CONFERENCE PROCEEDINGS/FULL PAPERS
ISBN: 978-625-99063-9-3/February 2024

RSEP International Multidisciplinary Conference
8-9 February 2024, ATANA HOTEL, Dubai, UAE

Model Checking-Based Application-Dependent Quantitive Metrics to Drive the Fault Probability for the Discrete State System

Saeid Pashazadeh
Full Professor, Department of Information Technology, Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran
E-mail: pashazadeh@tabrizu.ac.ir
Maryam Pashazadeh
Ph.D. candidate, Department of Computer Engineering, Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran
E-mail: mpashazadeh@tabrizu.ac.ir

DOI: https://doi.org/10.19275/RSEPCONFERENCES287

Abstract

Risk evaluation needs the probability of faults, and two common approaches for accomplishing it are gathering statistics from existing systems or relying on the experience of experts. In cases where we develop a novel system, none of these approaches is feasible. This paper proposes an application-dependent method to drive a quantitative metric that can yield fault probability for risk analysis. This method applies to discrete state-based systems. We model a system via formal methods like colored Petri nets and then generate the state-space graph of the system via the CPN tool. The state-space graph contains complete information about the states of the system, and we can prove some of the system's properties via model checking. Model checking permits us to define application/system-specific values that give feasible metrics to assign the probability value for a fault. A warehouse guard movement strategy is considered a case study, and we proposed a colored Petri net model of the system. A metric from model checking of the state-space graph is derived to assign the probability of the intruder's success in infiltrating the warehouse.

Keywords: Modeling, model checking, risk evaluation, state-space graph, quantitative metric
Jel codes: C49, C60, C69

1. Introduction

Risk analysis and its management require detecting failure sources, the probability of each failure and the cost it imposes on us. There are several known ways to determine the probability of any failure. If we have proper information about the history of failure, we can use this statistical data to calculate the probability of failure. When we do not have access to statistical information, the alternative solution is to use fuzzy methods based on the opinion of one or more experts. Sometimes, none of the above methods can be done. One of these situations is when a system is designed or deployed for the first time so that even experts do not have the necessary information regarding the likelihood of failures. This article proposes an innovative method to help experts create criteria to determine the probability of various failures. Naturally, the proposed method cannot be used in all cases, but it can be used in most discrete state systems that have a finite state space.

2. Literature Review

Modeling a system will be done for two purposes: 1) verification of the system's functional properties via mathematical-based methods known as formal methods, and 2) performance evaluation of the system via simulation tools. Many formal methods have been developed, each appropriate for specific systems and purposes. Carl Adam Petri designed the Petri net, an easy, formal method based on the bag theory. The Petri net has a graphical interface and good modeling capability. The classical Petri net has evolved over the years, and the colored Petri net is one of its latest famous variants (Jensen and Kristensen, 2009).

Colored Petri net benefits from a traditional artificial intelligence language named ML based on lambda calculus. Combining the ML with the classical Petri net kept the formal modeling property of the classical Petri net and significantly improved its modeling capability (Paulson, 1996). ML language used in the colored Petri net is slightly different from the clasical ML language. Some constructs have been added to it for compatibility with concepts like color sets and markings of colored Petri net (Harper, 2011). One of the best tools for modeling with colored Petri net is the CPN tool developed at the Aarhus University of Denmark (Jensen and Kristensen, 2009). We used this modeling tool in this paper.

The CPN tool automatically generates a state space graph of the model system. This graph has complete information about all states of the system. Generally, researchers use model-checking languages to study the modelled systems' various properties (Baier and Katoen, 2008). In this paper, we prefer using our developed function to analyze the state-space graph instead of using model-checking languages like linear or computational temporal logic.

Colored Petri net is used by many researchers and for a wide range of applications and studies. Colored Petri net has been used for modeling the puzzle games for automatic driving of the solutions and also driving some metrics for determining the difficulty level of each puzzle via analysis of the state-space diagram (Pashazadeh, 2016).

3. Case study system

As a case study, let us assume that we have a warehouse in a big field that needs to be protected against intruders, especially at night by night security guards (watchmen). Let's assume we use night security guards to walk regularly with a predetermined schedule and plan to protect the warehouse. Security experts can define many different security programs by considering various quality metrics. Some of these metrics are: 1) using fewer night security guards to decrease security costs, 2) decreasing the number of vehicle costs, and 3) capturing the intruders early. These quality metrics are mutually exclusive and need to be balanced. Decreasing the number of night guards decreases security costs but leads to the late capture of the intruder and may increase the probability of the intruder's success. We are looking for a security program that guarantees us that the intruder can not succeed in reaching the warehouse and uses a minimum number of security guards.

To simplify the problem statement, let's assume that the system model is shown in Figure 1. We divided the field into blocks of the same size. The number of blocks is parametrized, and Figure 1 is rectangular with 9x9 blocks. In this study, we considered only one intruder in the parametric position row number five and column number 9. We assumed that the intruder came from the field's border and tried to reach the warehouse. Its place is also parametric with row number 5 and column number 5. We have four guards whose movement beeline is parametric and can be vertical or horizontal, shown with dotted lines. Each night, security guards return when they reach the border of the field. The initial position and starting movement direction (forward or backward) are parametric.

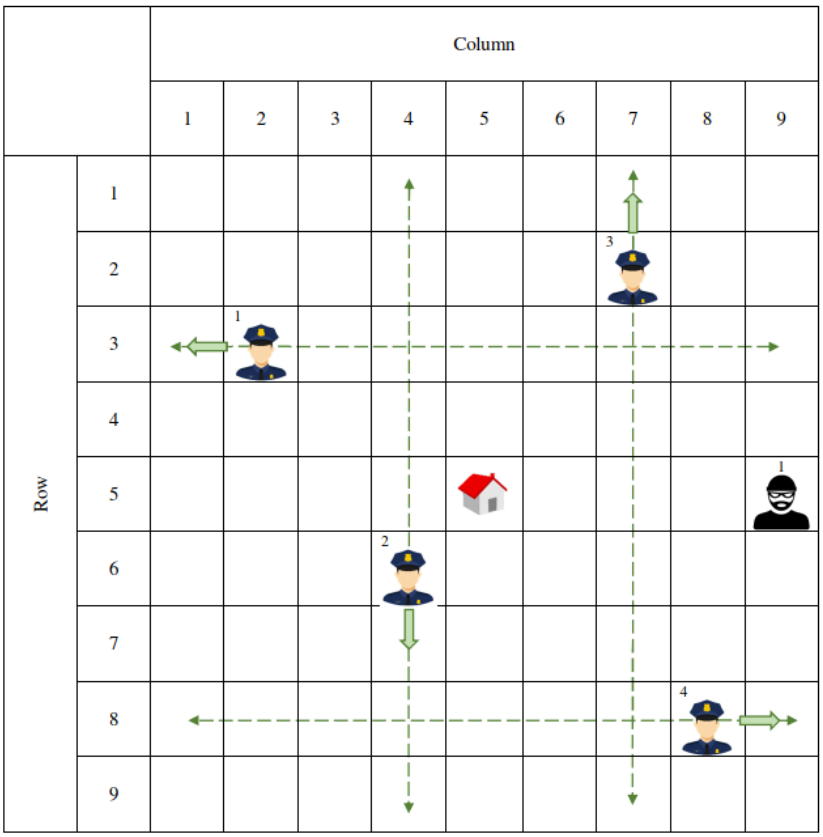


Figure 1. Simple schematic view of the warehouse, night guards, and an intruder.

The forward direction is the direction with increasing column or row number values. We used discrete movement and time for modeling the system. At each step, only one party can move. Let us assume that the intruder first moves, and then all guards move sequentially based on their predefined program. No guard follows the intruder and only walks based on the predetermined program. For simplicity, we assumed that all parties can move vertically or horizontally in four directions.

4. Hierarchical colored Petri net model of the system

Figure 2 shows the top-level module of our proposed system. This model contains two places and eight substitution transitions. Each substitution transition is related to the intruder's or watchman's (guard's) movement. Place TARGET represents the runner's status, and place PROCESS QUEUE represents all information about guards and intruders. We only considered one intruder in this model, but the model can manage more intruders.

Figure 3 shows the submodule of GUARD GO RIGHT. This submodule models a guard's movement based on its current position and predefined program to move to the right cell. Figure 3 shows the submodule of INTRUDER GO DOWN. This submodule models the intruder's movement to down when it is capable.

Table 1 demonstrates the definition of colorsets, variables, and values we used to model the system. We defined a few functions for modeling the system and also for the analysis of the state-space graph. The model stops and reaches the dead marking when one of these conditions reaches 1) a guard will capture the intruder. In this case, the marking of place TARGET will be "grabbed", and in the marking of place PROCESS QUEUE we see that the position of at least one guard and the intruder will be the same. 2) the intruder reaches the warehouse. In this case, the marking of place TARGET will be "successed," and we see that the intruder's position will be the same as the warehouse in the marking of place PROCESS QUEUE.

Table 2 shows the summary report of the state-space graph generation of the model. This table shows the statistics of the model when we considered the 9x9 mesh size of the field and one intruder in the initial position (5,9). The number of dead markings of the model is 40, and some are shown. Table 3 shows the summary report of the state-space generation of the model with the 11x11 mesh size and initial position of the intruder (1,1). Tables 1 and 2 show that the runtime of state-space graph generation by CPN tools is very low, with a small mesh grid size.

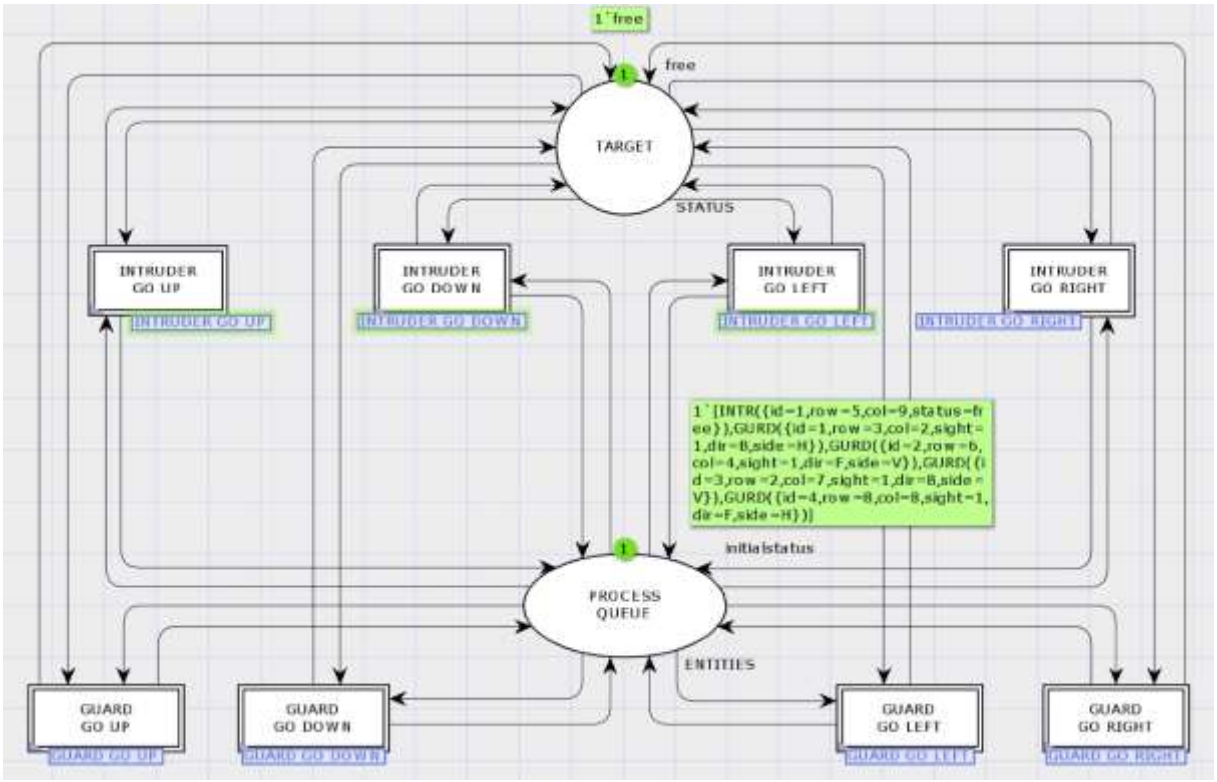


Figure 2. The top module of the proposed colored Petri net model of the system.

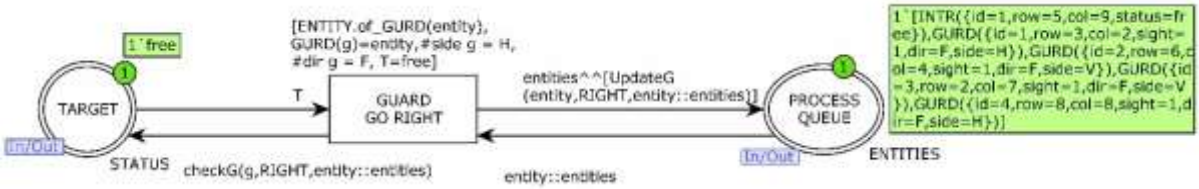


Figure 3. Colored Petri net model of the "Guard Go Right" submodule.

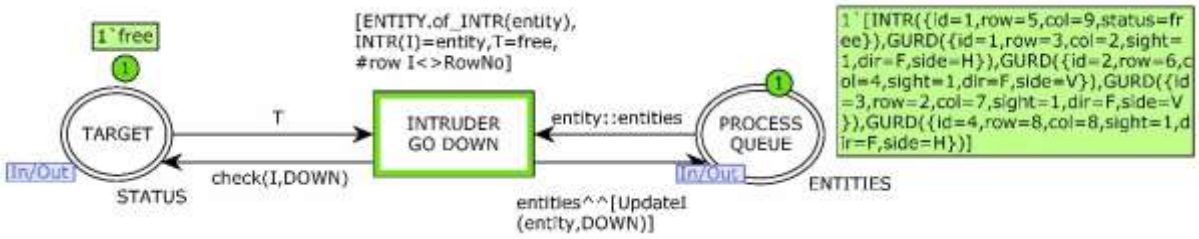


Figure 4. Colored Petri net model of the "Intruder Go Down" submodule.

Table 1. Definitions of color sets, variables, and values used in the proposed model.

	Definition
Color sets	colset MOVE = with UP|DOWN|LEFT|RIGHT; colset SIDE=with V | H;
	colset STATUS = with free | successed | grabbed; colset DIRECTION=with F | B;
	colset TURN = with intruder | guard;
	colset GUARDREC = record id:INT * row:INT * col:INT * sight:INT * dir:DIRECTIOside:SIDE;
	colset INTRUDERREC = record id:INT * row:INT * col:INT * status:STATUS;
	colset ENTITY = union INTR:INTRUDERREC + GURD:GUARDREC;
	colset ENTITIES = list ENTITY;
Variables	var entity, newentity: ENTITY; var entities: ENTITIES; var g: GUARDREC;
	var I, p: INTRUDERREC; var T: STATUS; var ES: ENTITIES;
Values	val RowNo = 9; val ColNo = 9; val row_target=5; val col_target=5; val a = 1;
	val initialstatus=[INTR({id=1,row=5,col=9,status=free}), GURD({id=1,row=3,col=2,sight=1,dir=B,side=H}), GURD({id=2,row=6,col=4,sight=1,dir=F,side=V}), GURD({id=3,row=2,col=7,sight=1,dir=B,side=V}), GURD({id=4,row=8,col=8,sight=1,dir=F,side=H})];

Table 2. State-space report of the proposed model with a mesh size 9x9.

Metrics	Nodes	Arcs	Seconds	Status	Dead Markings
State Space	3144	4644	1	Full	40 [939,870,842,757,695,...]
Scc Graph	105	188	0	-	

Table 3. State-space report of the proposed model with a mesh size of 11x11.

Metrics	Nodes	Arcs	Seconds	Status	Dead Markings
State Space	5930	8924	1	Full	50 [98,820,767,653,62,...]
Scc Graph	131	236	0	-	

5. Analysis of the state-space graph

The state-space graph of our proposed system with a mesh size of 9x9 has 3144 nodes and 4644 arcs. This graph will be bigger if a bigger mesh size of the field is used. This graph is a directed and, many times, cyclic graph. The leaf nodes of this graph show the dead markings. The cycle of the state-space graph in our model demonstrates an infinite loop in case the intruder can not reach the warehouse and guards can not capture it. A deeper analysis of the state-space graph gives more information about the metrics of the proposed security program. Let's assume the following key questions. 1) can the intruder reach the warehouse? 2) how many ways (plans) can the intruder find to reach the warehouse? 3) how many steps are required to reach the intruder to the warehouse? 4) what is the number of least steps for reaching the intruder to the warehouse? In this part of the paper, we want to demonstrate how we can answer these questions via the analysis of the state-space graph of the proposed model of the system.

Figure 5 shows some queries for analysis of the state-space graph of our proposed system model. Table 2 shows that the model has 40 dead markings and shows some of them. Calling the build-in function ListDeadMarkings() of CPN tools gives us the list of dead marking nodes of the state-space graph. Results show that node 939 is one of the dead markings of the state-space graph. Calling the build-in function NodesInPath() of CPN tools with parameters 1 and 939 gives us the list of nodes that start from the graph's initial node (1) and lead to the dead marking 939. The function call, List.length(NodesInPath(1, 939)) gives us the number of nodes in this path.

Now a question arises: Does dead marking 939 represent the failure of the intruder or its success? Calling the built-in function, Mark helps us answer this question. Calling the Mark.MASTER'TARGET 1 939 shows the marking of place TARGET in the top module of the model at state node 939. It shows that guards have grabbed the intruder. It is good news and shows the strength of our security program. The function call Mark.MASTER'TARGET 1 870 shows that the intruder will be grabbed in the second dead marking 870. We write a function named checkAlwaysGrabbed that gets a list of dead markings and checks that all dead markings lead to an intruder being grabbed. Results show that the answer is negative. This is bad news regarding our security program and shows that sometimes it fails. Therefore the probability of security program failure is not zero. We wrote a function named getSuccedList that gives the list of dead markings and returns the list of dead markings that leads to the intruder's success in reaching the warehouse. The length of this list can be used as our first simple metric to assign the failure probability of the security program. In this example, 8 dead markings from 40 dead markings represent the success of the intruder. The higher value of dead markings with intruder's success shows a higher failure rate of the security program.

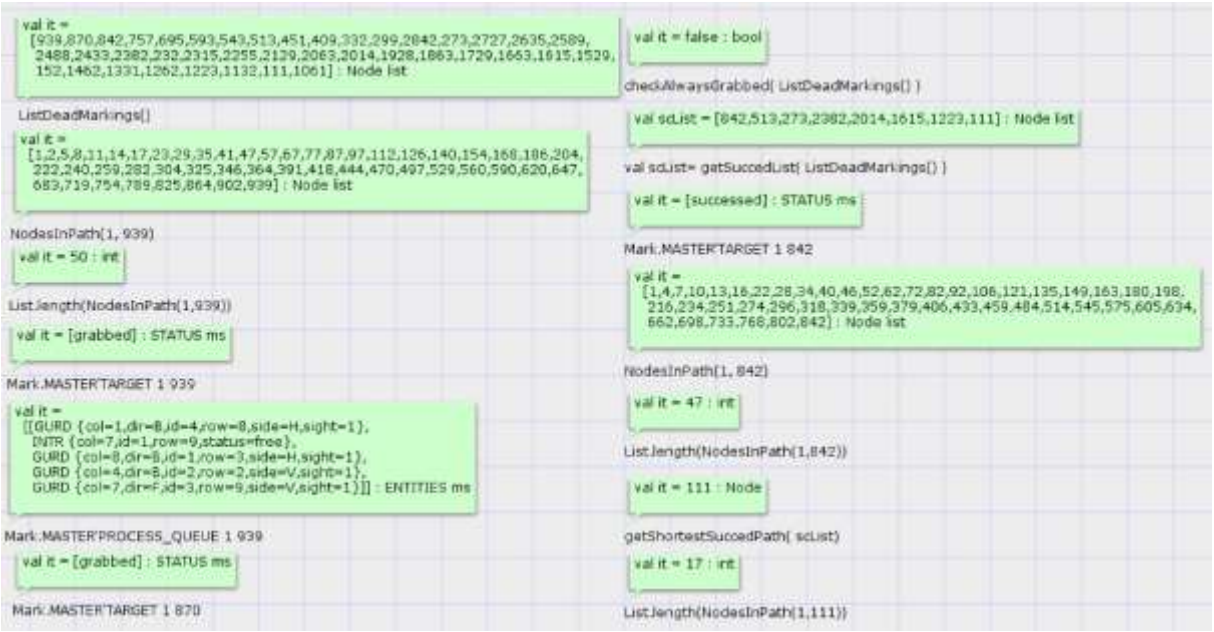
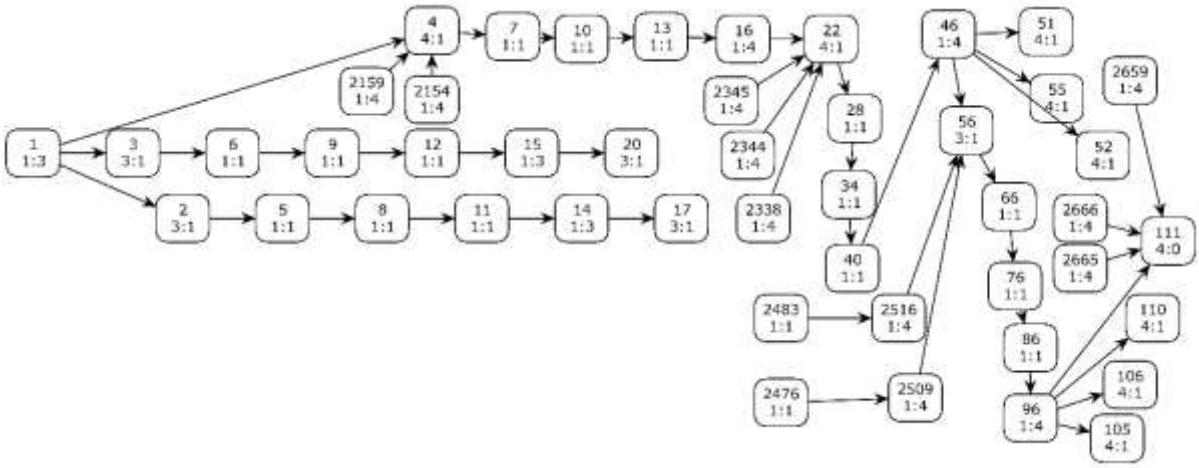


Figure 5. Results of some state-space analysis queries.

Another metric that helps us assign failure probability is the difficulty level (steps) of the intruder's success. Calling function NodesInPath 1 842 shows the list of nodes in the path from the initial node to the sead marking node 842. The length of this list is 47, indicating that the intruder had a long way to go to succeed in its work. A longer path length value represents a lower probability of the security program's failure. We wrote a function named getShortestSuccedPath that gives the list of dead markings of intruder success and returns the length of the shortest path that leads to its success. The result of this function also can be used as a metric for the assignment of failure probability to a security program. The lower value of this result shows the likelihood of the intruder's success in breaking the security program.

Graph 1 shows part of the state-space graph displaying the shortest path from initial node 1 to dead marking node 111, leading to the intruder's success. This graph shows many paths that start from initial node 1 and lead to dead marking node 111. The number of paths that lead to the intruder's success dead markings can be used as a good metric for representing the likelihood of the security program's failure rate.

Graph 1. Partial state-space graph of a path leading to success of intruder in node 111.



6. Conclusions

This paper presented a novel method based on the discrete-state modeling of a system using colored Petri net to drive some metrics for the assignment of failure probability for systems that we do not have statistics from its working history or are new systems that experts are not familiar with. We used a security guarding program as a case study and proposed a hierarchical model of it. Via state-space analysis, we defined a few metrics that can be used to assign failure probability. 1) the number of dead markings that represent the intruder's success, 2) the lengths of the paths that lead to these dead markings, 3) the length of the shortest path to these dead markings, and 4) the number of paths that reaches to these dead markings are four critical metrics for failure likelihood assignment. We can easily drive a customized formula using these four metrics to drive failure probability.

References

Baier C. & Katoen J.-P. (2008) *Principles of model checking*. MIT Press.

Berlin, Heidelberg: Springer Publishing Company, Incorporated. https://doi.org/10.1007/b95112

Harper R. (2011) *Programming in Standard ML*. San Francisco, California, USA: Carnegie Mellon University.

Jensen K. & Kristensen L. M. (2009) *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*.

Paulson L. C. (1996) *ML for the working programmer*, (2nd ed.), Cambridge University Press.