Review of: **Modal and Temporal Properties of Processes** by Colin Stirling

Vicky Weissman
Department of Computer Science
Cornell University

Overview

At a very high level, this book presents languages for describing processes and properties of processes. It then discusses techniques for checking if a process, or family of processes, has a given property.

What is a process? In essence, a process is something that can perform actions. By doing actions, processes may change. Examples of processes include a clock that can tick and a counter with value zero that can do an increment action to become a counter with value one. The following syntax is used to describe processes where E, possibly subscripted, is a process, both a and b are actions, and I is an indexing set.

$$E ::= a.E \mid E[a/b] \mid \sum_{i \in I} E_i$$

A process of the form $a.E$ can do action a to become process E. A process of the form $E[a/b]$ is identical to the process E, except that every occurrence of action b is replaced by action a. A process of the form $\sum_{i \in I} E_i$ refers to some process E_j where $j \in I$.

We would like to extend the language to describe interactions between processes, but first we need to explain what we mean by process interaction. Two processes interact when one sends information and the other receives it. We formalize this notion, by creating pairs of actions. Each pair consists of two actions with the same name, except that one has an overhead bar and is associated with out-going communication, while the other does not have a bar and is associated with in-coming communication. For example, a process E_1 may tell a process E_2 that it's idle by doing an action $\overline{a}$. E_2 receives this information by doing the action a. We say that a is the *co-action* of $\bar{a}$ and vice-versa.

To capture interactions between processes, we add the syntax $E_1|E_2$ and $E\backslash J$ to the language where E_1, E_2, and E are processes and J is a set of actions. A process of the form $E_1|E_2$ is the concurrent composition of processes E_1 and E_2. In other words, if E_1 can do an action a to become E_1' and E_2 can do the co-action $\bar{a}$ to become E_2', then $E_1|E_2$ can do a to become $E_1'|E_2$, can do $\bar{a}$ to become $E_1|E_2'$, and can do the internal action τ to become $E_1'|E_2'$. (We always use the symbol τ to refer to an internal action, which is one process doing an action and another process doing the co-action. The set of actions may not include $\bar{\tau}$.) The restriction operator $\backslash$ is used to synchronize processes, by forbidding actions to be performed unless the co-action is also done. More specifically, a process of the form $E\backslash J$ is identical to the process E except that it cannot do any action that is in J and it cannot do any action whose co-action is in J. For example, $(E_1|E_2)\backslash a$ synchronizes processes E_1 and E_2 with respect to action a, by forbidding the processes to do a or $\bar{a}$. If one of the processes does a (or $\bar{a}$), then it cannot do another action until the other process has done $\bar{a}$ (or a) and, thus, $(E_1|E_2)\backslash a$ has done τ instead of a or $\bar{a}$.

We write $E_1 \stackrel{a}{\rightarrow} E_2$ to say that process E_1 can do action a to become process E_2. For example, $a.E \stackrel{a}{\rightarrow} E$ is always true and $\sum_{i \in I} E_i \stackrel{a}{\rightarrow} F$ is true if there is some $j \in I$ such that $E_j \stackrel{a}{\rightarrow} F$. We use the symbol $\Rightarrow$ to discuss *observable actions* where an observable action is any action other than τ. If a is an observable action, then $E_1 \stackrel{a}{\Rightarrow} E_2$ means that process E_1 can become process E_2 by doing a sequence of actions in which the only observable action is a. The notation can be extended in a straightforward way to allow an action sequence in place of a single action. For example, if a and b are actions, then $a.b.E \stackrel{ab}{\rightarrow} E$ is always true and, if ϵ is the empty sequence, then $E_1 \stackrel{\epsilon}{\Rightarrow} E_2$ is true if E_1 can become E_2 without doing any observable actions.

Given a process, we would like to know if it has certain properties. For example, we may want to know if the process is deadlocked or if it could become deadlocked without doing any observable action. To discover if a property holds for a process, we need a language for properties and we need a formal definition of what it means for a process to have a property. The syntax for a simple property language is given below where Φ, possibly with a subscript, is a property and K is a set of actions. As an aside, we refer to the set of all actions except those in an action set K as the set $-K$. Similarly, the set of all actions is $-$ (for $-\emptyset$).

$$\Phi ::= tt \mid ff \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [K]\Phi \mid \langle K \rangle \Phi$$

tt stands for true; it is the property that every process has. Similarly, ff stands for false; it is the property that no process has. Conjunction and disjunction have their standard meanings. A process has the property $[K]\Phi$ if, after doing any action in K, it becomes a process that has the property Φ. A process has the property $\langle K \rangle \Phi$ if, after doing some action in K, it can become a process that has the property Φ. Formally, we define the following satisfiability relation where $E \models \Phi$ means that process E has (satisfies) property Φ and $E \not\models \Phi$ means that it does not.

- $E \models tt$ and $E \not\models ff$
- $E \models \Phi_1 \wedge \Phi_2$ iff $E \models \Phi_1$ and $E \models \Phi_2$
- $E \models \Phi_1 \vee \Phi_2$ iff $E \models \Phi_1$ or $E \models \Phi_2$
- $E \models [K]\Phi$ iff $\forall F \in \{E' : E \stackrel{a}{\rightarrow} E' \text{ and } a \in K\}.F \models \Phi$
- $E \models \langle K \rangle \Phi$ iff $\exists F \in \{E' : E \stackrel{a}{\rightarrow} E' \text{ and } a \in K\}.F \models \Phi$

In this language, a process E is deadlocked if it has the property $[-]ff$. Unfortunately, it may not be clear if E has this property when E contains the restriction operator $\backslash$, the concurrency operator $|$, or renaming. To simplify our analysis, we note that formulas written in our language (properties) adhere to the following:

- If $a \notin K$ then $a.E \models [K]\Phi$ and $a.E \not\models \langle K \rangle \Phi$.
- If $a \in K$ then $a.E \models [K]\Phi$ iff $a.E \models \langle K \rangle \Phi$ iff $E \models \Phi$.
- $\Sigma\{E_i : i \in I\} \models [k]\Phi$ iff for all $j \in I.E_j \models [k]\Phi$.
- $\Sigma\{E_i : i \in I\} \models \langle k \rangle \Phi$ iff for some $j \in I.E_j \models \langle k \rangle \Phi$.

These facts can be used to show that $E\backslash J \models \Phi$ iff $E \models \Phi\backslash J$ where $\Phi\backslash J$ is obtained easily from Φ. The concurrency operator and renaming can be handled similarly, although the solution for concurrency is not entirely satisfactory.

While the above property language is sufficiently expressive for the first deadlock example, it cannot capture the second one, namely a process may become deadlocked after some sequence of unobservable actions. To write such properties, we extend the logic to include the syntax $[\![]\!]\Phi$ and $\langle\langle\rangle\rangle\Phi$ where Φ is a property. If a process has the property $[\![]\!]\Phi$, then doing any internal action sequence results in a process that has the property Φ. (Formally, $E \models [\![]\!]\Phi$ iff $\forall F \in \{E' : E \stackrel{\epsilon}{\Rightarrow} E'\}.F \models \Phi$.) If a process has the property $\langle\langle\rangle\rangle\Phi$, then doing some internal action sequence, possibly the empty sequence, results in a process that has the property Φ. (Formally, $E \models \langle\langle\rangle\rangle\Phi$ iff $\exists F \in \{E' : E \stackrel{\epsilon}{\Rightarrow} E'\}.F \models \Phi$.) Using the extended logic, the ability to evolve silently into a deadlocked process can be written as $\langle\langle\rangle\rangle[-]ff$.

We may want to extend the logic further to handle convergence and divergence. A process converges ($\downarrow$) if it cannot perform internal actions indefinitely, otherwise it diverges ($\uparrow$). We write $[\![\downarrow]\!]\Phi$ to say that the process converges and has the property $[\![]\!]\Phi$. We write $\langle\langle\uparrow\rangle\rangle\Phi$ to say that the process either diverges or has the property $\langle\langle\rangle\rangle\Phi$. For example, the property that an action a must (and will) happen next is written as $[\![\downarrow]\!]\langle\langle-\rangle\rangle tt \wedge [\![-a]\!]ff$ where $[\![]\!][K][\![]\!]\Phi$ is abbreviated as $[\![K]\!]\Phi$ and $\langle\langle\rangle\rangle\langle K\rangle\langle\langle\rangle\rangle\Phi$ is abbreviated as $\langle\langle K\rangle\rangle\Phi$ for any action set K.

In the text, the simplest language consisting of $tt, ff, \wedge, \vee, [K]\Phi,$ and$\langle K\rangle\Phi$ is called M for modal logic. The language consisting of $tt, ff, \wedge, \vee, [\![K]\!]\Phi, [\![]\!]\Phi, \langle\langle K\rangle\rangle\Phi,$ and $\langle\langle\rangle\rangle\Phi$ is called M^O for observable modal logic. Finally, M^O with $[\![\downarrow]\!]\Phi$ and $\langle\langle\uparrow\rangle\rangle\Phi$ is the language $M^{O\downarrow}$.

When should we consider two processes to be equivalent? A popular definition is to say that two processes E and F are equivalent if for any action a,

- if $E \stackrel{a}{\rightarrow} E'$, then $F \stackrel{a}{\rightarrow} F'$ for some F' such that E' and F' are equivalent.
- if $F \stackrel{a}{\rightarrow} F'$, then $E \stackrel{a}{\rightarrow} E'$ for some E' such that E' and F' are equivalent.

Two processes that meet the above criteria are said to be *bisimilar*. A binary relation B is a *bisimulation* if for any process pair (E, F) in B, the processes E and F are bisimilar.

If E and F are bisimilar processes, then for any process G, for any set of actions K, for any action a, and for any renaming function r, the following hold where $X \sim Y$ means that processes X and Y are bisimilar.

1. $a.E \sim a.F$
2. $E + G \sim F + G$
3. $E|G \sim F|G$
4. $E[r] \sim F[r]$
5. $E\backslash K \sim F\backslash K$

Because of this, bisimilar processes can replace one another in process descriptions. Furthermore, bisimilar processes either both have or both fail to have any property that is expressible in $M^{O\downarrow}$.

Two processes that share the same $M^{O\downarrow}$ properties may or may not be bisimilar. To characterize the class that are, let a process E be immediately image-finite if for all actions a the set $\{E' : E \stackrel{a}{\rightarrow} E'\}$ is finite. A process is image-finite if it can only become an immediately image-finite process, regardless of which action sequence it does. Two image-finite processes that have the same $M^{O\downarrow}$ properties are bisimilar.

We can demonstrate that two processes are bisimilar by constructing a bisimulation that contains them. Alternatively, we can write a proof that uses algebraic and semi-algebraic theorems (such as $(a.x)\backslash K \sim a.(x)\backslash K$ if $\{a, \bar{a}\} \cap K = \emptyset$) to show the equivalence.

The definitions for bisimulation and image-finite processes can be modified in a straightforward way to handle observable actions. The relationship between bisimulations and $M^{O\downarrow}$ occurs between observable bisimulations and M^{O}. Many other properties hold for both bisimulations and observable bisimulations.

The property language $M^{O\downarrow}$ is sufficiently expressive to distinguish one image-finite process from another, provided that the two processes are not bisimilar. This does not mean, however, that all interesting properties can be written in $M^{O\downarrow}$. An example of an inexpressible property is 'the process can never become deadlocked'. More generally, $M^{O\downarrow}$ lacks the expressive power to state long term properties, such as safety properties (something never happens), liveness properties (something eventually happens), and repeating properties (something happens again and again, forever).

We can discuss these events, by adding propositional variables to M. (As we shall see, $M^{O\downarrow}$ is equivalent to a fragment of this logic.) To complement the new syntax, valuations are added to the semantics. The valuation V assigns each variable z to a set of processes $V(z)$. If a process is in $V(z)$, then it is said to satisfy the formula z relative to the valuation V.

We use formulas in the extended logic to assign variables to sets of processes. More specifically, we write $z = \Phi$ if the variable z is assigned to the set of processes that satisfy formula Φ. For example, if $z = [-]ff$, then z is the set of processes that are deadlocked. If the variable is assigned to a formula that contains it (e.g. $z = [-]z$), then there may be a number of process sets that satisfy the equation. Roughly speaking, the smallest set is the least fixed point, written $\mu z.\Phi$, and the largest is the greatest fixed point, $\nu z.\Phi$.

Least and greatest fixed points are found through an iterative process. The number of iterations needed, in general, is an open problem. However, if the variables do not occur under an alternation of fixed points, then the number of iterations is, at most, $k * n$ where k is the number of fixed points and n is the number of processes.

The extended logic is called Modal Mu-Calculus (or μM) and its syntax is:

$$\Phi ::= tt \mid ff \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [K]\Phi \mid \langle K\rangle\Phi \mid \mu z.\Phi \mid \nu z.\Phi$$

where Φ, Φ_1, and Φ_2 are formulas, K is a set of actions, and z is a variable. The meanings of tt, ff, $\Phi_1 \wedge \Phi_2$, $\Phi_1 \vee \Phi_2$, $[K]\Phi$, and $\langle K\rangle\Phi$ are similar to the ones for the modal logic M; the only difference is that the valuation must be considered. For example, a process has the μM property $\Phi_1 \wedge \Phi_2$ relative to a valuation V, if it has the property Φ_1 relative to V and the property Φ_2 relative to V. The meaning of fixed points is given below where the notation $V[S/z]$ refers to the valuation that is identical to V, except that variable z is assigned to the process set S. (Formally, $V[S/z](z) = S$ and for all variables $y \neq z$, $V[S/z](y) = V(y)$.)

- $E \models_V \mu z.\Phi$ iff for all sets of processes S, if $E \notin S$ then there exists a process $F \notin S$ such that $F \models_{V[S/z]} \Phi$
- $E \models_V \nu z.\Phi$ iff there exists a set of processes S that contains E and for all $F \in S, F \models_{V[S/z]} \Phi$.

μM is very expressive. It can capture any $M^{O\downarrow}$ formula. In particular, assuming the variable z is not in Φ, $\langle\langle\rangle\rangle\Phi = \mu z.\Phi \vee \langle\tau\rangle z$, $[\![]\!]\Phi = \nu z.\Phi \wedge [\tau]z$, $\langle\langle\uparrow\rangle\rangle\Phi = \nu z.\Phi \vee \langle\tau\rangle z$, and $[\![\downarrow]\!]\Phi = \mu z.\Phi \wedge [\tau]z$. Also, many safety, liveness, and repeating properties that are inexpressible in $M^{O\downarrow}$ can be written in μM. For example, $\nu z.[K]ff \wedge [-]z$ says that no action in K ever happens, $\mu z.\langle -\rangle tt \wedge [-K]z$ says that some action in K eventually happens, and $\nu z.\langle K\rangle z$ says that actions in K can be done over-and-over again, forever.

Despite the greater expressive power, the relationship between μM and bisimulation is similar to that between M and bisimulation. Specifically, two bisimilar processes share the same μM properties and a

weakened version of image-finiteness is needed for processes that share the same μM properties to be bisimilar.

A disadvantage of using μM is that the satisfiability problem for it is undecidable. In general, we cannot determine if a process satisfies a μM formula relative to a valuation. For the other modal logics (M, M^O, and $M^{O\downarrow}$), we can grind through the satisfiability definition in a straightforward manner, but handling fixed points in μM is more challenging. One approach is to view answering a satisfiability question in terms of playing a game.

We want to play a game whose outcome will tell us if a process E satisfies a μM formula Φ relative to a valuation V. The game has two players, a verifier V who wants to show that the formula is satisfied and a refuter R who wants to show that it is not. A play of the game is a sequence of the form $(E_0, \Phi_0) \ldots (E_i, \Phi_i) \ldots$ where each subscripted E is a process and each subscripted Φ is a formula in μM. The rules of the game are:

- assuming we want to know if E satisfies Φ, E_0 is E and Φ_0 is Φ.
- if $\Phi_i = \Phi_j \wedge \Phi_k$ then player R chooses one of the conjuncts to be the formula in the next pair. In other words, E_{i+1} is E and R decides if Φ_{i+1} is Φ_j or Φ_k.
- if $\Phi_i = \Phi_j \vee \Phi_k$ then E_{i+1} is E and V decides if Φ_{i+1} is Φ_j or Φ_k.
- if $\Phi_i = [k]\Psi$, then player R chooses a transition $E_i \xrightarrow{a} E_j$ where $a \in k$. The next process, E_{i+1}, is E_j and the next formula, Φ_{i+1}, is Ψ.
- if $\Phi_i = \langle k \rangle \Psi$, then player V chooses a transition $E_i \xrightarrow{a} E_j$ where $a \in k$. The next process, E_{i+1}, is E_j and the next formula, Φ_{i+1}, is Ψ.
- if $\Phi_i = \sigma z.\Psi$ where $\sigma \in \{\mu, \nu\}$, then E_{i+1} is E_i and Φ_{i+1} is Ψ. Whenever z is encountered in the future, it will be replaced by Ψ. For example, if $\Phi_i = \mu z.\langle tick \rangle z$ then $\Phi_{i+1} = \langle tick \rangle z$, $\Phi_{i+2} = z$, and $\Phi_{i+3} = \langle tick \rangle z$. (For simplicity, we're assuming that each fixed point in Φ_i bounds a different variable and none of the bound variables are free elsewhere in the formula.)

Player R wins the game if the play contains a pair (E_i, Φ_i) such that $E_i \models_V \Phi_i$ is clearly false. Similarly, player V wins the game if the play contains a pair (E_i, Φ_i) such that $E_i \models_V \Phi_i$ is clearly true. If the play doesn't contain such a pair, then there must be some variables that are replaced according to the last rule infinitely often. If the outermost variable is bound to a least fixed point operator, then player R wins. Otherwise, player V wins.

If player R can always win the game, regardless of player V's moves, then E does not satisfy Φ relative to V. Otherwise, player V must be able to win the game, regardless of player R's moves, and E satisfies Φ relative to V.

Variations of this game have been developed to improve efficiency, particularly for fragments of μM. To prove that every process in a set satisfies some formula, we can play one of these games multiple (possibly infinitely-many) times. Alternatively, we can construct a proof tree, according to rules that are based on the satisfiability relation.

Contents

The book has seven chapters and is 181 pages. The chapter summaries are given below:

Chapter 1, **Processes**, introduces the notion of processes and gives a language in which to describe them.

Chapter 2, **Modalities and Capabilities**, presents the modal logics M, M^O, and $M^{O\downarrow}$ for discussing the properties of processes.

Chapter 3, **Bisimulations**, gives a formal definition for when two processes should be considered equivalent, argues that this choice matches our intuition, and examines the relationship between equivalent processes and those that are indistinguishable using the modal logics presented in Chapter 2.

Chapter 4, **Temporal Properties**, shows that the modal logics cannot be used to write safety and liveness properties. A variant of CTL is proposed as a more expressive logic. Fixed points are then introduced.

Chapter 5, **Modal Mu-Calculus**, defines the Modal Mu-Calculus (μM) and gives algorithms for finding fixed points using iterative methods.

Chapter 6, **Verifying Temporal Properties**, uses game theory to address the satisfiability problem for μM and the variant of CTL given in Chapter 4.

Chapter 7, **Exposing Structure**, uses proof trees to address the satisfiability problem for sets of processes.

Opinion

I recommend this book as a good introduction to process algebra, although an additional text will be needed for a solid understanding. Many examples are given throughout the text and exercises are provided at the end of each section. The examples complement the text very well and the exercises are given in increasing difficulty. This allows readers to test their general understanding first and then try to challenge themselves. The book also uses game theory as a way of presenting the same material from different perspectives without being overly redundant. I found this approach to be both effective and fun. Finally, the book does not assume much (if any) prior knowledge of the field.

I have only two negative comments. First, Chapters 4 and 5, while still good, are not as well written as the rest of the book. Second, the author occasionally refers to examples and definitions without telling the reader where the ideas are introduced. As a result, I had to flip through the book looking for text that I remembered reading, but whose details I couldn't recall. This situation may be helped by adding a glossary to future editions.