space, it is often wise to investigate and transform a linear process, because this easily leads to a substantial reduction of the state space. For instance by (automatically) proving a linear process τ-confluent, which is the prerequisite for a form of partial order reduction, exponential reductions of the generated labeled transition systems can be obtained.

The book contains a theorem, explaining how arbitrary constellations of linear processes can be combined into a single linear process. For example, the distributed summing protocol specified in section 2.1 is translated using this theorem in the description in table 2. Without going into detail about the particulars of this formulation, the expression in table 2 encompasses the complete distributed summing protocol for an arbitrary number of processes using an arbitrary ad hoc network for communication.

3.3 Cones and foci proof technique

The book contains a chapter on a very effective proof technique to show that an implementation is equal to a specification. This technique is called the cones and foci technique and it is presented in chapter 12. In the case of the distributed summing protocol, it is used to show that under certain sanity conditions, the linear process equation shown in table 2 is equivalent to the following linear process:

$$
\begin{aligned}
&\textit{L-Impl}(n{:}\mathbb{N}^+, \boldsymbol{t}{:}\mathbb{N}^+{\to}\mathbb{N}, \boldsymbol{n}{:}\mathbb{N}^+{\to}\textit{List}(\mathbb{N}), \boldsymbol{p}, \boldsymbol{w}, \boldsymbol{s}{:}\mathbb{N}^+{\to}\mathbb{N}) = \\
&\quad (\boldsymbol{n}(1){\approx}[] \wedge \boldsymbol{w}(1){\approx}1 \wedge \boldsymbol{s}(1){\approx}1) \to \overline{\textit{rep}}(\boldsymbol{t}(1)){\cdot}\textit{L-Impl}(\boldsymbol{s}{=}\boldsymbol{s}[1{\to}2]){+} \\
&\quad \textstyle\sum_{i,j:\mathbb{N}^+}(\boldsymbol{s}(i){\approx}1 \wedge i \in \boldsymbol{n}(j) \wedge \boldsymbol{s}(j){\approx}1 \wedge i{\not\approx}j \wedge i{\leq}n \wedge j{\leq}n) \to \\
&\qquad \textit{int}{\cdot}\textit{L-Impl}(\boldsymbol{n}{=}\boldsymbol{n}[j{\to}\textit{rem}(i, \boldsymbol{n}(j)), i{=}\textit{rem}(j, \boldsymbol{n}(i))], \\
&\qquad\qquad \boldsymbol{p}{=}\boldsymbol{p}[i{\to}j], \boldsymbol{w}{=}\boldsymbol{w}[i{\to}\#\boldsymbol{n}(i){-}1], \boldsymbol{s}{=}\boldsymbol{s}[i{\to}1]){+} \\
&\quad \textstyle\sum_{i,j:\mathbb{N}^+}(\boldsymbol{s}(i){\approx}1 \wedge i \in \boldsymbol{n}(j) \wedge \boldsymbol{s}(j){\approx}1 \wedge i{\not\approx}j \wedge i{\leq}n \wedge j{\leq}n) \to \\
&\qquad \textit{int}{\cdot}\textit{L-Impl}(\boldsymbol{n}{=}\boldsymbol{n}[j{\to}\textit{rem}(i, \boldsymbol{n}(j))], \boldsymbol{w}{=}\boldsymbol{w}[i{\to}\boldsymbol{w}(i){-}1]){+} \\
&\quad \textstyle\sum_{j:\mathbb{N}^+}(\boldsymbol{n}(j){\approx}[] {\wedge} \boldsymbol{w}(j){\approx}0 {\wedge} \boldsymbol{s}(j){\approx}1 {\wedge} \boldsymbol{s}(\boldsymbol{p}(j)){\approx}1 {\wedge} j{\not\approx}1 {\wedge} j{\not\approx}\boldsymbol{p}(j) {\wedge} j{\leq}n {\wedge} \boldsymbol{p}(j){\leq}n){\to} \\
&\qquad \textit{int}{\cdot}\textit{L-Impl}(\boldsymbol{t}{=}\boldsymbol{t}[\boldsymbol{p}(j){\to}\boldsymbol{t}(\boldsymbol{p}(j)) + \boldsymbol{t}(j)], \boldsymbol{w}{=}\boldsymbol{w}[\boldsymbol{p}(j){\to}\boldsymbol{w}(\boldsymbol{p}(j)){-}1], \boldsymbol{s}{=}\boldsymbol{s}[j{\to}2]).
\end{aligned}
$$

Table 2: Linearization of the distributed summing protocol

$$
\textbf{proc}\,\textit{DSum}(n{:}\mathbb{N}^+, \boldsymbol{t}_0{:}\mathbb{N}^+{\to}\mathbb{N}, \boldsymbol{n}_0{:}\mathbb{N}^+{\to}\textit{List}(\mathbb{N}^+)) = \overline{\textit{rep}}(\textstyle\sum_{i=1}^{n} \boldsymbol{t}_0(i)){\cdot}\delta
$$

The above-given process sends a $\overline{rep}$ message with as the only parameter an integer that is the sum of the original numbers in the nodes, denoted by $\sum_{i=1}^{n} \boldsymbol{t}_0(i)$. After this, it turns into deadlock, denoted by δ. This process can be considered as an obviously correct specification of the distributed summing protocol and hence the equivalence proof provides clear evidence of correctness for our original specification presented in section 2.1. In chapter 13 of the book the cones and foci method is applied to various other distributed systems, including to the third sliding window protocol by Andrew Tanenbaum.

3.4 Model Checking

A method to verify the correctness of a model is to verify whether it satisfied the specified modal μ-calculus properties. In order to perform this type of verification, one can combine the linearized version of the model and its modal μ-calculus property into a Parameterized Boolean Equation System (PBES).

Solving such a PBES provides an answer to the verification problem, which is either positive, i.e., the model is correct with respect to the specified properties, or negative, in which case the modal requirement is not valid for the process. There are numerous ways to solve parameterized boolean equation systems, varying from well-known fixed point iteration, Gaussian elimination, to the recognition of particular patterns of PBESs for which known solutions exist. The structure of PBESs and the methods for solving them are explained in Chapter 14 of the book.

4 Semantics

The last chapter of the book is dedicated to the formal (mathematical) semantics of the various formalisms presented throughout the book. There is a type system to determine well-typedness of all formalisms. Data specifications are provided with a model-class semantics. Processes are characterized by a structural operational semantics. Modal μ-calculus formulae are provided with a typical logical fixed point semantics in terms of the states in transitions of a transition system. Parameterized Boolean Equation Systems are also characterized by a logical fixed point semantics.

4.1 Appendices

The book is provided with a number of appendices. These contain a primer for the use of the toolset, all defining equations for the built-in data types, a complete syntax of all formalisms used by the tools and answers to all exercises in the book.

5 Conclusions

We have used this textbook to teach various graduate level courses with students from very different backgrounds (such as computer science, electrical engineering and mechanical engineering). The students by and large appreciated the resulting courses and used the learned techniques successfully to specify cyber-physical systems in their domains. In order to allow the students with different backgrounds (than computer science) to develop a feeling about the concepts taught in these courses, we have cut on the number of subjects and chapters treated. For an easy 5 ects credit masters course, we teach the first 6 chapters of the book. For more advanced masters courses, chapters 8 to 12 and chapter 14 can additionally be included.

We always run the course in parallel with a practical project. In such a project students design a (sometimes slightly simplified) controller for an actual device. The student starts with informally specifying the behavioral properties of the system under design. They subsequently model the requirements as modal formulas, and the behavior of the controller as an mCRL2 specifications. Subsequently, they must prove all requirements to hold on the controller, often detecting and improving flaws, both in the controller and the requirements. The end result is an abstract high quality design of the controller, which is an excellent basis for an implementation.

Acknowledgments Many people contributed directly or indirectly to this book. We are thankful for all those that have been contributing to the field of concurrency, both in the form of tools and theory, for constantly pushing the technological barriers in this field forward. For finalizing the final drafts of this book, thanks go to Sjoerd Cranen, Veronica Gaspes, Jeroen Keiren, Michel Reniers, and Erik de Vink for their careful proofreading. We have received valuable feedback from various students and colleagues throughout the process of teaching and writing. Also the MIT press staff have provided valuable feedback and fantastic cooperation in the process of editing and publishing the book. In particular, we are thankful to Virginia Crossman, Marc H. Lowenthal, and Marie Lufkin Lee.

References

[1] L. Aceto, A. Ingolfsdottir, K.G. Larsen, and J. Srba. *Reactive systems: modelling, specification and verification*. Cambridge University Press, Cambridge, U.K., 2007.

[2] J.C.M. Baeten, T. Basten, and M.A. Reniers. *Process algebra: Equational theories of communicating processes*. Cambridge tracts in theoretical computer science, Vol. 50. Cambridge University Press, Cambridge, U.K., 2010.

[3] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, Cambridge, MA, 2008.

[4] H. Bekič. Towards a mathematical theory of processes. In *Programming languages and their definitions: H. Bekič (1936–1982)*. Lecture notes in computer science, Vol. 177 (pp. 156–167), Springer-Verlag, Berlin, Germany, 1984.

[5] M. Ben-Ari and Y. Ben-Davied Kolikant. Thinking Parallel: The Process of Learning Concurrency. Proceedings of the 4th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE 1999), pp. 21–24, ACM Press, 1999.

[6] S. Cranen, J.F. Groote, and M.A. Reniers. A linear translation from CTL* to the first-order modal μ-calculus. https://www.sharelatex.com/project/5444fff6c1f7e0511439397e Theoretical Computer Science, 412(28):3129–3139, 2011.

[7] W.J. Fokkink. *Modelling distributed systems*. Texts in theoretical computer science. Springer-Verlag, Berlin, Germany, 2007

[8] J.F. Groote and M.R. Mousavi. Modeling and Analysis of Communicating Systems. MIT Press, Boston, MA, USA, 2014.

[9] C.A.R. Hoare. *Communicating sequential processes*. Prentice Hall, Englewood Cliffs, NJ, 1985.

[10] R. Milner. *Communication and concurrency*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[11] C.A. Petri. *Kommunikation mit automaten*. Ph.D. thesis, Institut fuer Instrumentelle Mathematik, Bonn, Germany, 1962.

[12] A.W. Roscoe. *Understanding concurrent systems.* Springer-Verlag, Berlin, Germany, 2007.