Formalization of Birth-Death and IID Processes in Higher-order Logic

Liya Liu
Department of Electrical and Computer Engineering
Concordia University
Montreal, Canada
Email: liy_liu@ece.concordia.ca

Osman Hasan
Department of Electrical and Computer Engineering
Concordia University
Montreal, Canada
Email: o_hasan@ece.concordia.ca

Sofiène Tahar
Department of Electrical and Computer Engineering
Concordia University
Montreal, Canada
Email: tahar@ece.concordia.ca

***Abstract*—Markov chains are extensively used in the modeling and analysis of engineering and scientific problems. Usually, paper-and-pencil proofs, simulation or computer algebra software are used to analyze Markovian models. However, these techniques either are not scalable or do not guarantee accurate results, which are vital in safety-critical systems. Probabilistic model checking has been proposed to formally analyze Markovian systems, but it suffers from the inherent state-explosion problem and unacceptable long computation times. Higher-order-logic theorem proving has been recently used to overcome the above-mentioned limitations but it lacks any support for discrete Birth-Death process and Independent and Identically Distributed (IID) random process, which are frequently used in many system analysis problems. In this paper, we formalize these notions using formal Discrete-Time Markov Chains (DTMC) with finite state-space and classified DTMCs in higher-order logic theorem proving. To demonstrate the usefulness of the formalizations, we present the formal performance analysis of two software applications.**

I. Introduction

In probability theory, Markov chains are used to model time varying random phenomena that exhibit the memoryless property [BW90]. In fact, most of the randomness that we encounter in engineering and scientific domains has some sort of time-dependency. For example, noise signals vary with time, the duration of a telephone call is somehow related to the time it is made, population growth is time dependent and so is the case with chemical reactions. Therefore, Markov chains have been extensively investigated and applied for designing systems in many branches of science and engineering. Moreover, various discrete time random processes, i.e., Independent and Identically Distributed (IID) random processes, can be proved to be Discrete-time Markov chains (DTMC).

Mathematically, Markov chain models are divided into two main categories. It may be *time homogeneous*, which refers to the case where the underlying Markov chains exhibit the constant transition probabilities between the states, or *time inhomogeneous*, where the transition probabilities between the states are not constant and are time dependent. A DTMC consists of a list of the possible states of the system along with the possible transition paths among these states. Based on the state space, DTMCs are also classified in terms of the characteristics of their states. For example, in a DTMC, if every state in its state space is aperiodic and can be reached from any other state including itself in finite steps, then it is considered as an aperiodic and irreducible DTMCs [HÖ2]. This type of DTMCs are considered to be the most widely used ones in analyzing Markovian systems because of their attractive stationary properties, i.e., their limit probability distributions are independent of the initial distributions. Discrete-time Birth-Death process is such a typical aperiodic and irreducible DTMC and it is a fundamental for Queuing Models.

Traditionally, engineers have been using paper-and-pencil proof methods to perform probabilistic and statistical analysis of Markov chain systems. Nowadays, real-world systems have become considerably complex and the behaviors of some critical subsystems need to be analyzed accurately. However, due to the increasing complexity, it becomes practically impossible to analyze a complex system precisely by paper-and-pencil methods due to the risk of human errors. Therefore a variety of computer-based techniques, such as simulation, computer algebra systems and probabilistic model checking have been used to analyze Markovian models.

The simulation based analysis is irreverently inaccurate due to the usage of pseudo random number generators and the sampling based nature of the approach. To improve the accuracy of the simulation results, Markov Chain Monte Carlo (MCMC) methods [Mac98], which involve sampling from desired probability distributions by constructing a Markov chain with the desired distribution, are frequently applied. The major limitation of MCMC is that it generally requires hundreds of thousands of simulations to evaluate the desired probabilistic quantities and becomes impractical when each simulation step involves extensive computations. Computer Algebra Systems (CAS) provide automated support for analyzing Markovian models and symbolic representations of Markovian systems using software tools, such as Mathematica [Mat17] and Maple [Map17]. However, the usage of huge symbolic manipulation algorithms, which have not been verified, in the cores of CASs also makes the analysis results untrustworthy. In addition, the computations in CAS cannot be completely trusted as well since they are based on numerical methods.

Due to the extensive usage of Markov chains for safety-critical systems, and thus the requirement of accurate analysis

978-1-5090-4623-2/17/$31.00 ©2017 IEEE

in these domains, probabilistic model checking has been recently proposed for formally analyzing Markovian systems. However, some algorithms implemented in these model checking tools are based on numerical methods too. For example, the Power method [Par02], which is a well-known iterative method, is applied to compute the steady-state probabilities (or limiting probabilities) of Markov chains in PRISM [PRI17]. Moreover, model checking cannot be used to verify generic mathematical expressions with universally quantified continuous variables for the properties of interest (i.e., variable values have to be bounded and discredited to avoid endless computation time). Finally, model checkers suffer from the state-exploration problem when the analyzed systems are complex.

Higher-order-logic interactive theorem proving [Gor89] is a formal method that provides a conceptually simple formalism with precise semantics. It allows to construct a computer based mathematical model of the system and uses mathematical reasoning to formally verify systems properties of interest. The formal nature of the analysis allows us to solve the inaccuracy problem mentioned above. Due to the highly expressive nature of higher-order logic and the inherent soundness of theorem proving, this technique is capable of conducting the formal analysis of various Markov chain models including hidden Markovian models [Bul06], which, to our best knowledge, probabilistic model checking cannot cater for. Moreover, interactive theorem proving using higher-order logic is capable of verifying generic mathematical expressions and it does not suffer from the state-exploration problem of model checking.

Leveraging upon the above-mentioned strengths of higher-order-logic theorem proving and building upon a formalization of probability theory in HOL [MHT11]. In earlier work, we have formalized the definitions of DTMC [LHT11] [LHT13b] and classified DTMC [LAHT13] in higher-order logic. These definitions have then been used to formally verify classical properties of DTMCs, such as the joint probability theorem, Chapman-Kolmogorov Equation and Absolute probability [LHT13b], as well as Classified DTMCs, such as the stationary properties [LAHT13]. The above-mentioned formalization has been successfully used to verify some real-world applications, such as a binary communication channel [LHT11], an Automatic Mail Quality Measurement protocol [LHT13b], a generic LRU (least recently used) stack model [LAHT13] and a memory contention problem in a Multiprocessor System [LHT13a], and to also formalize Hidden Markov Models (HMMs) [LAHT14].

These previous works clearly indicate the great potential and usefulness of higher-order-logic theorem proving based formal analysis of Markovian models. In the current paper, we utilize the core formalizations of DTMC [LHT13b] and classified DTMC [LAHT13] to formally prove that the Independent and Identically Distributed (IID) process is a DTMC. Moreover, we formalize the discrete time Birth Death process in this paper as it is also a widely used phenomenon in while analyzing DTMCs. We illustrate our findings on two basic software applications.

II. IID Random Process

In this section, we formally validate that an *Independent and Identically Distributed (IID)* random process (model) is a DTMC.

In probability theory, a collection of random variables is called independent and identically distributed if all of the random variables have the same probability distribution and are mutually independent [Dat08]. *IID* random processes play an important role in modelling repeated independent trials, such as Bernouli trails. In HOL, the IID random process can be formally defined as:

Definition 2.1: $\vdash \forall$ p X s. iid_rp p X s = $\forall$i. random_variable (X i) p s $\wedge$ FINITE (space s) $\wedge$ ($\forall$i. x$\in$space s $\Rightarrow$ {x} $\in$ subsets s) $\wedge$ ($\forall$i. i$\in$space s $\Rightarrow$ (p$_0$ i = $\mathbb{P}${x | X 0 x = i})) $\wedge$ $\forall$B st. (st $\subseteq$ {(i, j) | i $\in$ univ $\wedge$ {B j} $\in$ subset s}) $\Rightarrow$ ($\mathbb{P}$ ($\bigcap_{(i,j) \in st}$ {x | X i x = B j}) = PROD (λ(i, j). $\mathbb{P}${x | X i x = B j} st) $\wedge$ $\forall$a i j. $\mathbb{P}${x | X i x = a} = $\mathbb{P}${x | X j x = a}

where the first conjunction defines this random process as a collection of random variables $\{X_i\}$ (i is a natural number), which corresponds to the function X i in higher-order logic, the second condition defines this random process on a *finite state-space*, the third condition ensures that the events associated with the state-space (space s) are in the event space (subsets s), which is a *discrete* space, the next conjunction $\forall$i. i $\in$ space s $\Rightarrow$ (p$_0$ i = $\mathbb{P}${x | X 0 x = i}) defines a general initial distribution p_0 for all states in the state-space space s. The last two conditions define the mutual independence and identical distribution properties.

It is important to note that the notion of mutual independence, also called *stochastic mutual independence*, is different from *mutually exclusive* and *pairwise independence* [Goo88]. It refers to the case when the random variables are measurable functions from the set of possible outcomes x to an event set subsets s, where events are represented by E_{i_k} and the random variables satisfy $\mathcal{P}r(E_{i_1}, \cdots, E_{i_k}) = \prod_{k=0}^{n-1} \mathcal{P}r(E_{i_1}) \cdots \mathcal{P}r(E_{i_k})$.

Note that the events $E_{i_1} \cdots E_{i_k}$ do not have to be successive. Thus, in Definition 2.1, a set st is defined as a subset of a pair set (i, j), {(i, j) | i $\in$ univ(:num) $\wedge$ {B j} $\in$ subset s}, in which the index of a random variable i can be any natural number, while the event {B j} is in the event set subsets s. In higher-order logic, (λx. t x) refers to a function that maps *x* to *t(x)*; PROD (λt. x t) s indicates to the mathematical product $\prod_{t \in s} x_t$.

The last condition $\mathbb{P}${x | X i x = a} = $\mathbb{P}${x | X j x = a} refers to the property that the random variables in the process have an identical distribution for any event in the event set.

Now, we can prove that a discrete IID random process with a finite space is a DTMC using Definitions 2.1 as follows.

Theorem 2.1: ⊢ ∀ X p s p_0.
iid_rp X p s p_0 ⇒
dtmc X p s p_0 (\t i j.
$\mathbb{P}$({x|X (t+1) x = j}|{x|X t x = i})

To prove that a finite discrete IID random process is a DTMC, we first have to prove

$\mathbb{P}$({x|X (t+1) x = j}|{x|X t x = i}) =
if i ∈ space s ∧ j ∈ space s then
$\mathbb{P}$({x|X (t+1) x = j}|{x|X t x = i})
else 0

and then have to prove the second condition in the *Markov Property* defined in [LAHT13] as:

Definition 2.2: ⊢ ∀ X p s. mc_property X p s=
(∀ t. random_variable (X t) p s) ∧
∀ f t n. increasing_seq t ∧
$\mathbb{P}(\bigcap_{k\in[0,n-1]}\{x|X\ t_k\ x = f\ k\}) \neq 0$ ⇒
($\mathbb{P}(\{x|X\ t_{n+1}\ x = f\ (n+1)\}|$
$\{x|X\ t_n\ x = f\ n\} \cap$
$\bigcap_{k\in[0,n-1]}\{x\ |\ X\ t_k\ x = f\ k\})$ =
$\mathbb{P}(\{x|X\ t_{n+1}\ x = f\ (n+1)\}|$
$\{x|X\ t_n\ x = f\ n\})$)

This step can be executed for two cases: n = 0 and $n > 0$. The first case is to prove

$\mathbb{P}$({x|X (t+1) x = f j}|{x|X t x = f i})=
$\mathbb{P}$({x|X (t+1) x = f j})

which can be verified by applying the mutual independence property of Definition 2.1. The second case can be verified by using some properties of product and the mutual independence.

The verification of the above theorem is one of the prerequisites to formalize random walk and gambler's ruins, which are frequently applied in modelling many interesting systems, such as behavioral ecology [CPB08], financial status prediction (modelling the price of a fluctuating stock as a random walk) [Fam65], etc.

III. Discrete-time Birth-Death Process

In this section, the discrete-time Birth-Death Process is formalized and applied to formally verify its stationary features (such as limit probabilities and stationary distributions). A discrete-time Birth-Death process [Tri02] is an important sub-class of Markov chains as it involves a state-space with nonnegative integers. Its remarkable feature is that all one-step transitions lead only to the nearest neighbor state. Discrete-time Birth-Death Processes are mainly used in analyzing software stability, for example, verifying if a data structure will have overflow problems.

A discrete-time Birth-Death Process, in which the states refer to the population, can be described as a state diagram, as depicted in Figure 1. In this diagram, the states 0, 1, $\cdots$, i, $\cdots$ are associated with the population. The transition probabilities b_i represent the probability of a birth when the population is i, d_i denotes the probability of a death when the population becomes i, and a_i refers to the probability of the population in the state i. Considering $0 \leq a_i \leq 1$, $0 < b_i < 1$ and $0 < d_i < 1$ (for all i, $1 \leq i \leq n$), the Birth-Death process described here is not a pure birth or pure death process as the population is finite. Thus, the Birth-Death process can be modeled as an aperiodic and irreducible DTMC [Tri02]. In this DTMC model, a_i, b_i and d_i should satisfy the additivity of probability axiom [PT00]. Then, in this DTMC model, the amount of population is greater than 1. Also, a_i, b_i and d_i should satisfy the additivity of probability axiom. Now, the discrete-time Birth-Death process can be formalized as:

Definition 3.1: ⊢ ∀ a b d t i j.
DBLt a b d t i j =
if (i = 0) ∧ (j = 0) then a 0
else if (i = 0) ∧ (j = 1) then b 0
else if (0 < i) ∧ (i−j=1) then d i
else if (0 < i) ∧ (i = j) then a i
else if (0 < i) ∧ (j−i=1) then b i
else 0;

This definition leads to the following formalization of the discrete-time Birth-Death process:

Definition 3.2: ⊢ ∀ X p a b c d n p_0.
DB_MODEL X p a b d n p_0 =
Aperiodic_MC X p
([0,n], POW [0,n]) p_0 (DBLt a b d) ∧
Irreducible_MC X p
([0,n],POW [0,n])p_0 (DBLt a b d) ∧
1 < n ∧ (a 0 + b 0 = 1) ∧
(∀j. 0<j ∧ j<n ⇒(a j + b j + d j=1)) ∧
(∀j. j<n ⇒ 0 < a j ∧ 0 < b j ∧ 0 < d j)

In this definition, this process is formally described as an aperiodic and irreducible DTMC, in which the state-space is expressed as a pair ([0,n], POW [0,n]). The set [0,n] represents the population and POW [0,n] is the sigma-algebra of the set [0,n]. Since the aperiodic and irreducible DTMC is independent of the initial distribution, the parameter p_0 in this model is a general function. The other conjunctions shown in Definition 3.2 are the requirements described in the specification of the discrete-time Birth-Death process mentioned above.

Now, we can prove that this discrete-time Birth-Death process has the limiting probabilities.

Theorem 3.1: ⊢ ∀ X p a b d n p_0.
DB_MODEL X p a b d n p_0 ⇒
(∃ u. $\mathbb{P}${x | X t x = i} → u)

Here, the right arrow → is a higher-order logic notation and that denotes "tends to". This theorem can be verified by rewriting the goal with Definition 3.2 and then applying Theorem 7 in [LAHT13].

Now, we can prove that the limit probabilities are the stationary distributions and are independent of the initial probability vector as the following theorem.

Theorem 3.2: ⊢ ∀ X p a b d n p_0.
DB_MODEL X p a b d n p_0 ⇒
(∃ f. stationary_dist p X f s)

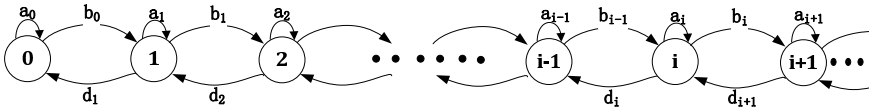

Fig. 1. The State Diagram of Birth-Death Process

We proved this theorem by first instantiating f to be the limiting probabilities, lim (λt. $\mathbb{P}\{$x|X t x = i$\}$, and then by applying Theorem 3.1.

The last two theorems verify that the Birth-Death process holds the steady-state probability vector $V_i = \lim_{t\to\infty} \mathbb{P}\{X_t = i\}$. The computation of the steady-state probability vector V_i is mainly based on the following two Equations (1a) and (1b):

$$v_0 = a_0 v_0 + d_1 v_1 \tag{1a}$$

$$v_i = b_{i-1} v_{i-1} + a_i v_i + d_{i+1} v_{i+1} \tag{1b}$$

Now, these two equations can be formally verified by the following two theorems.

Theorem 3.3: $\vdash \forall$ X p a b d n p_0.
DB_MODEL X p a b d n p_0 $\Rightarrow$
(lim (λt. $\mathbb{P}\{$x|X t x = 0$\}$) =
a 0 * lim (λt. $\mathbb{P}\{$x|X t x = 0$\}$) +
d 1 * lim (λt. $\mathbb{P}\{$x|X t x = 1$\}$))

The proof steps use Absolute Probability Theorems and Theorems 3.2 to simplify the main goal and the resulting subgoal can be verified by applying the conditional probability additivity theorem, along with some arithmetic reasoning.

Theorem 3.4: $\vdash \forall$ X p a b d n i p_0.
DB_MODEL X p a b d n p_0 $\wedge$ i+1 $\in$ [0,n] $\wedge$
i-1 $\in$ [0,n] $\Rightarrow$
(lim (λt. $\mathbb{P}\{$x|X t x = i$\}$) =
b (i-1) * lim (λt. $\mathbb{P}\{$x|X t x = i-1$\}$) +
a i * lim (λt. $\mathbb{P}\{$x|X t x = i$\}$) +
d (i+1) * lim (λt. $\mathbb{P}\{$x|X t x = i+1$\}$))

We proceed with the proof of this theorem by applying Absolute Probability Theorems [LAHT13], Theorem 3.2, Theorem 3.3 and the total probability theorem along with some arithmetic reasoning.

The general solution of the linear Equations (1a) and (1b) are expressed as:

$$v_{i+1} = \prod_{j=1}^{i+1} \frac{\mathtt{b_{j-1}}}{\mathtt{d_j}} v_0 \tag{2a}$$

$$v_0 = \frac{1}{\sum_{i=0}^{n} \prod_{j=1}^{i+1} \frac{\mathtt{b\ (j-1)}}{\mathtt{d\ j}}} \tag{2b}$$

These two equations are the major targets of the long-term behavior analysis and have been verified in HOL as the following two theorems:

Theorem 3.5: $\vdash \forall$ X p a b d n i Linit.
DB_MODEL X p a b d n Linit $\wedge$
i + 1 $\in$ [0,n] $\Rightarrow$
(lim (λt. $\mathbb{P}\{$x|X t x = i + 1$\}$) =
lim (λt. $\mathbb{P}\{$x|X t x = 0$\}$) *
PROD (1, i + 1) (λj. $\frac{b\ (j-1)}{d\ j}$))

The proof of this theorem starts by induction on the variable n. The base case can be verified by Theorem 3.3 and some arithmetic reasoning. The proof of the step case is then completed by applying a lemma that proves the following Equation (3) based on the DB_MODEL, which describes the discrete-time Birth-Death process model, of Definition 3.2:

$$v_{i+1} = \frac{b_i}{d_{i+1}} v_{i+1} \tag{3}$$

The formal proof of Equation (3) is mainly done by induction on the variable i. The base case is proved by applying Absolute Probability Theorems, Theorem 3.2 and Theorem 3.3 as well as some arithmetic reasoning. The proof of the step case is completed by using Theorem 3.4 along with some arithmetic reasoning.

Theorem 3.6: $\vdash \forall$ X p a b d n i Linit.
DB_MODEL X p a b d n Linit $\wedge$
i + 1 $\in$ [0,n] $\Rightarrow$
(lim (λ t. $\mathbb{P}\{$x|X t x = 0$\}$) =
$\frac{1}{\mathtt{SIGMA}\ \ (\lambda i.\ \mathtt{PROD}\ (1,\ i+1)\ (\lambda j.\ \frac{b\ (j-1)}{d\ j})\ (0,\ n+1)}$)

The proof of this theorem begins by rewriting the goal as lim (λt. $\mathbb{P}\{$x|X t x = 0$\}$) * SIGMA (λi. PROD (1, i + 1) (λj. $\frac{b_{j-1}}{d_j}$)) (0, n + 1) = 1.

Then we split the summation into two terms: $\frac{b_0}{d_1}$ and SIGMA (1, n + 1) (λi. PROD (1, i + 1) (λj. $\frac{b_{j-1}}{d_j}$)) (0, n + 1). The proof is then concluded by applying Theorems 3.3 and 3.5 and the probability additivity theorem and some real arithmetic reasoning. The proof details of the above theorem can be found in [LHT16].

Once these theorems verified, the limit probabilities of any state in this model can be calculated by instantiating the parameter n and transition probabilities a, b and d. Thus, it becomes unnecessary for the potential user to employ any numerical arithmetic to analyze the long-term behaviors of this model. The solution, shown in Equations (2a) and (2b), is mainly used to predict safety properties in the development of the population in a long period. This is used in various domains, such as statistics and biological.

Furthermore, when the birth-death coefficients are $b_i = \lambda$ and $d_i = \mu$ (λ and μ are constants) for all the i's in the state-space, the model described in Definition 3.2 represents a classical M/M/1 queueing system [Kle75] (in this case, the average inter arrival time becomes $\frac{1}{\lambda}$ and the average service time is $\frac{1}{\mu}$). For this particular case, our formally verified

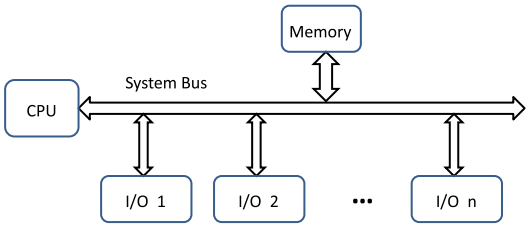

Fig. 2. Basic Computer Architecture

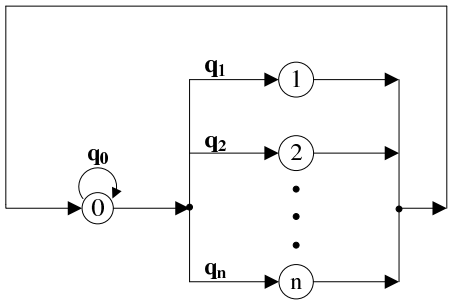

Fig. 3. A Discrete-Time Markov Chain Model of a Program

theorems can be directly applied for analyzing the ergodicity of M/M/1 queueing.

IV. Applications

In order to illustrate the usefulness of the developed Markov Chain formalization framework, we present in this section the formal performance analysis of two software applications.

A. Formal Analysis of Program Performance

The basic architecture of a modern multi-processor based computer system can be illustrated by Figure 2. Each processor in such a system is usually connected with a memory module and several input/output (I/O) ports. Usually, a main program is designed to control the requests from the devices connected to these I/O ports. Requests from various devices at the end of a CPU burst are independent from the past behavior.

Consider a program that manages a CPU with n I/O devices. It is assumed that the program will finish the execution phase at the end of a CPU burst with probability q_0 and the probability of requests from the device connected with the i^{th} I/O is q_i. Moreover, all devices are assumed to be available, i.e., $0 < q_i < 1$ (for $i = 0, 1, \cdots, n$) and $\sum_{i=0}^{n} q_i = 1$, where n is the number of the I/Os or the devices connected to the CPU. In [Tri02], the behavior of this program can be modeled as an aperiodic and irreducible discrete-time Markov chain, which is shown in Figure 3.

From this diagram, we can obtain the transition probability matrix and formally express it as a function in HOL as:

$$P = \begin{pmatrix} q_0 & q_1 & q_2 & \cdot & \cdot & \cdot & q_n \\ 1 & 0 & 0 & \cdot & \cdot & \cdot & 0 \\ & & & & & & \\ 1 & 0 & \vdots & \cdot & \cdot & \vdots & 0 \\ 1 & 0 & \cdot & \cdot & \cdot & 0 & 0 \end{pmatrix} \quad (4)$$

Definition 4.1:
```
⊢ ∀q t i j. pmatrix q t i j =
if (i = 0) then q j
else if (j = 0) then 1 else 0
```

In order to evaluate the performance of this program, we can prove some interesting properties of this system. First of all, we verify that there exists a steady-state probability for every state in the state-space. Then, we can prove that the steady-state vector satisfies $v_j = v_0 q_j$ (for all $j = 1, 2, \cdots, m$). Furthermore, the following two equations, which are usually used to analyze the long-term behaviors of a multi-processor, can be verified:

$$v_0 = \frac{1}{2 - q_0} \quad (5)$$

$$v_j = \frac{q_j}{2 - q_0} \quad (6)$$

which are the steady-state probabilities of visiting the CPU (corresponding to Equation (5)) and different devices (corresponding to Equation (6)) in the system.

Now, we first define this model as a predicate in higher-order logic:

Definition 4.2:
```
⊢ ∀ X p q n p₀.
PROGRAM_MODEL X p q n p₀ =
Aperiodic_MC X p
  ([0,n], POW [0,n]) p₀ (pmatrix q) ∧
Irreducible_MC X p
  ([0,n],POW [0,n]) p₀ (pmatrix q) ∧
(∀i. i∈[0,n]⇒0 < q i ∧ q i < 1) ∧
(SIGMA (λi. q i) [0,n] = 1)
```

where the first and second assumptions describe that the program's behavior can be modeled as an aperiodic and irreducible DTMC and the last two conjuncts constraint the probabilities.

Then, we prove that there exists steady-state probabilities for all states in the state-space:

Theorem 4.1:
```
⊢ ∀ X p q n p₀.
PROGRAM_MODEL X p q n p₀ ∧ 0 < n ⇒
(∀j. ∃u. (λt. ℙ{x|X t x = j}) → u)
```

The properties expressed using Equations (5) and (6) are verified as the following two theorems:

Theorem 4.2:
```
⊢ ∀ X p q n p₀.
PROGRAM_MODEL X p q n p₀ ∧ 0 < n ⇒
 (lim (λt. ℙ{x|X t x = 0}) = 1 / (2 − q 0))
```

Theorem 4.3:
```
⊢ ∀ X p q n p₀.
PROGRAM_MODEL X p q n p₀ ∧ 0 < n ∧
j∈[1, n] ⇒
 (lim (λt. ℙ{x|X t x = j}) = q j / (2 − q 0))
```

If we use probabilistic model checking to analyze the performance of this system, then the steady-state probabilities of visiting each device can only be obtained by solving a group of linear equations. Thus, if the system involves n devices, then the computations would increase linearly. In the case of

using simulation for analyzing this model, the final results will be obtained as a vector including many zeroes, which are not accurate enough (an event with very low probability will never be an impossible event). This is because if some q_i ($i \in [0, n]$) becomes very small (as the number of the devices increases) during the simulation process, the accuracy of the calculations is constrained by the underlying algorithms and the available computation resources.

As shown in Theorems 17 and 18, we were able to provide generic results. The HOL code for the above verification comprises of only around 300 lines of proof script and the reasoning was based on our foundational results, presented in the previous sections. Moreover, the verified generic results largely reduce the computation time for obtaining steady-state probabilities for the aperiodic and irreducible DTMCs. In fact, the steady-state probabilities computed based on the previous two theorems can also be used to interpret the average visiting time, for example, if the real-time interval is T, then the average number of visits to device j will be v_jT [Tri02] in the long run.

B. Formal Analysis of a Data Structure

In software engineering, resource usage is one of the major quality attributes of a software. For example, the amount of memory consumptions by certain data structures, e.g., a linear list, being manipulated in a program is usually of interest in evaluating the performance of this program. The amount of the occupied memory units can be regarded as the population in a discrete-time Birth-Death process, where the insertion of a data corresponds to the birth transmission, the release of a memory unit can be considered as the death transmission and the access of a memory unit represents that the system stays in a state. Assuming that this data structure in a program has a stable transition probability, which is independent of time, then the state diagram for this data structure can be depicted as Figure 4, where the transition probabilities are described as:

b_i = P("next operation is an insert" |
"current i units of memory is occupied")
d_i = P("next operation is a delete" |
"current i units of memory is occupied")

With the assumed stable transition probabilities, we have b_i = b ($i \geq 0$) and d_i = d ($i \geq 1$) in the process. We are interested in learning the probabilities of an overflow and underflow in a long run, which can be obtained by computing the steady state probabilities of the full-size of the accessible memory units. Also, we can predict the probability that all usable memory units are released in a long-run. In order to formally reason about this steady-state probability, we proceed by first formally describing the data structure behaviors by instantiating the discrete-time birth-death process in higher-order logic.

Definition 4.3:

```
⊢ ∀d. ra d = λn. if n=0 then d else 0;
⊢ ∀b. rb b = λn. b;
⊢ ∀d. rd d = λn. d
```

Then the following model can be used to describe the behavior of this data structure:

Definition 4.4:

```
⊢ ∀ X p b d m p0.
Data_Struc_MODEL X p b d m p0 =
DB_MODEL X p (ra d) (rb b) (rd d) m p0
```

as a discrete-time birth-death process where `b` and `d` are the *birth* and *death* transition probabilities, respectively, `m` denotes the amount of useable memory size, p_0 is a general initial distribution. Assuming that the potential number of the memory units (m) is very large ($m > 0$) for allocation in this model and the parameters satisfy `b` $<$ `d`, we can prove the existence of steady-state probabilities (v_i, $1 < i$) of this system by applying Theorem 3.1. Then, using Theorem 3.6, it is easy to verify the steady-state probability that all the memories are released in the long-run, i.e, $v_0 = \lim_{t\to\infty} \mathcal{P}r\ (X_t = 0)$ is given by

$$v_0 = \frac{1 - \frac{b}{d}}{1 - (\frac{b}{d})^m}$$

and it is verified as the following theorem in HOL.

Theorem 4.4:

```
⊢ ∀ X p d b m p0.
Data_Struc_MODEL X p b d m p0 ∧
b < d ∧ 0 < m ⇒
(lim (λ t. ℙ{x|X t x = 0}) = 1−b/d / 1−(b/d)^m)
```

The steady-state probability of i memory units required in such a model is

$$v_i = (\frac{b}{d})^i v_0$$

which is proved as the following theorem in HOL.

Theorem 4.5:

```
⊢ ∀ X p d b m p0.
Data_Struc_MODEL X p b d m p0 ∧
b < d ∧ 0 < m ⇒
(lim (λt. ℙ{x|X t x = i}) =
 (b/d)^i * lim (λt. ℙ{x | X t x = 0}))
```

Then, the probability of an overflow is given by

$$bv_m = b(\frac{b}{d})^m \frac{1 - \frac{b}{d}}{1 - (\frac{b}{d})^{m+1}} = \frac{d^{m+1} * (d - b)}{d^{m+1} - b^{m+1}}$$

which means that all memory units available for allocation are used and the probability of a further insertion occurring in a long-run is $\frac{d^{m+1}*(d-b)}{b*(d^{m+1}-b^{m+1})}$. This property is proved in a theorem as follows:

Theorem 4.6:

```
⊢ ∀ X p d b m p0.
Data_Struc_MODEL X p b d m p0 ∧
b < d ∧ 0 < m ⇒
(b*lim (λt. ℙ{x|X t x = m}) = d^(m+1)*(d−b) / d^(m+1)−b^(m+1))
```

Similarly, the probability of underflow represents the probability that a delete operation will occur when all available memory units are occupied and it is proved as:

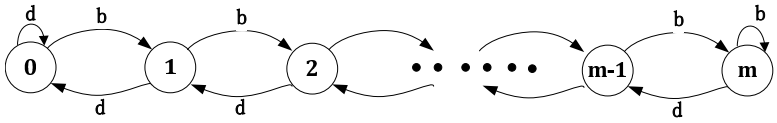

Fig. 4. The State Diagram of Data Structure Behavior

Theorem 4.7: $\vdash \forall$ X p d b m p_0. Data_Struc_MODEL X p b d m $p_0 \wedge$ b < d $\wedge$ 0 < m $\Rightarrow$ (lim (λt. $\mathbb{P}${x|X t x = 0}) = $\frac{b^{m-1}*(d-b)}{d^m-b^m}$)

Using simulation or probabilistic model checking to analyze this kind of data structure model would involve an enormous amount of computation time and memory. It is also obvious (from Theorem 4.7) that the computation may encounter some errors, like dividing by zero with an increase in the value of m (d and b are both between 0 and 1 and the power of such a small positive number tends to zero). These features are unacceptable while analyzing safety-critical systems. The proposed approach shows quite promising results in this context as it is capable of overcoming the above mentioned limitations, with around 500 lines of HOL code to verify these interesting properties about the given data structure.

V. Conclusion

This paper presents a methodology to formally analyze Markovian systems based on the formalization in higher-order logic of DTMCs and classified DTMCs with finite state-space. Due to the inherent soundness of theorem proving, our work guarantees accurate results, which is a very useful feature while analyzing stationary or long-run behaviors of a system associated with safety or mission-critical systems. We proposed an efficient approach to formalize the Discrete-time Birth-Death process and validate that a discrete IID random process with finite state-space is a DTMC. Moreover, we use the definitions and verified properties of classified DTMCs in analyzing the performance of a couple of software applications, i.e., a program controlling the CPU interactions with its connected devices and a data structure used in a program.

The paper provides a new method to formally analyze DTMCs with finite-state-space and avoid the state-explosion problem or the unacceptable computation time issue which are commonly encountered in model checking and simulation, respectively, for analysing the stationary properties of a safety-critical system with a large number of states. Hence, the presented work opens the door to a new and very promising research direction, i.e., integrating HOL theorem proving in the domain of analyzing DTMC systems and validating DTMC models.

References

[Bul06] J. Bulla. *Application of Hidden Markov Models and Hidden Semi-Markov Models to Financial Time Series*. PhD. thesis, University Gottingen, Germany, 2006.

[BW90] R. N. Bhattacharya and E. C. Waymire. *Stochastic Processes with Applications*. John Wiley & Sons, 1990.

[CPB08] E. A. Codling, M. J. Plank, and S. Benhamou. Random walk models in biology. *Journal of The Royal Society Interface*, 5(25):813–834, 2008.

[Dat08] S. Datta. *Probabilistic Approximate Algorithms for Distributed Data Mining in Peer-to-peer Networks*. University of Maryland, Baltimore County, USA, 2008.

[Fam65] Eugene F. Fama. Random Walks in Stock-Market Prices. *Financial Analysts Journal*, 21:55–59, 1965.

[Goo88] R. Goodman. *Introduction to Stochastic Models*. Benjamin/Cummings Pub. Co., 1988.

[Gor89] M.J.C. Gordon. Mechanizing Programming Logics in Higher-0rder Logic. In *Current Trends in Hardware Verification and Automated Theorem Proving*, Lecture Notes in Computer Science, pages 387–439. Springer, 1989.

[HÖ2] O. Häggström. *Finite Markov Chains and Algorithmic Applications*. Cambridge University Press, 2002.

[Kle75] L. Kleinrock. *Queueing Systems*, volume I: Theory. Wiley Interscience, 1975.

[LAHT13] L. Liu, V. Aravantinos, O. Hasan, and S. Tahar. Formal Reasoning about Classified Markov Chains in HOL. In *Interactive Theorem Proving*, volume 7998 of *LNCS*, pages 295–310. Springer, 2013.

[LAHT14] L. Liu, V. Aravantinos, O. Hasan, and S. Tahar. On the Formal Analysis of HMM Using Theorem Proving. In *Formal Methods and Software Engineering*, volume 8829 of *LNCS*, pages 316–331. Springer, 2014.

[LHT11] L. Liu, O. Hasan, and S. Tahar. Formalization of Finite-State Discrete-Time Markov Chains in HOL. In *Automated Technology for Verification and Analysis*, volume 6996 of *LNCS*, pages 90–104. Springer, 2011.

[LHT13a] L. Liu, O. Hasan, and S. Tahar. Formal Analysis of Memory Contention in a Multiprocessor System. In *Formal Methods: Foundations and Applications*, volume 8195 of *LNCS*, pages 195–210. Springer, 2013.

[LHT13b] L. Liu, O. Hasan, and S. Tahar. Formal Reasoning About Finite-State Discrete-Time Markov Chains in HOL. In *Journal of Computer Science and Technology*, volume 28, pages 217–231. 2013.

[LHT16] L. Liu, O. Hasan, and S. Tahar. *Formalization of Birth-Death and IID Processes in Higher-order Logic*. Technical report, http://hvg.ece.concordia.ca/Publications/TECH_REP/IIDBD_TR16/, ECE Department, Concordia University, November 2016.

[Mac98] D.J.C. MacKay. Introduction to Monte Carlo Methods. In *Learning in Graphical Models, NATO Science Series*, pages 175–204. Kluwer Academic Press, 1998.

[Map17] Maple. http://www.maplesoft.com, 2017.

[Mat17] Mathematica. www.wolfram.com, 2017.

[MHT11] T. Mhamdi, O. Hasan, and S. Tahar. Formalization of Entropy Measures in HOL. In *Interactive Theorem Proving*, volume 6898 of *LNCS*, pages 233–248. Springer, 2011.

[Par02] D. A. Parker. *Implementation of Symbolic Model Checking for Probabilitics Systems*. PhD thesis, University of Birmingham, UK, 2002.

[PRI17] PRISM. http://www.prismmodelchecker.org, 2017.

[PT00] K. G. Popstojanova and K. S. Trivedi. Failure Correlation in Software Reliability Models. volume 49, pages 37–48, 2000.

[Tri02] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. John Wiley & Sons, 2002.