Verifying the Evolution of Probability Distributions Governed by a DTMC

YoungMin Kwon and Gul Agha, *Fellow*, *IEEE*

Abstract—We propose a new probabilistic temporal logic, *iLTL*, which captures properties of systems whose state can be represented by *probability mass functions* (pmfs). Using iLTL, we can specify reachability to a state (i.e., a pmf), as well as properties representing the aggregate (expected) behavior of a system. We then consider a class of systems whose transitions are governed by a *Markov Chain*—in this case, the set of states a system may be in is specified by the transitions of pmfs from all potential initial states to the final state. We then provide a model checking algorithm to check iLTL properties of such systems. Unlike existing model checking techniques, which either compute the portions of the computational paths that satisfy a specification or evaluate properties along a single path of pmf transitions, our model checking technique enables us to do a complete analysis on the expected behaviors of large-scale systems. Desirable system parameters may also be found as a counterexample of a negated goal. Finally, we illustrate the usefulness of iLTL model checking by means of two examples: assessing software reliability and ensuring the results of administering a drug.

Index Terms—Probabilistic model checking, linear temporal logic, Discrete Time Markov Chain, pharmacokinetics.

1 Introduction

Markov processes have long been used as a probabilistic model for real-world systems in a number of disciplines, such as economics, biology, chemistry, and engineering. The popularity of Markov process can be explained by two useful properties of Markov processes, the *memoryless property* and the *unique limiting probability distribution property*.[1] The memoryless property, often referred to simply as the *Markov property*, simplifies the computation of the future probability distributions over the possible *concrete states* of a system.[2] The unique limiting probability distribution property simplifies the *steady state analysis* of a system. Based on these two properties, many useful applications of steady state analysis have been carried out: queueing systems [19], the performance analysis of communication protocols [38], [36], and the reliability analysis of software systems [32].

Compared to steady state analysis, *transient state analysis* has drawn relatively less attention. We believe this is partly because transient state analysis is difficult without the aid of computer automation: While the steady state of a system can be determined simply by the structure of the Markov process governing its transitions, transient states depend on numerous possible initial conditions. Nonetheless, it is often necessary to analyze the transient state of a system. The intuition behind understanding the importance of transient analysis is as follows: Observe that the normal function of a system may be represented by its steady state, while transient states occur during perturbations to a system. We conjecture that *anomalous* behaviors are more likely to occur in the course of perturbations to a system than during its normal operations. In fact, there are systems in which the study of transient states is of primary interest, for example, in *Pharmacokinetics* in the states of a system during the time when the drug is still being actively metabolized, i.e., during transient states when the drug concentration levels are changing.

We propose a probabilistic temporal logic called *iLTL* and provide a model checking algorithm for iLTL formulas when the system meets certain requirements.[3] Specifically, one can specify properties of the transitions of *probability mass functions* (pmfs) using iLTL. We provide a new model checking algorithm which allows us to check the iLTL properties of systems whose behavior is governed by *Discrete Time Markov Chains* (DTMCs). Recall that model checking is a state exploration technique; our model checking algorithm checks that the specification is satisfied over all pmfs, starting from each possible initial pmf.

To the best of our knowledge, no other technique has attempted to check pmf transitions from all possible initial pmfs. Existing approaches either analyze the portions of computational paths that satisfy a given specification [16], [2] or they analyze a single path corresponding to the pmf

1. Not every Markov process has a unique limiting probability distribution, but it is a commonly used assumption and many interesting Markov chains have this property.

2. In this paper, we use the term *state* for probability distributions, as opposed to the concrete states represented by each probability distribution. In particular, the existence of a steady state implies that there is a limiting probability distribution.

- *Y. Kwon is with Microsoft Corporation, One Microsoft Way, Redmond, WA 98052. E-mail: ykwon4@cs.uiuc.edu.*
- *G. Agha is with the Department of Computer Science, University of Illinois at Urbana Champaign, 201 N Goodwin Ave, Urbana, IL 61801. E-mail: agha@cs.uiuc.edu.*

Manuscript received 22 July 2008; revised 29 Apr. 2009; accepted 1 Sept. 2009; published online 19 Aug. 2010.
Recommended for acceptance by S. Donatelli.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2008-07-0229.
Digital Object Identifier no. 10.1109/TSE.2010.80.

3. We first described the logic and basic model checking algorithm in a conference paper [21]. The software reliability example appeared in workshop [24]. The analysis of multiple Markov Chains, the Pharmacokinetics example, and some of the details in this journal paper have not been previously published.

transitions from a given initial pmf [9]. Observe that the transitions of a pmf for a Markov process are linear, rather than branching. In other words, given an initial pmf, its trailing pmf predictions are deterministic. Checking a single linear path might be relatively straightforward; what makes iLTL model checking useful is that it evaluates on pmf transitions from *all* possible initial pmfs.

Because pmf transitions of Markov processes are linear and the choice of initial pmfs is nondeterministic, *Linear Temporal Logic* (LTL) [27] has a suitable set of temporal connectives to describe the evolution of such systems. iLTL shares the logical and the temporal operators of LTL, which makes the logic expressive and intuitive in its interpretation over linear paths. However, there are differences between LTL and iLTL: The states and the computational paths of LTL are the concrete states of the model and their sequences, whereas the states whose properties are represented by iLTL are the pmfs generated by the pmf transitions. As mentioned earlier, each pmf represents a probabilistic distribution over the concrete states of a system.

The atomic propositions of iLTL are equalities or inequalities about *expected* rewards. Each equality defines a hyperplane, and each inequality defines a half space in the pmf space. By taking the intersections of these subspaces, one can specify a convex region, and by taking unions or complements of the convex regions, one can specify arbitrary regions in the pmf space. Moreover, iLTL's temporal operators enable one to specify how these regions should change over time. In the model checking process, the regions in a future pmf spaces are transformed into their corresponding regions in the initial pmf space using a *normal form*. Observe that the nondeterminism in the choice of initial pmf has *uncountably* many choices. These uncountably many choices in the initial pmfs are resolved by the feasibility checking of the regions in the collapsed time frame.

iLTL model checking enables new forms of system analysis that are not possible using existing approaches. Specifically, iLTL provides three advantages over current analysis methods:

1. The iLTL specification on pmf transitions enables analysis of large-scale systems. Suppose that there is a system of 100 nodes, each of which has three possible states. This system has 3^{100} (global) concrete states, a number making it impossible to even enumerate the concrete states. For such systems, iLTL allows us to specify properties based on statistical abstractions. For example, iLTL can be used to specify requirements for availability, reliability, energy consumption, etc.
2. Our model checking algorithm for iLTL specifications covers all initial states to the final steady state, providing completeness in system analysis. By way of contrast, simulation-based analysis techniques such as Monte Carlo simulations [5] do not guarantee the nonexistence of a counterexample. Similarly, analysis techniques based on a single initial state do not provide completeness. In particular, using these existing techniques, one cannot completely evaluate systems whose state estimates are given as interval estimates.
3. Using our model checking algorithm, users can check not only the *satisfiability* of the specification, they can also *compute desirable parameters*. The counterexample generated by our model checking is an initial pmf which violates a specification. Thus, by checking for counterexamples to the negation of a goal, one can compute desirable initial pmfs. For example, in the drug administration case study (Section 6.2), we compute the desired dose as a counterexample of a negated goal.

The iLTL model checking tool, *iLTLChecker* [1], [22], is available from http://osl.cs.uiuc.edu/~ykwon4/cgi/iLTL.html.

2 Related Works

The concurrent systems are often modeled by a labeled transition system called *Kripke structure* [18]; the computational paths of a Kripke structure result from nondeterministic choices inherent in a concurrent execution. Temporal logics such as *Linear Temporal Logic* [27], [17], *Computational Tree Logic* (CTL) [12], and CTL* [13] have been developed to describe the properties of concurrent systems. Of these, CTL* is the most expressive: It allows unrestricted use of temporal operators and path quantifiers. CTL is a restricted form of CTL* in which every temporal operator is prefixed with a path quantifier. Because of the restriction on temporal operators, the truth value of each subformula can be recursively determined at the nodes of the Kripke structure, which leads to an efficient, polynomial time model checking algorithm. LTL is another restricted form of CTL*; LTL does not allow any path quantifiers except for an implicit universal path quantifier at the root of the specification. Thus, LTL specifies properties that are true for all paths of a Kripke structure. However, unlike CTL, there is no efficient polynomial time algorithm for LTL model checking.

If we add probability to a Kripke structure, then there are probabilistic extensions to the temporal logic, such as PCTL [16], PCTL* [3], and *Continuous Stochastic Logic* (CSL) [4], [20]. These logics replace the universal or the existential path quantifiers of the traditional temporal logic with the probability of satisfaction or the probability of a violation. That is, these logics build a probability space on the set of all computational paths and assign a probability measure to the events of paths that satisfy each subformula of the specification. The path quantifiers of CTL and CTL* are replaced with comparisons about the probability that the subformula will be satisfied.

In PCTL-like logics, probabilities do not represent the likelihood that a process is in a certain *state*, but rather the likelihood that a *computational path* satisfies a given specification. Consider a simple example. Suppose a probabilistic automaton M consists of two states A and B; in each state it moves to other state with probability 1. In a PCTL-like logic, a specification like *the probability of* M *being in* A *state is always larger than 0.3* is false because, on each step of a path, M changes state: $\mathrm{P}[M = A]$ is either 0 or 1. Now, suppose that there are 100 processes such that initially 50 of them are in state A and the other 50 are in state B. Now 50 percent of processes are always of the processes in state A. Thus, it is reasonable to assert that depending on the initial probability distribution, the specification maybe satisfiable. To specify

this property in a PCTL-like logic, we have to represent the transitions of multiple processes by building a Cartesian product consisting of multiple Markov processes. However, the resulting state space grows exponentially in the number of processes.

Two approaches to address the above state explosion problem (and save storage) are lumping symmetric states [8] and using efficient data structures such as Multivalued Decision Diagram [30] or Matrix Diagram [11]. Even with these approaches, specifying the statistical properties of very large state space systems is not tractable. For example, the concentration levels of a drug at different organs change as the drug molecules move from one organ to another. Because each drug molecule can be modeled as a DTMC, millions of them will define a system where the total number of (global) states is the number of local states raised to its millionth power.

Constructing DTMC or *Continuous Time Markov Chain* (CTMC) models for the real system could be challenging, especially when the system states have complex interactions. One approach to constructing such a probabilistic model from abstract descriptions about the system is *Stochastic PEPA* [9]. In Stochastic PEPA, the system behaviors are described by a process algebra in which actions have been extended with their rates. From this description, a CTMC can be obtained on which an ODE-based quantitative evaluation is possible. In [9], this method is applied to a cell biology problem. However, the analysis carried out only from a specified state. We believe iLTL could provide an insight about a system because it analyzes behavior from all possible initial states.

The Markov model of a real system can be obtained by sampling the system. For example, in [23], we estimated a DTMC from the samples of a *Wireless Sensor Network* (WSN) consisting of 90 nodes and measuring its behavior. iLTL has been used for the quantitative evaluations of two sorts: the performance of WSNs [23], [29] and the reliability of software systems [24]. Two important extensions of iLTL have been defined: Inspired by how state is handled in iLTL, the *Linear Spatial-Superposition Logic* is a logic used to specify and reason about properties of *Cardiac Myocytes* [15]. Another extension to iLTL, *Linear Temporal Logic for Control* (LTLC), enables us to specify the properties of *general linear systems* [25]: Here, the pmf transitions of a Markov chain can be regarded as a homogeneous output of a linear system with zero external input. LTLC allows finite sequence of inputs to be used in model checking. The external input in LTLC specifications accounts for nondeterminism that the DTMC models of iLTL do not have.

3 Markov Chain Model

We first define *Discrete Time Markov Chain* (DTMC) models and briefly describe some of their relevant properties. We then define the *Markov Reward Model* (MRM). The atomic propositions of iLTL involve comparisons of *expected rewards* based on this model.

3.1 Discrete Time Markov Chain Model

Let S be a countable set of states and let $\mathcal{P} = (\Omega, \mathcal{F}, \mathrm{P})$ be a probability space, where Ω is a sample description space, $\mathcal{F} \subseteq 2^{\Omega}$ is a Borel σ field of events, and $\mathrm{P} : \mathcal{F} \to [0,1]$ is a probability measure. A *Discrete Time Markov Chain* is a function $X : \mathbb{N} \times \Omega \to S$ satisfying the following memoryless property [31]:

$$\begin{aligned}\mathrm{P}[X(t) = s_t \mid X(t-1) = s_{t-1}, \ldots, X(0) = s_0] \\ = \mathrm{P}[X(t) = s_t \mid X(t-1) = s_{t-1}],\end{aligned}$$

where $\mathrm{P}[X(t) = s]$ stands for $\mathrm{P}(\{\zeta \in \Omega : X(t, \zeta) = s\})$. A DTMC $X : \mathbb{N} \times \Omega \to S$ may be represented by a *labeled directed graph* called a *State Transition Diagram* (STD), where:

- The nodes of the graph represent the states S. Note that by state here, we refer to the concrete state of an element in the system.
- The directed edges of the graph represent possible transitions from the state labeled with the conditional probabilities of the corresponding transition [36]. Thus, there is an edge $\{(s_j, s_i)$ if $\mathrm{P}[X(t) = s_i \mid X(t-1) = s_j] \neq 0\}$.

In this paper, we represent the STD $\mathcal{T}_X$ for a Markov process X by a tuple:

$$\mathcal{T}_X = (S,\ \mathbf{M}),$$

where $S = \{s_1, \ldots, s_n\}$ is a set of states and $\mathbf{M} \in \mathbb{R}^{n \times n}$ is a probability transition matrix such that $\mathbf{M}_{ij} = \mathrm{P}[X(t) = s_i | X(t-1) = s_j]$.

To simplify the explanation, we define a *probability vector function* $\mathbf{x} : \mathbb{N} \to \mathbb{R}^{n \times 1}$ such that

$$\mathbf{x}(t) = [\ \mathrm{P}[X(t) = s_1]\ \cdots\ \mathrm{P}[X(t) = s_n]\ ]^T. \tag{1}$$

The probability vector function $\mathbf{x}$ corresponds to the computational path of classical LTL which at time t is in the state represented by the pmf vector $\mathbf{x}(t)$. Using the probability vector function, we can rewrite the prediction equation

$$\begin{aligned}\mathrm{P}[X(t) = s_i]\ = \sum_{j=1}^{n} \mathrm{P}[X(t) = s_i | X(t-1) = s_j] \\ \cdot\ \mathrm{P}[X(t-1) = s_j]\end{aligned}$$

concisely as follows:

$$\mathbf{x}(t)\ =\ \mathbf{M} \cdot \mathbf{x}(t-1)\ =\ \mathbf{M}^t \cdot \mathbf{x}(0). \tag{2}$$

Because $\mathbf{M}$ is a probability transition matrix, its elements are nonnegative, the sum of each column adds up to one, and the absolute values of its eigenvalues are not larger than 1. The condition about the column sum ensures that 1 is an eigenvalue of all probability transition matrices: The matrix $\mathbf{M} - \mathbf{I}$ is a singular matrix because the sum of all row vectors is the zero vector [37].

Suppose that the probability matrix $\mathbf{M}$ is diagonalizable[4] such that $\mathbf{M} = \mathbf{P} \cdot \Lambda \cdot \mathbf{P}^{-1}$. Then, we can easily compute $\mathbf{M}^t$ as $\mathbf{P} \cdot \Lambda^t \cdot \mathbf{P}^{-1}$. Observe that Λ^t is a diagonal matrix of λ_i^t, where λ_i is the ith diagonal element of Λ. Similarly,

4. The matrices that are not diagonalizable are called *defective* matrices; they lack the full set of independent eigenvectors. However, the full set of eigenvectors can be easily recovered by slightly changing the matrix [37]. Because of the continuous nature of probabilities, a slight change in the matrix has a negligible effect on behavior.

$e^{\mathbf{M}} = \mathbf{P} \cdot e^{\Lambda} \cdot \mathbf{P}^{-1}$, where e^{Λ} is a diagonal matrix of e^{λ_i}. Because $\lim_{t\to\infty} \lambda_i^t = 0$ if $|\lambda_i| < 1$, the limiting pmf $\mathbf{x}(\infty)$ is determined solely by the columns of $\mathbf{P}$ that correspond to the eigenvalue 1. Suppose that there is an eigenvalue $\lambda \neq 1$ such that $|\lambda| = 1$. Then, because λ^t changes phases without being diminished, λ is an oscillating mode that prevents $\mathbf{x}(t)$ from converging to a limiting pmf. Note that if more than one eigenvalue is equal to 1, the limiting pmf is not unique. For example, let $\mathbf{M} = \left[\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right]$ and $\mathbf{M}' = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$. Then, for any $p \in [0,1]$, $\mathbf{M}$ alternates pmfs between $[p, 1-p]^T$ and $[1-p, p]^T$ forever because it has an eigenvalue -1. Also, because $\mathbf{M}'$ has two eigenvalues of 1, for any $p \in [0,1]$, the pmf $[p, 1-p]^T$ can be a limiting pmf. However, if the absolute values of all eigenvalues is strictly less than 1 except for the 1, the DTMC has a unique limiting pmf.

It should be noted that some systems are more naturally represented by *Continuous Time Markov Chains*. Fortunately, conversions between DTMCs and CTMCs are possible; the discrete sequence of pmfs $\mathbf{x}(t)$ with $t \in \mathbb{N}$ can be obtained by periodically sampling the continuous pmf transitions of a CTMC. This conversion is useful when building a bigger system by combining multiple subsystems [26]. However, DTMCs with different sampling periods cannot be combined directly. There are techniques, such as Stochastic PEPA [9], that can build a CTMC out of abstract descriptions of a system. Suppose $(S, \mathbf{R})$ is a STD of a CTMC, where $\mathbf{R}$ is a rate matrix such that

$$\frac{d}{d\tau}\mathbf{x}(\tau) = \mathbf{R} \cdot \mathbf{x}(\tau), \quad \tau \in \mathbb{R}.$$

By solving the differential equation, we get $\mathbf{x}(\tau) = e^{\mathbf{R}\cdot\tau} \cdot \mathbf{x}(0)$. Thus, the discrete sequence $\mathbf{x}(T \cdot t)$ can be obtained from the DTMC whose STD is $(S, e^{\mathbf{R}\cdot T})$, where T is a sampling period. The main challenge is to pick an appropriate sampling period.

3.2 Multiple Discrete Time Markov Chain Model

When evaluating the performances of a system, we often compare the system against one or more reference systems. For this comparison, we need to model parallel pmf transitions of multiple systems. Fortunately, such parallel behaviors can be simply modeled by taking a disjoint union of the multiple DTMCs.

A *multiple DTMC* model is a set of DTMCs. Suppose that there is a set of n DTMCs $\{X^{(1)}, \ldots, X^{(n)}\}$, where $X^{(i)}$ is a DTMC with $\mathcal{T}_{X^{(i)}} = (S^{(i)}, \mathbf{M}^{(i)})$ for $i = 1, \ldots, n$. If we assume that the set of states $S^{(i)}$ is disjoint[5] ($S^{(i)} \cap S^{(j)} = \emptyset$ if $i \neq j$), then we can use the STD representation of the single DTMC for the multiple DTMC model $\mathcal{M}$ as well: $\mathcal{T}_{\mathcal{M}} = (S, \mathbf{M})$, where

$$S = S^{(1)} \cup \cdots \cup S^{(n)}, \quad \mathbf{M} = \begin{bmatrix} \mathbf{M}^{(1)} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{M}^{(n)} \end{bmatrix}.$$

In fact, $\mathcal{M}$ can be regarded as a DTMC and can benefit from the useful results of DTMC theories. The block diagonal matrix $\mathbf{M}$ is a probability transition matrix because each element is nonnegative and each column adds up to one. However, this matrix does not transfer a single pmf; it transfers an array of pmfs. With this fact in mind, we extend the definition of the probability vector function $\mathbf{x} : \mathbb{N} \to \mathbb{R}^{|S|\times 1}$ as follows:

$$\mathbf{x}(t) = \left[\mathbf{x}^{(1)}(t)^T, \ldots, \mathbf{x}^{(n)}(t)^T\right]^T.$$

Observe that $\mathbf{x}(t)$ is not a pmf: Its sum adds up to n and it is structured such that the partial sums corresponding to each individual probability vector add up to one. The extended probability vector function $\mathbf{x}$ defines the parallel pmf transitions of individual DTMCs, and its value at time t is a snapshot of the parallel systems; the extended $\mathbf{x}$ and $\mathbf{x}(t)$ correspond to the computational path and the state of the classical LTL. With this extended definition, the prediction equation (2) still holds for the multiple DTMC model:

$$\mathbf{x}(t) = \mathbf{M}^t \cdot \mathbf{x}(0). \tag{3}$$

Because we extensively use the eigenvalues and the eigenvectors of $\mathbf{M}$ and $\mathbf{M}^{(i)}$, we index them by their absolute values so that we can easily reference them: We name the eigenvalues of $\mathbf{M}^{(i)}$ as $\lambda_1^{(i)}, \ldots, \lambda_{|S^{(i)}|}^{(i)}$ such that $|\lambda_1^{(i)}| \geq \cdots \geq |\lambda_{|S^{(i)}|}^{(i)}|$ and the eigenvalues of $\mathbf{M}$ as $\lambda_1, \ldots, \lambda_{|S|}$, where $|\lambda_1| \geq \cdots \geq |\lambda_{|S|}|$. We also designate the common eigenvalue 1 of all Markov probability matrices as $\lambda_1^{(i)}$ for $\mathbf{M}^{(i)}$ and $\lambda_1, \ldots, \lambda_n$ for $\mathbf{M}$. We call the eigenvectors corresponding to $\lambda_a^{(i)}$ and λ_a, respectively, $\mathbf{v}_a^{(i)}$ and $\mathbf{v}_a$.

Observe that $\mathcal{M}$ has a unique limiting state if $|\lambda_{n+1}| < 1$: Because $|\lambda_2^{(i)}| < 1$ for $i = 1, \ldots, n$, each individual DTMC $X^{(i)}$ has a unique limiting pmf $\mathbf{x}^{(i)}(\infty) = c_i \cdot \mathbf{v}_1^{(i)}$, where $c_i = 1/(\sum_{j=1}^{|S^{(i)}|} \mathbf{v}_{1\ j}^{(i)})$. Thus, the unique limiting state of $\mathcal{M}$ is $\mathbf{x}(\infty) = [\mathbf{x}^{(1)}(\infty)^T, \ldots, \mathbf{x}^{(n)}(\infty)^T]^T$.

3.3 Markov Reward Model

A *Markov Reward Model* is a Markov chain extended with a *reward function* that associates reward values with states [10]. The reward associated with a state can be regarded as an earned value when a process visits that state. As we show later in this section, many interesting properties can be expressed in MRM.

We represent a Markov reward model as a triple $(S, \mathbf{M}, r)$, where $(S, \mathbf{M})$ is a STD of a DTMC and $r : S \to \mathbb{R}$ is a reward function. We extend the definition of MRM to the multiple DTMC model by taking S as the domain of r. Because r is defined across the states of individual DTMCs, we can compare different DTMCs together. In this paper, we consider only the time invariant rewards which can be represented by a constant row vector $\mathbf{r} \in \mathbb{R}^{1\times|S|}$ as follows:

$$\mathbf{r}_i = r(s_i) \quad \text{for} \quad i = 1, \ldots, |S|.$$

The expected reward at time t can be written as

$$\mathrm{E}[r](t) \;=\; \sum_{i=1}^{n} \sum_{s\in S^{(i)}} r(s) \cdot \mathrm{P}\left[X^{(i)}(t) = s\right] \;=\; \mathbf{r} \cdot \mathbf{x}(t).$$

The atomic propositions of iLTL are comparisons between the expected rewards. Many interesting properties about DTMCs can be expressed in this form. For example, comparisons between two probabilities can be written as follows:

5. These assumptions can be easily achieved by renaming, using fresh names, the states whose names clash.

$$\mathrm{P}\big[X^{(i)}(t) = s_1\big] > \mathrm{P}\big[X^{(j)}(t) = s_2\big] \Leftrightarrow \mathrm{E}[r](t) > 0,$$

where $r(s_1) = 1$, $r(s_2) = -1$, and $r(s) = 0$ for the other states. Future probabilities can also be expressed as an expected reward:

$$\begin{aligned}\mathrm{P}\big[X^{(k)}(t) = s_i\big] &= \sum_{s_j \in S} r(s_j) \cdot \mathrm{P}\big[X^{(k)}(0) = s_j\big], \text{where } r(s_j)\\ &= \big(\mathbf{M}^t\big)_{ij}.\end{aligned}$$

When evaluating the performances of Markov processes, accumulated rewards are often computed. For example, we may be interested in the expected total energy required to complete an application. Because accumulated rewards are frequently used, we define a special operator Q such that:

$$\mathrm{Q}[X(t) = s] = \sum_{\tau=t}^{\infty} \mathrm{P}[X(\tau) = s].$$

With this operator, the accumulated expected rewards can be expressed as simply as

$$\sum_{\tau=t}^{\infty} \sum_{s \in S} r(s) \cdot \mathrm{P}[X(\tau) = s] = \sum_{s \in S} r(s) \cdot \mathrm{Q}[X(t) = s].$$

Note that the sum does not always exist. If the reward vector $\mathbf{r}$ is not orthogonal to the limiting state vector $\mathbf{x}(\infty)$, then the sum diverges. Thus, in this paper, we restrict the use of Q operator to the cases where $\mathbf{r} \cdot \mathbf{x}(\infty) = 0$. Given this restriction, the accumulated reward can be expressed as an expected reward as follows: Let $\mathbf{P} \cdot \Lambda \cdot \mathbf{P}^{-1}$ be a diagonalization of the probability matrix $\mathbf{M} \in \mathbb{R}^{|S| \times |S|}$ and let $\mathbf{r}$ be a reward vector such that $\mathbf{r} \cdot \mathbf{x}(\infty) = 0$. Then,

$$\begin{aligned}\sum_{\tau=t}^{\infty} \mathbf{r} \cdot \mathbf{x}(\tau) &= (\mathbf{r} \cdot \mathbf{Q}) \cdot \mathbf{x}(t), \text{where}\\ \mathbf{Q} &= \mathbf{P} \cdot \begin{bmatrix} s(\lambda_1) & & 0 \\ & \ddots & \\ 0 & & s(\lambda_{|S|}) \end{bmatrix} \cdot \mathbf{P}^{-1},\\ s(\lambda_i) &= \begin{cases} \frac{1}{1-\lambda_i}, & \text{if } \lambda_i \neq 1,\\ 0, & \text{otherwise.}\end{cases}\end{aligned}$$

Because $\mathbf{r}$ is orthogonal to the steady state vector, we can safely replace the summation for $\lambda_i = 1$ with 0. The Q operator can also be defined as an expected reward as follows:

$$\mathrm{Q}[X(t) = s_i] = \sum_{j=1}^{|S|} \mathbf{Q}_{ij} \cdot \mathrm{P}[X(t) = s_j].$$

In case of transient states, where $\mathrm{P}[X(\infty) = s_i] = 0$, we can write $\sum_{\tau=t}^{\infty} \mathrm{P}[X(\tau) = s_i] = \mathrm{Q}[X(t) = s_i]$.

4 iLTL: A Probabilistic Temporal Logic

We now describe the syntax and the semantics of iLTL. As mentioned earlier, iLTL has the same logical and temporal operators as LTL, but its specifications are over states that are pmfs, whereas states in LTL are elements of a Kripke structure. In fact, in an LTL for a DTMC, states would correspond to what we called concrete states. Note that the cardinality of the pmf state space is *uncountable*. The atomic propositions of iLTL are comparisons between expected rewards; their values are evaluated, as needed, in the pmf space (instead of being predefined as in LTL). We define a *normal form* of atomic propositions. Using the normal form enables us to transform atomic propositions about future pmfs into atomic propositions about the initial pmfs.

4.1 Syntax

The syntax of an iLTL formula ψ is as follows:

$$\begin{aligned}\psi ::= {} & \mathtt{T} \mid \mathtt{F} \mid ap \mid (\ \psi\ ) \mid\\ & \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \psi \rightarrow \psi \mid \psi \leftrightarrow \psi \mid\\ & \mathtt{X}\ \psi \mid \psi\ \mathtt{U}\ \psi \mid \psi\ \mathtt{R}\ \psi \mid \Box\ \psi \mid\ \diamond\ \psi\\ ap ::= {} & exp \bowtie exp\\ exp ::= {} & mexp \mid mexp + exp \mid mexp - exp\\ mexp ::= {} & num \mid term \mid num \cdot term\\ term ::= {} & pop[proc = state] \mid pop[proc(nat) = state]\\ pop ::= {} & \mathtt{P} \mid \mathtt{Q}\\ \bowtie ::= {} & < \mid \leq \mid = \mid \geq \mid >,\end{aligned}$$

where *proc* is a DTMC, *state* is its state, *num* is a real number, and *nat* is a natural number. In *term*, we introduce a time offset in the syntax, $pop[proc(nat) = state]$, to refer to future probabilities. Because the Q operator can be expressed as a Markov reward form, with a little abuse of notation we write "$\sum r_i \cdot \mathrm{P}[X^{(i)}(t_i) = k_i] \bowtie c$" as a representative syntax for the atomic propositions *ap*. We call the set of all atomic propositions *AP*.

4.2 Normal Form

Before we explain the semantics of iLTL, we define the *normal form* of atomic propositions. In the normal form, atomic propositions about future pmfs are converted to predicates about initial pmfs. The conversion is explained by helper functions $\mathbf{n}_{term}$, $\mathbf{n}_{exp}$, and $\mathbf{n}$ that compute expected reward vectors, and $\mathbf{a}$ and $\mathbf{a}_{set}$ that convert atomic propositions to their normal forms.

Let $\mathcal{M}$ be a multiple DTMC model with $\mathcal{T}_{\mathcal{M}} = (S, \mathbf{M})$, then constants and probabilities with time offsets are converted to a reward vector by $\mathbf{n}_{term} : (S \cup \mathbb{R}) \times \mathbb{N} \rightarrow \mathbb{R}^{1 \times (|S|+1)}$ defined as

$$\mathbf{n}_{term}(k, t) = \begin{cases} \big[\big(\mathbf{M}^t\big)_{i^*}\ 0\big], & \text{if } k \in S \text{ is } s_i,\\ [\mathbf{0}\ k], & \text{if } k \in \mathbb{R},\end{cases}$$

where $\big(\mathbf{M}^t\big)_{i^*}$ is the ith row of $\mathbf{M}^t$. Let $\mathbf{x}' : \mathbb{N} \rightarrow \mathbb{R}^{(|S|+1) \times 1}$ be $\mathbf{x}'(t) = [\mathbf{x}(t)^T, 1]^T$, and let X be a DTMC of $\mathcal{M}$, then

$$\mathbf{n}_{term}(k, t) \cdot \mathbf{x}'(0) = \begin{cases} \mathrm{P}[X(t) = k], & \text{if } k \in S,\\ k, & \text{if } k \in \mathbb{R}.\end{cases}$$

Let *ap* be the atomic proposition "$\sum_{i=1}^{m} r_i \cdot \mathrm{P}[X^{(i)}(t_i) = k_i] \bowtie c$," then the function $\mathbf{n}_{exp} : AP \rightarrow \mathbb{R}^{1 \times (|S|+1)}$ which converts the *exps* of *ap* to a reward vector is defined as:

$$\mathbf{n}_{exp}(ap) = \left(\sum_{i=1}^{m} r_i \cdot \mathbf{n}_{term}(k_i, t_i)\right) - \mathbf{n}_{term}(c, 0).$$

With $\mathbf{n}_{exp}$, the following holds:

$$\sum_{i=1}^{m} r_i \cdot \mathrm{P}\left[X^{(i)}(t_i) = k_i\right] \bowtie c$$
$$\Leftrightarrow \ \mathbf{n}_{exp}\left(\text{“}\sum_{i=1}^{m} r_i \cdot \mathrm{P}\left[X^{(i)}(t_i) = k_i\right] \bowtie c\text{”}\right) \cdot \mathbf{x}'(0) \bowtie 0.$$

The extra dimension in the extended state vector $\mathbf{x}'$ and the conversion function $\mathbf{n}_{term}$ can be naturally introduced by adding a single state DTMC to $\mathcal{M}$ such that $\mathcal{T}_{\mathcal{M}'} = (S \cup \{s\}, \mathbf{M}')$, where $\mathbf{M}' = \left[\begin{smallmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{0} & 1 \end{smallmatrix}\right]$ and s is a fresh state. Because the extra dimension in $\mathbf{n}_{term}$ and $\mathbf{x}'$ can be explained by a multiple DTMC model, we drop the prime mark when it is obvious from the context.

We define the function $\mathbf{n} : AP \times \mathbb{N} \to \mathbb{R}^{(|S|+1)\times 1}$ to introduce the time index into the conversion as follows:

$$\mathbf{n}(ap, t) = \mathbf{n}_{exp}(ap) \cdot \mathbf{M}'^{t}.$$

With the conversion function, the following holds.

$$\sum_{i=1}^{m} r_i \cdot \mathrm{P}\left[X^{(i)}(t + t_i) = k_i\right] \bowtie c$$
$$\Leftrightarrow \ \mathbf{n}_{exp}\left(\text{“}\sum_{i=1}^{m} r_i \cdot \mathrm{P}\left[X^{(i)}(t_i) = k_i\right] \bowtie c\text{”}\right) \cdot \mathbf{x}(t) \bowtie 0$$
$$\Leftrightarrow \ \mathbf{n}\left(\text{“}\sum_{i=1}^{m} r_i \cdot \mathrm{P}\left[X^{(i)}(t_i) = k_i\right] \bowtie c\text{”},\ t\right) \cdot \mathbf{x}(0) \bowtie 0.$$

Observe that the reward vector of the second line is evaluated at the state at time t, and the reward vector of the third line is evaluated at the initial state. Using the normal form, we convert the satisfiability problems of $\mathbf{x}(t)$ for $t > 0$ into those of $\mathbf{x}(0)$. That is, after this conversion, we can disregard the probability transition dynamics of DTMCs in the model checking process.

Finally, we define a conversion function $\mathbf{a} : AP \times \mathbb{N} \to AP$ that converts an *ap* to its normal form. Using the λ notation,

$$\mathbf{a}(ap, t) = \lambda \mathbf{z} \,.\, (\mathbf{n}(ap, t) \cdot \mathbf{z} \bowtie 0),$$

where $\mathbf{z}$ is an array of pmfs. We also define $\mathbf{a}_{set} : 2^{AP} \times \mathbb{N} \to 2^{AP}$ so that the normalization can be applied to a set of atomic propositions

$$\mathbf{a}_{set}(A, t) = \{\mathbf{a}(ap, t) : ap \in A\},$$

and write $\mathbf{a}_{set}(A, t)(\mathbf{z})$ for $\bigwedge_{ap \in A} \mathbf{a}(ap, t) \cdot \mathbf{z}$.

4.3 Semantics

An iLTL formula is comprised of atomic propositions, logical connectives, $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$, and temporal connectives, $\mathtt{X}$, $\mathtt{U}$, $\mathtt{R}$, $\Box$, $\diamond$. An atomic proposition of iLTL is a comparison between the expected rewards. The meaning of an atomic proposition at any given time t is whether the comparison at the time satisfies the usual meaning of $\bowtie$. In the normal form, this comparison is transformed into a comparison about expected rewards at the initial time step.

The meaning of logical operators is as follows: $\neg\psi$ is true if and only if ψ is false; $\psi \vee \phi$ is true if and only if ψ or ϕ are true; and $\psi \wedge \phi$ is true if and only if ψ and ϕ are both true.

$\mathcal{M}, \mathbf{z} \models \mathtt{T}$		
$\mathcal{M}, \mathbf{z} \not\models \mathtt{F}$		
$\mathcal{M}, \mathbf{z} \models \text{“}\sum_i r_i \cdot \mathrm{P}[X^{(i)}(t_i) = k_i] \bowtie c\text{”}$		
	$\Leftrightarrow$	$\mathbf{n}_{exp}(\text{“}\sum_i r_i \cdot \mathrm{P}[X^{(i)}(t_i) = k_i] \bowtie c\text{”}) \cdot \mathbf{z} \bowtie 0$
$\mathcal{M}, \mathbf{z} \models \neg\psi$	$\Leftrightarrow$	$\mathcal{M}, \mathbf{z} \not\models \psi$
$\mathcal{M}, \mathbf{z} \models \psi \wedge \phi$	$\Leftrightarrow$	$\mathcal{M}, \mathbf{z} \models \psi$ and $\mathcal{M}, \mathbf{z} \models \phi$
$\mathcal{M}, \mathbf{z} \models \psi \vee \phi$	$\Leftrightarrow$	$\mathcal{M}, \mathbf{z} \models \psi$ or $\mathcal{M}, \mathbf{z} \models \phi$
$\mathcal{M}, \mathbf{z} \models \mathtt{X}\,\psi$	$\Leftrightarrow$	$\mathcal{M}, \mathbf{M} \cdot \mathbf{z} \models \psi$
$\mathcal{M}, \mathbf{z} \models \psi\,\mathtt{U}\,\phi$	$\Leftrightarrow$	there is $j \geq 0$ such that $\mathcal{M}, \mathbf{M}^j \cdot \mathbf{z} \models \phi$ and $\mathcal{M}, \mathbf{M}^i \cdot \mathbf{z} \models \psi$ for $i = 0, \ldots, j-1$
$\mathcal{M}, \mathbf{z} \models \psi\,\mathtt{R}\,\phi$	$\Leftrightarrow$	for all $i \geq 0$ if $\mathcal{M}, \mathbf{M}^j \cdot \mathbf{z} \not\models \psi$ for $0 \leq j < i$ then $\mathcal{M}, \mathbf{M}^i \cdot \mathbf{z} \models \phi$.

Fig. 1. Ternary satisfaction relation $\models$: whether the model $\mathcal{M}$ with $\mathcal{T}_{\mathcal{M}} = (S, \mathbf{M})$ satisfies the specification at a state $\mathbf{z}$.

Implies ($\rightarrow$) is defined as $\psi \rightarrow \phi \ \Leftrightarrow \ \neg\psi \vee \phi$ and equivalent ($\leftrightarrow$) is defined as $\psi \leftrightarrow \phi \ \Leftrightarrow \ (\psi \rightarrow \phi) \wedge (\phi \rightarrow \psi)$.

The meaning of temporal operators is as follows: $\mathtt{X}\,\psi$ is true if and only if ψ is true at the next step; $\psi\,\mathtt{U}\,\phi$ is true if and only if ϕ eventually becomes true and until ϕ becomes true ψ is true; $\psi\,\mathtt{R}\,\phi$ is true if and only if ϕ is true while ψ is false. The other operators can be defined in terms of these: $\Box\,\psi$ is true if ψ is *always* true, which is equivalent to $\mathtt{F}\,\mathtt{R}\,\psi$; and $\diamond\,\psi$ is true if ψ *eventually* becomes true, which is equivalent to $\mathtt{T}\,\mathtt{U}\,\psi$.

Formally, the semantics of iLTL formulas is defined by a binary satisfaction relation $\models \subset \mathcal{M} \times \psi$. In order to help explain the binary satisfaction relation $\models$, we define an overloaded ternary satisfaction relation $\models \subset \mathcal{M} \times \mathbb{R}^{|S|} \times \psi$ which is described in Fig. 1. For simplicity, we write $\mathcal{M} \models \psi$ for $(\mathcal{M}, \psi) \in \models$ and $\mathcal{M}, \mathbf{z} \models \psi$ for $(\mathcal{M}, \mathbf{z}, \psi) \in \models$.

The ternary satisfaction relation is about a single path $\mathbf{x}$ with $\mathbf{x}(0) = \mathbf{z}$: whether the state transitions from the given initial state satisfy the iLTL formula. Using the definition of the ternary satisfaction relation, the binary satisfaction relation $\models$ is defined as:

$$\mathcal{M} \models \psi \Leftrightarrow \mathcal{M}, \ \mathbf{z} \models \psi \text{ for all initial state } \mathbf{z}.$$

The binary satisfaction relation is about all paths: $\mathcal{M}$ is a model of ψ if and only if all paths $\mathbf{x}$ starting from every initial state $\mathbf{z}$ satisfy ψ.

5 Model Checking

We present two iLTL model checking algorithms: 1) an intuitive algorithm based on *Disjunctive Normal Form* (DNF), and 2) a more efficient algorithm based on a Büchi automaton. We also prove the soundness of iLTL model checking and find the conditions under which iLTL model checking is complete.

Before we explain the details of the model checking algorithms, we examine a simple model checking example here. This simplified example explains the key ideas of the iLTL model checking algorithm.

Example 1. Let M be a DTMC with $\mathcal{T}_M = (\{A, B\}, \mathbf{M})$, where

$$\mathbf{M} = \begin{bmatrix} 0.5 & 0 \\ 0.5 & 1 \end{bmatrix}.$$

M has a sink state B, and on every step the probability $\mathrm{P}[M(t) = A]$ is halved. Suppose that we want to know whether $\mathrm{P}[M(t) = B] > 0.8$ within two steps if $\mathrm{P}[M(0) = A] > 0.9$ initially.

To explain the problem, suppose we have the atomic propositions as follows:

$$a : \mathrm{P}[M = A] > 0.9,\ b : \mathrm{P}[M = B] > 0.8,\ b' : \mathrm{P}[M = B] \le 0.8.$$

Let an iLTL formula ψ be $a \rightarrow (b \vee \mathtt{X}\, b \vee \mathtt{X}\, \mathtt{X}\, b)$. Then, the problem is checking whether $M \models \psi$. Because $M \models \psi$ if and only if $M, \mathbf{z} \models \psi$ for all initial pmfs $\mathbf{z}$, we find an initial pmf $\mathbf{z}$ such that $M, \mathbf{z} \models \neg\psi$ is a counterexample. The negated specification $\neg\psi$ is $a \wedge b' \wedge \mathtt{X}\, b' \wedge \mathtt{X}\, \mathtt{X}\, b'$. We can write it in the following vector form, which is the normal form except for the place of the constant term:

$$\begin{array}{llll} [\,1\,0\,] \cdot \mathbf{z} > 0.9 & \wedge & [\,0\,1\,] \cdot \mathbf{z} \le 0.8 & \wedge \\ [\,0\,1\,] \cdot \mathbf{M} \cdot \mathbf{z} \le 0.8 & \wedge & [\,0\,1\,] \cdot \mathbf{M}^2 \cdot \mathbf{z} \le 0.8, & \end{array}$$

where $\mathbf{z} = [\mathrm{P}[M(0) = A], \mathrm{P}[M(0) = B]]^T$. Any pmf $\mathbf{z}$ that satisfies the four constraints is a counterexample we are seeking. In summary,

$$M \models \psi \Leftrightarrow \left\{ \mathbf{z} \in \mathbb{R}^2 : \begin{array}{ll} [\,1\,0\,] \cdot \mathbf{z} \ge 0 \wedge & [\,0\,1\,] \cdot \mathbf{z} \ge 0 \wedge \\ [\,1\,1\,] \cdot \mathbf{z} = 1 \wedge & [\,1\,0\,] \cdot \mathbf{z} > 0.9 \wedge \\ [\,0\,1\,] \cdot \mathbf{z} \le 0.8 \wedge & [\,0.5\,1\,] \cdot \mathbf{z} \le 0.8 \wedge \\ [\,0.25\,1\,] \cdot \mathbf{z} \le 0.8 & \end{array} \right\} = \emptyset,$$

and any feasible $\mathbf{z}$ is a counterexample. Checking the emptiness of the constraint set can be done by linear programming [28]. In general, iLTL model checking involves a series of feasibility checking.

Observe that regardless of the initial pmf, there is a unique limiting pmf: On every step the probability $\mathrm{P}[M(t) = A]$ is halved and eventually the probability becomes zero. Hence, the unique limiting pmf is $[\,0\,1\,]^T$; eventually, the atomic proposition a becomes false and b becomes true. In fact, the longest duration a is true is a single step when $\mathrm{P}[M(0) = A] = 1$. From the second step, it becomes false. Also, the longest duration b is false is three steps when $\mathrm{P}[M(0) = B] = 0$: $\mathrm{P}[M(0) = B] = 0$, $\mathrm{P}[M(1) = B] = 0.5$, $\mathrm{P}[M(2) = B] = 0.75$, and $\mathrm{P}[M(3) = B] = 0.875$. In this paper, we derive the minimum step N such that all atomic propositions in a given iLTL formula become constants.

As shown in the above example, iLTL model checking process works as follows: Given an iLTL specification and a multiple DTMC model, we compute a search depth N and build a Büchi automaton for the negated specification. Then, we look for a feasible path of length N in the Büchi automaton.

5.1 Computation of Search Depth

We now derive the minimum time step N from which all atomic propositions of a given iLTL formula become constants. If all atomic propositions become constants, then we can immediately determine the truth values of the X, U, and R formulas, which leads to an algorithm in the next section that removes all temporal operators from the iLTL formula.

Theorem 1. *Given an iLTL formula ψ and a multiple DTMC model $\mathcal{M}$ of n individual DTMCs such that $\mathcal{T}_{\mathcal{M}} = (S, \mathbf{M})$, if*

$$\begin{array}{l} - \mathbf{M} \text{ is diagonalizable,} \\ - |\lambda_{n+1}| < 1, \\ - \mathbf{n}_{exp}(ap) \cdot \mathbf{x}(\infty) \neq 0 \text{ for all atomic propositions } ap \text{ of } \psi, \end{array} \tag{4}$$

then there is a time step N from which all atomic propositions of ψ become constants.

Proof. If the first two conditions of (4) are satisfied, then the unique final state $\mathbf{x}(\infty)$ can be computed. For an atomic proposition ap of ψ, let δ be $\mathbf{n}_{exp}(ap) \cdot \mathbf{x}(\infty)$, then

$$\mathbf{n}_{exp}(ap) \cdot \mathbf{x}(t) \bowtie 0 \Leftrightarrow \mathbf{n}_{exp}(ap) \cdot (\mathbf{x}(t) - \mathbf{x}(\infty)) \bowtie -\delta. \tag{5}$$

Note that δ in (5) is not zero, but $\mathbf{n}_{exp}(ap) \cdot (\mathbf{x}(t) - \mathbf{x}(\infty))$ converges to zero as t increases. Thus, depending on the comparator $\bowtie$ and the sign of δ, the truth value of *ap* will eventually be settled on either *true* or *false*.

Now, we find a monotonically decreasing upper bound and a monotonically increasing lower bound of $\mathbf{n}_{exp}(ap) \cdot (\mathbf{x}(t) - \mathbf{x}(\infty))$ that converge toward each other. Thus, when one of the bounds becomes $-\delta$, the value of *ap* does not change any more. Let $\mathbf{P} \cdot \Lambda \cdot \mathbf{P}^{-1}$ be a diagonalization of $\mathbf{M}$, then the upper and the lower bounds can be found as follows:

$$\begin{aligned} &|\mathbf{n}_{exp}(ap) \cdot (\mathbf{x}(t) - \mathbf{x}(\infty))| \\ &\quad = \left| \sum_{i=n+1}^{|S|} \mathbf{r}_i \cdot \lambda_i^t \cdot \left(\sum_{j=1}^{|S|} \mathbf{P}_{ij}^{-1} \cdot \mathbf{x}(0)_j \right) \right| \\ &\quad \le \sum_{i=n+1}^{|S|} |\mathbf{r}_i| \cdot |\lambda_i|^t \cdot \left(\sum_{j=1}^{|S|} \left|\mathbf{P}_{ij}^{-1}\right| \cdot |\mathbf{x}(0)_j| \right) \\ &\quad \le \sum_{i=n+1}^{|S|} |\mathbf{r}_i| \cdot |\lambda_i|^t \cdot \max_{j=1}^{|S|} \left|\mathbf{P}_{ij}^{-1}\right|, \end{aligned} \tag{6}$$

where $\mathbf{r} = \mathbf{n}_{exp}(ap) \cdot \mathbf{P}$. The outer sum of the second line begins from $n + 1$; the first n summation corresponds to $\mathbf{x}(\infty)$ and it cancels the $\mathbf{x}(\infty)$ of the first line. The last inequality holds because $0 \le \mathbf{x}(0)_i \le 1$ for $i = 1, \ldots, |S|$ and $\sum_{i=1}^{|S|} \mathbf{x}(0)_i = 1$. Thus, the time bound N_{ap} for *ap* is

$$N_{ap} = \operatorname*{argmin}_{t \in \mathbf{N}} \sum_{i=n+1}^{|S|} \left(|\mathbf{r}_i| \cdot \max_{j=1}^{|S|} \left|\mathbf{P}_{ij}^{-1}\right| \right) \cdot |\lambda_i|^t < |\delta|.$$

The summation converges to 0 because $|\lambda_i| < 1$ for $i = n + 1, \ldots, |S|$, thus the bound N_{ap} exists. The time bound for all atomic propositions of ψ is

$$N = \max_{ap \text{ of } \psi} N_{ap}.$$

□

The first condition of (4) is used in computing the search depth N. Although not every matrix is diagonalizable, the diagonalizability condition can be easily regained by slightly changing a defective matrix [37]. The second condition is necessary and sufficient condition for the

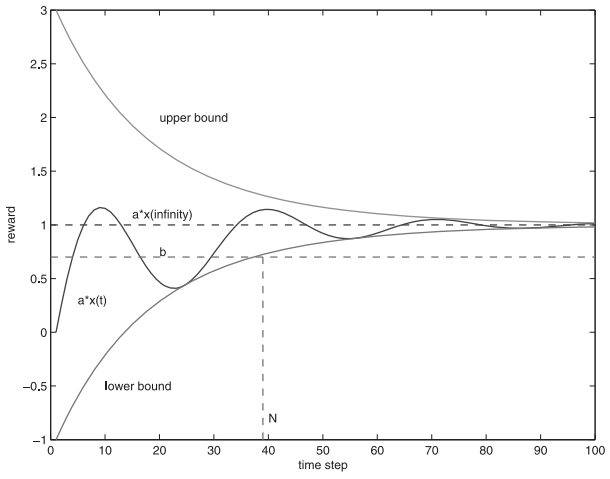


Fig. 2. The upper and the lower bounds for an expected reward. The graph labeled as `a* x(t)` is $\mathbf{a}\cdot\mathbf{x}(t)$ and the line labeled as `a* x(infinity)` is $\mathbf{a}\cdot\mathbf{x}(\infty)$.

existence of a unique limiting state. The last condition prevents the transient oscillating modes, which can keep changing the truth values of atomic propositions forever. In most cases, this condition can be satisfied by adding or subtracting a small reward to the atomic propositions.

Fig. 2 shows an illustration about the bounds of Theorem 1. Let an atomic proposition *ap* be $\mathbf{a}\cdot\mathbf{x}(t) > b$. The oscillating line in the graph shows the expected reward $\mathbf{a}\cdot\mathbf{x}(t)$ over time which converges to 1 ($\mathbf{a}\cdot\mathbf{x}(\infty) = 1$). The monotonically decreasing and increasing lines are the upper and the lower bounds of (6). Note that the truth value of *ap* is changing over time: false at step 0, true at step 10, false at step 20, and true at step 30. However, from step 39 (when the lower bound meets b) onward it is true.

5.2 Model Checking as Feasibility Checking

In this section, we explain how to remove all temporal operators from iLTL formulas based on the search depth explained in the previous section. After removing the temporal operators, the formula can be converted to DNF whose satisfiability can be checked by examining the feasibility of each conjunctive subformula as we have seen in Example 1.

Let N be the time bound found from Theorem 1, then the value of iLTL formula for any time $t \geq N$ can be determined instantly because

$$\begin{aligned} \mathcal{M},\mathbf{x}(t) &\models \mathtt{X}\ \psi \Leftrightarrow \mathcal{M},\mathbf{x}(t+1) \models \psi,\\ \mathcal{M},\mathbf{x}(t) &\models \psi\ \mathtt{U}\ \phi \Leftrightarrow \mathcal{M},\mathbf{x}(t) \models \phi,\\ \mathcal{M},\mathbf{x}(t) &\models \psi\ \mathtt{R}\ \phi \Leftrightarrow \mathcal{M},\mathbf{x}(t) \models \phi. \end{aligned} \tag{7}$$

Using (7) and the normalization function $\mathbf{a}$, we can remove all temporal operators from an iLTL formula ψ. For example, function `f`, in Fig. 3, removes all temporal operators from a given iLTL formula ψ such that ψ and $\mathtt{f}(\psi, 0)$ are equivalent. The following theorem proves their equivalence:

Theorem 2. *Given a multiple DTMC model $\mathcal{M}$ and an iLTL formula ψ, if the conditions of (4) are satisfied, then $\mathcal{M} \models \psi \Leftrightarrow \mathcal{M} \models \mathtt{f}(\psi, 0)$.*

```
f(ψ,t) {
   if(ψ is T)      return T
   if(ψ is F)      return F
   if(ψ is an AP) return a(ψ,t)
   if(ψ is (φ))    return f(φ,t)
   if(ψ is ¬φ)     return ¬(f(φ,t))
   if(ψ is φ ∨ η)  return (f(φ,t)) ∨ (f(η,t))
   if(ψ is φ ∧ η)  return (f(φ,t)) ∧ (f(η,t))
   if(ψ is X φ)
      if(t ≥ N)    return f(φ,t)
      else         return f(φ,t+1)
   if(ψ is φ U η)
      if(t ≥ N)    return f(η,t)
      else         return (f(η,t)) ∨
                          ( (f(φ,t)) ∧ (f(ψ,t+1)) )
   if(ψ is φ R η)
      if(t ≥ N)    return f(η,t)
      else         return ( (f(φ,t)) ∧ (f(η,t)) ) ∨
                          ( (f(η,t)) ∧ (f(ψ,t+1)) )
}
```

Fig. 3. Function `f` removes all temporal operators from an iLTL formula ψ.

Proof. We prove a stronger condition for any $\mathbf{x}$ and t,

$$\mathcal{M},\mathbf{x}(t) \models \psi \Leftrightarrow \mathcal{M},\mathbf{x}(0) \models \mathtt{f}(\psi, t).$$

We prove the equivalence by induction on the structural tree of iLTL formula ψ. The leaf nodes of ψ are `T`, `F`, or atomic propositions (*ap*), which are the induction base.

$$\begin{aligned} \mathcal{M},\mathbf{x}(0) \models \mathtt{f}(\mathtt{T},t) &\Leftrightarrow \mathcal{M},\mathbf{x}(t) \models \mathtt{T},\\ \mathcal{M},\mathbf{x}(0) \models \mathtt{f}(\mathtt{F},t) &\Leftrightarrow \mathcal{M},\mathbf{x}(t) \models \mathtt{F},\\ \mathcal{M},\mathbf{x}(0) \models \mathtt{f}(ap,t) &\Leftrightarrow \mathbf{n}(ap,t)\cdot\mathbf{x}(0) \bowtie 0\\ &\Leftrightarrow \mathbf{n}(ap,0)\cdot\mathbf{x}(t) \bowtie 0\\ &\Leftrightarrow \mathcal{M},\mathbf{x}(t) \models ap. \end{aligned}$$

The induction hypothesis is $\mathcal{M},\mathbf{x}(t) \models \psi \Leftrightarrow \mathcal{M},\mathbf{x}(0) \models \mathtt{f}(\psi, t)$ for any iLTL formula ψ and ϕ of height h or less. Let η be a formula of height $h+1$. Here, we prove only the following four cases out of the seven cases of Fig. 3. The rest can be proved similarly.

- $\eta = \neg\psi$:

$$\begin{aligned} &\mathcal{M},\mathbf{x}(0) \models \mathtt{f}(\neg\psi, t)\\ &\quad\Leftrightarrow \mathcal{M},\mathbf{x}(0) \models \neg\mathtt{f}(\psi,t) && \textit{by the def. of } \mathtt{f}\\ &\quad\Leftrightarrow \mathcal{M},\mathbf{x}(0) \not\models \mathtt{f}(\psi,t) && \textit{by the def. of } \models\\ &\quad\Leftrightarrow \mathcal{M},\mathbf{x}(t) \not\models \psi && \textit{by the induction hypothesis}\\ &\quad\Leftrightarrow \mathcal{M},\mathbf{x}(t) \models \neg\psi && \textit{by the def. of } \models. \end{aligned}$$

- $\eta = \psi \vee \phi$: by the same sequence of reasoning of the case $\eta = \neg\psi$, $\mathcal{M},\mathbf{x}(0) \models \mathtt{f}(\psi \vee \phi, t) \Leftrightarrow \mathcal{M},\mathbf{x}(t) \models \psi \vee \phi$.

- $\eta = \mathtt{X}\ \psi$:
  $-$case $1 : t+1 < N$
  $\mathcal{M}, \mathbf{x}(0) \models \mathtt{f}(\mathtt{X}\ \psi, t)$
  $\Leftrightarrow \mathcal{M}, \mathbf{x}(0) \models \mathtt{f}(\psi, t+1)$ *by the def. of* f
  $\Leftrightarrow \mathcal{M}, \mathbf{x}(t+1) \models \psi$ *by the induction hypothesis*
  $\Leftrightarrow \mathcal{M}, \mathbf{M} \cdot \mathbf{x}(t) \models \psi$
  $\Leftrightarrow \mathcal{M}, \mathbf{x}(t) \models \mathtt{X}\ \psi$ *by the def. of* $\models$.
  $-$case $2 : t+1 \geq N$
  $\mathcal{M}, \mathbf{x}(0) \models \mathtt{f}(\mathtt{X}\ \psi, t)$
  $\Leftrightarrow \mathcal{M}, \mathbf{x}(0) \models \mathtt{f}(\psi, t)$ *by the def. of* f
  $\Leftrightarrow \mathcal{M}, \mathbf{x}(t) \models \psi$ *by the induction hypothesis*
  $\Leftrightarrow \mathcal{M}, \mathbf{x}(t) \models \mathtt{X}\ \psi$ *by the steady state relation*(7).

- $\psi\ \mathtt{U}\ \phi$:
  - Case 1: $t+1 < N$: By the same sequence of reasoning as the first case of $\eta = \mathtt{X}\ \psi$ and by equivalence $\psi\ \mathtt{U}\ \phi \equiv \phi \vee (\psi \wedge \mathtt{X}\ (\psi\ \mathtt{U}\ \phi))$, $\mathcal{M}, \mathbf{x}(0) \models \mathtt{f}(\psi\ \mathtt{U}\ \phi, t) \Leftrightarrow \mathcal{M}, \mathbf{x}(t) \models \psi\ \mathtt{U}\ \phi$.
  - Case 2: $t+1 \geq N$: By the same sequence of reasoning of the second case of $\eta = \mathtt{X}\ \psi$, $\mathcal{M}, \mathbf{x}(0) \models \mathtt{f}(\psi\ \mathtt{U}\ \phi, t) \Leftrightarrow \mathcal{M}, \mathbf{x}(t) \models \psi\ \mathtt{U}\ \phi$. □

When an iLTL formula is converted to DNF, the number of conjunctive subterms can grow exponentially in terms of the search depth N. For example, the conjunctive formulas of $\square\ (a \vee b)$ have $\bigwedge_{t=0}^{N}(a(t) \mid b(t))$, where $\mid$ is a choice operator. Thus, the number of conjunctive terms is 2^{N+1}; for large N, model checking becomes practically impossible. However, notice that there are many common terms in the conjunctive formulas. If infeasibility is found in the common terms, then we can skip checking all the conjunctive formulas with the common terms. For this computational efficiency, we build a Büchi automaton [7]; the conjunctive terms are generated by unrolling the structure of the Büchi automaton.

5.3 Büchi Automaton

A Büchi automaton is a finite state automaton that accepts infinite strings. As regular automata have final states, Büchi automata have *accepting states* that must be visited infinitely often for a string to be accepted. For a finite state automaton to accept an infinite string, there must be a loop in the automaton.

Formally, a Büchi automaton is a quintuple $\mathcal{A} = (\Sigma, Q, \Delta, Q^0, F)$, where Σ is a finite alphabet (2^{AP} in the expand algorithm), Q is a set of states, $\Delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $Q^0 \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of accepting states. Let $\sigma : \mathbb{N} \rightarrow \Sigma$ be an infinite string, then a *run* $\rho : \mathbb{N} \rightarrow Q$ of $\mathcal{A}$ over σ is a mapping such that $\rho(0) \in Q^0$ and $(\rho(i), \sigma(i), \rho(i+1)) \in \Delta$. An infinite string σ is accepted by a Büchi automation $\mathcal{A}$ if there is a run ρ of $\mathcal{A}$ over σ such that $inf(\rho) \cap F \neq \emptyset$, where $inf(\rho)$ is a set of states of $\mathcal{A}$ that appears infinitely often in ρ.

A Büchi automaton $\mathcal{A}_\psi$ for an iLTL formula ψ can be built by the expand algorithm so that $\mathcal{A}_\psi$ accepts those and only those strings accepted by ψ [14]. While building the Büchi automaton, the expand algorithm manipulates three sets of formulas: *New*, *Old*, and *Next* at each node, where *New* has yet-to-be-processed formulas, *Old* has already processed consistent formulas, and *Next* has the formulas that should be satisfied in the next step. After recursively splitting, transforming, and merging the nodes, the expand algorithm returns a graph such that the formulas, except for the *until* formulas, in the *Old* state of a node are satisfiable if there is an infinite path from the node. The satisfiability for the *until* formula can be checked using the acceptance conditions of the Büchi automaton: The accepting states of $\mathcal{A}_\psi$ for a formula $\psi\ \mathtt{U}\ \phi$ are the nodes q such that $\phi \in q.Old$ or $\psi\ \mathtt{U}\ \phi \notin q.Old$. That is, an accepting run should eventually satisfy ϕ or the formula can be satisfied without satisfying $\psi\ \mathtt{U}\ \phi$. For more details about the expand algorithm, please see [14].

5.4 Model Checking with Büchi Automaton

In this section, we explain an efficient iLTL model checking algorithm based on a Büchi automaton. We also prove the soundness of iLTL model checking and find the conditions under which iLTL model checking is complete.

Let T be a tree of height N (the search depth found in Theorem 1), generated by unrolling the graph structure of the Büchi automaton for the given formula, then each path of T from the root node to a leaf node can be regarded as a conjunctive formula of DNF if the acceptance condition for *until* formula is considered additionally. The path to a subtree of T can be considered as a common prefix of all conjunctive terms in the subtree. Because we are looking for a feasible path as a counterexample, if the constraints collected along the prefix are infeasible, then we can skip checking the subtree.

For simplicity, we define the set of atomic propositions that needs to be satisfied at a node q as $q.AP = q.Old \cap AP$. Also, because the time index for a node $\rho(1)$ is 0, to reduce the confusion, we assume that the index to the runs begins from -1. That is, $\rho : \mathbb{N} \cup \{-1\} \rightarrow Q$ such that $\rho(-1) \in Q^0$.

Let a labeling function $L : \mathbb{R}^{|S|} \rightarrow 2^{AP}$ be

$$L(\mathbf{z}) = \{ap \in AP : \mathbf{a}(ap, 0)(\mathbf{z}) \text{ is true}\},$$

where $\mathbf{z} \in \mathbb{R}^{|S|}$ is an array of pmfs. The labeling function L returns a subset of AP that is true at the given state. An infinite string $\sigma_{\mathbf{x}} : \mathbb{N} \rightarrow 2^{AP}$ generated by $\mathcal{M}$ and $\mathbf{x}$ is defined as

$$\sigma_{\mathbf{x}}(t) = L(\mathbf{x}(t)),$$

and the language $\mathcal{L}_{\mathcal{M}}$ generated by $\mathcal{M}$ is defined as

$$\mathcal{L}_{\mathcal{M}} = \bigcup \sigma_{\mathbf{x}} \text{ for all probability vector functions } \mathbf{x}.$$

Let $\mathcal{A}_\psi$ be the Büchi automaton of an iLTL formula ψ built from the expand algorithm [14] and let $\mathcal{L}_{\mathcal{A}_\psi}$ be the set of infinite strings accepted by $\mathcal{A}_\psi$. Then, $\mathcal{M}$ is a model of ψ if its behaviors are within the behaviors corresponding to the specification.

$$\mathcal{M} \models \psi \Leftrightarrow \mathcal{L}_{\mathcal{M}} \subseteq \mathcal{L}_{\mathcal{A}_\psi}.$$

In practice, we check the emptiness of the intersection between $\mathcal{L}_{\mathcal{M}}$ and $\mathcal{L}_{\mathcal{A}_{\neg\psi}}$:

$$\mathcal{M} \models \psi \Leftrightarrow \mathcal{L}_{\mathcal{M}} \cap \mathcal{L}_{\mathcal{A}_{\neg\psi}} = \emptyset.$$

If the intersection is not empty, then any string $\sigma_{\mathbf{x}} \in \mathcal{L}_{\mathcal{M}} \cap \mathcal{L}_{\mathcal{A}_{\neg\psi}}$ is a counterexample that proves $\mathcal{M}, \mathbf{x}(0) \not\models \psi$.

The models of the classical LTL are Kripke structures, which have a finite number of states. In the model checking, an intersection automaton between the Kripke structure and the Büchi automaton for the specification is built and its emptiness is checked. However, in iLTL model checking, with its uncountably large pmf space, the intersection automaton cannot be constructed. Instead, we collect the normal form atomic propositions from the runs of the automaton, and check whether this set of constraints is feasible. The transitions from any feasible vector satisfy the atomic propositions in the run. That is, let ρ be an accepting run of the Büchi automaton, then we check whether there is a probability vector function $\mathbf{x}$ that satisfies the constraint

$$\bigwedge_{t=0}^{\infty} \mathbf{a}_{set}(\ \rho(t).AP,\ 0)(\mathbf{x}(t)).$$

Because $\mathbf{a}_{set}(\rho(t).AP, 0)(\mathbf{x}(t)) \Leftrightarrow \mathbf{a}_{set}(\rho(t).AP, t)(\mathbf{x}(0))$, instead of finding the function $\mathbf{x}$ itself, we look for its initial state $\mathbf{x}(0)$. Also, because $L(\mathbf{x}(t)) = L(\mathbf{x}(N))$ for $t \geq N$, we can reduce the infinite number of conjunctions to a finite number as the following theorem shows:

Theorem 3. *If the conditions in (4) are satisfied, then $\mathcal{A}_\psi$ accepts a string $\sigma_{\mathbf{x}}$ if and only if there is a run ρ such that*

$$\begin{aligned} &-\rho(m) = \rho(n),\ \text{for } n > m \geq N, \\ &-\rho(k) \in F \text{ for some } k \in [m, n], \\ &-\bigwedge_{t=0}^{n-1} \mathbf{a}_{set}(\rho(t).AP, t) \text{ is feasible.} \end{aligned} \tag{8}$$

Proof. (only if) Let $\sigma_{\mathbf{x}}$ be a string accepted by $\mathcal{A}_\psi$, then there is a run ρ over $\sigma_{\mathbf{x}}$ such that $(\rho(t-1), L(\mathbf{x}(t)), \rho(t)) \in \Delta$ for $t \geq 0$. Thus, $\bigwedge_{t=0}^{\infty} \mathbf{a}_{set}(\rho(t).AP, t)(\mathbf{x}(0))$ is true. Because ρ is an infinite run and $\mathcal{A}_\psi$ has finite number of states, there is a loop in ρ such that $\rho(m) = \rho(n)$ and $n > m \geq N$. Also, because $inf(\rho) \cap F \neq \emptyset$, $\rho(k) \in inf(\rho) \cap F$ for some $k \in [m, n]$.

(if) Let $\mathbf{x}(0)$ be a feasible vector for the constraint set $\bigwedge_{t=0}^{n-1} \mathbf{a}_{set}(\rho(t).AP, t)$. Then, because $\sigma_{\mathbf{x}}(t) = \sigma_{\mathbf{x}}(N)$ for $t \geq N$, the following run ρ' is accepted by $\mathcal{A}_\psi$.

$$\rho'(t) = \begin{cases} \rho(t), & \text{if } t < m, \\ \rho(t'), & \text{otherwise,} \end{cases}$$
$$\text{where } t' = m + (t - m) \bmod (n - m + 1).$$

That is, ρ' is a string $\rho(0), \ldots, \rho(m-1)(\rho(m), \ldots, \rho(n-1))^\omega$. Thus, $\sigma_{\mathbf{x}}$ is accepted by $\mathcal{A}_\psi$. □

Theorem 4. *iLTL Model checking is sound.*

Proof. If the conditions in (4) are not satisfied, then the iLTL checker aborts model checking. If the conditions in (4) are met, the iLTL checker returns true if and only if there is no prefix $\rho(0), \ldots, \rho(n-1)$ of a run ρ for $\mathcal{A}_{\neg\psi}$ that satisfies the conditions of (8). Hence, because of Theorem 3, iLTL model checking is sound. □

Theorem 5. *If the conditions in (4) are satisfied, iLTL model checking is complete.*

Proof. If the conditions in (4) are satisfied, then we can compute the search depth N. Let R and R' be subsets of the state of $\mathcal{A}_{\neg\psi}$ such that $R \subseteq R'$, then if $\mathbf{x}$ is a feasible point of $\bigwedge_{q \in R'} \mathbf{a}_{set}(q.AP, N)$, then $\mathbf{x}$ is also a feasible point of $\bigwedge_{q \in R} \mathbf{a}_{set}(q.AP, N)$. Thus, in order to find a feasible run after index N, we only need to check the shortest run to each loop. Since there are a finite number of loops in $\mathcal{A}_{\neg\psi}$, a finite number of paths of length N in $\mathcal{A}_{\neg\psi}$, and a finite number of shortest paths from $\rho(N)$ to each loop, we can check whether there is a prefix of a run of $\mathcal{A}_{\neg\psi}$ that satisfies the conditions in (8). □

As described in Theorem 3, iLTL model checking algorithm looks for a feasible loop after the search depth N. This search can be duplicated for many runs. Using the fact that the atomic propositions become constants after step N, these redundant searches can be eliminated: We build a minimal set $F^+ \subseteq Q$ that has 1) the nodes of the loops of $\mathcal{A}_\psi$ that have at least a node in F and that could be cycled by $L(\mathbf{x}(\infty))$, and 2) the nodes reachable to the loops by $L(\mathbf{x}(\infty))$. That is, let $\rho(l) \cdots \rho(m) \cdots \rho(n)$ be a fragment of a run such that $\rho(m) = \rho(n)$, $\rho(k) \in F$ for a $k \in [m, n)$, and $\bigwedge_{t=l}^{n-1} \mathbf{a}_{set}(\rho(t).AP, \infty)$ is feasible. Then, F^+ is a minimal set of nodes that has the nodes $\rho(l) \cdots \rho(n)$ of all such fragments of runs. With F^+, a string $\sigma_{\mathbf{x}}$ is accepted by $\mathcal{A}_\psi$ if and only if $\bigwedge_{t=0}^{N} \mathbf{a}_{set}(\rho(t).AP, t)$ is feasible and $\rho(N) \in F^+$.

The time complexity of iLTL model checking is exponential in terms of the search depth N as it examines all paths of length N of $\mathcal{A}_{\neg\psi}$. However, because we check the feasibility of common prefixes of the paths together and skip checking the subtree if the prefix is infeasible, model checking usually ends much faster; the examples in this paper are checked in less than a minute on a Pentium III machine. Also, because the iLTL checker does not build the intersection automation of the classical LTL, its automaton is usually small.

6 Example

We provide two case studies to illustrate the usefulness of iLTL model checking (the iLTLChecker model checker is available at [1]). The first case study is a *software reliability* assessment. This example shows how iLTL model checking can be used to provide guidance in software design. The second example is about *drug administration safety*. This example shows how counterexamples can be used in computing desired parameters.

6.1 Case Study: Software Reliability

The *reliability* of a software system is the probability that the system can successfully complete its tasks. Because reliability is a key software quality metric, many companies use it as their software quality gate. With the popularity of the component-oriented software designs, the reliability of a system is often computed in terms of its component reliabilities. A system maybe modeled as a Markov chain, where its concrete states are those of the software components and its transition probabilities are the relative frequencies of the control transfers between components. Such frequencies can be obtained by the operation profiling.

In [33], Markov chains are extended with a *failure* state to assess the system reliability; the transition probabilities are scaled with the reliability of the source component and directed edges are added from each component to the failure state labeled with $1 - reliability$ of the component. We extend this reliability model with ideas from *scenario-based software reliability model* [39]: We condition the transition probabilities based on user scenarios. To do so, we prepare a DTMC for each scenario and examine the system reliability in terms of the fractions of users in each scenario.

We illustrate the application of our approach by using the example of an online banking system (Fig. 4). The system comprises of four components: *Session*, *Transaction*, *Logging*, and *DataBase*. We define two DTMCs based on different scenarios: 1) a *User scenario* for user information management scenarios such as changing a PIN number, and 2) a *Bank scenario* for bank account management such as transferring balances. Both scenarios begin by creating a session in the Session component.

In order to assess the reliability of the system, we extend the system with three states, two of them representing (successfully) *Completed* states, one for each scenario, and a *Finished* state. When the system completes an operation, the control moves to the Finished state through one of the Completed states. But, if there is a failure in a component, control moves directly to the Finished state from the component. Thus, the total accumulated probability at the Completed states is the reliability of the system. Thus, suppose $rS = 0.997$, $rT = 0.992$, $rL = 0.998$, and $rD = 0.992$ are the component reliabilities for the Session, the Transaction, the Logging, and the DataBase components, respectively, then the DTMC B for the online banking system is

$$\mathcal{T}_B = (S, \mathbf{M}), \text{where}$$
$$S = \{Su, Tu, Lu, Du, Cu\} \cup \{Sb, Tb, Lb, Db, Cb\} \cup \{Fn\},$$
$$\mathbf{M} = \begin{bmatrix} \mathbf{M}_u & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_b & \mathbf{0} \\ \mathbf{R} & \mathbf{R} & 1 \end{bmatrix},$$
$$\mathbf{M}_u = \begin{bmatrix} rS \cdot 0.6 & rT \cdot 0.3 & rL \cdot 0.2 & rD \cdot 0.3 & 0 \\ rS \cdot 0.02 & rT \cdot 0.5 & rL \cdot 0.1 & rD \cdot 0.1 & 0 \\ rS \cdot 0.05 & rT \cdot 0.1 & rL \cdot 0.5 & rD \cdot 0.1 & 0 \\ rS \cdot 0.03 & rT \cdot 0.1 & rL \cdot 0.2 & rD \cdot 0.5 & 0 \\ rS \cdot 0.3 & 0 & 0 & 0 & 0 \end{bmatrix},$$
$$\mathbf{M}_b = \begin{bmatrix} rS \cdot 0.3 & rT \cdot 0.2 & rL \cdot 0.2 & rD \cdot 0.1 & 0 \\ rS \cdot 0.1 & rT \cdot 0.5 & rL \cdot 0.1 & rD \cdot 0.1 & 0 \\ rS \cdot 0.1 & rT \cdot 0.1 & rL \cdot 0.3 & rD \cdot 0.3 & 0 \\ rS \cdot 0.2 & rT \cdot 0.2 & rL \cdot 0.4 & rD \cdot 0.5 & 0 \\ rS \cdot 0.3 & 0 & 0 & 0 & 0 \end{bmatrix},$$
$$\mathbf{R} = [\,1 - rS \quad 1 - rT \quad 1 - rL \quad 1 - rD \quad 1\,].$$

Note that we have named the states by the first letters of the components suffixed with either a u for the user scenario or a b for the bank scenario.

As our first example, we check whether the reliability of the system is larger than 95 percent regardless of the scenarios. Observe that the reliability of the system depends on the portion of users in each scenario. This problem can be solved by checking the following iLTL formula:

$$\psi_1 : a \rightarrow b, \text{where}$$
$$a : \mathrm{P}[B = Su] + \mathrm{P}[B = Sb] = 1,$$
$$b : \mathrm{Q}[B = Cu] + \mathrm{Q}[B = Cb] > 0.95.$$

The initial condition that all processes begin from the Session state is enforced in the implication ψ_1 by the premise a. In b, the `Q` operator sums the accumulated probability at the Completed states, thus measuring the reliability of the system. The reliability requirement can be established by checking whether $B \models \psi_1$. Results of executing the model checker show that the system does not meet its reliability specification: If more than 94.1 percent of the users are in the bank scenario, then the reliability is less than 95 percent.

As a second example, suppose that the DataBase component is not designed to handle more than 10 processes accessing it concurrently, and suppose that the system does an admission control so that up to 100 users can use the system at the same time. Then, it would be reasonable to constrain the system such that the probability of a process being in the DataBase state never exceeds 0.1. Under these assumptions, we can again check whether we achieve 95 percent reliability. The following iLTL formula describes this requirement:

$$\psi_2 : (a \wedge \Box\, c) \rightarrow b, \text{where}$$
$$c : \mathrm{P}[B = Du] + \mathrm{P}[B = Db] < 0.1.$$

$\Box\, c$ specifies that the probability of a process being in the DataBase state is always less than 0.1. If this condition is satisfied for each process and if there are at most 100 users in the system, then the expected number of processes at the DataBase component is always fewer than 10. Thus, ψ_2 is a specification about the reliability with this constraint. The model checker shows that $B \models \psi_2$ is true, i.e., the system meets the 95 percent reliability threshold.

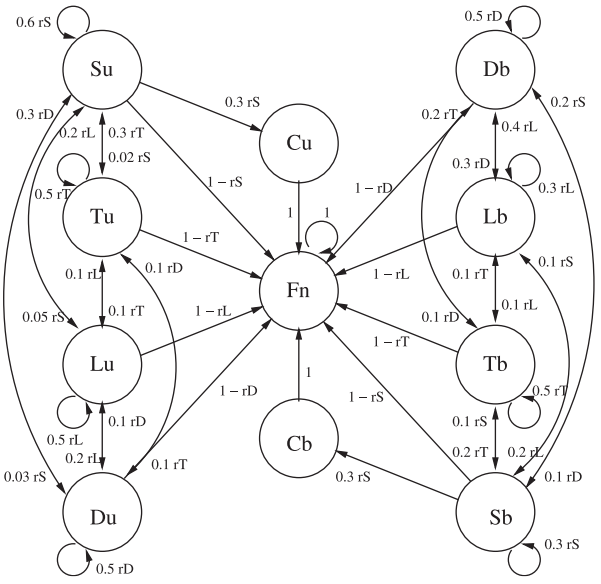


Fig. 4. A state transition diagram for the software reliability example.

6.2 Case Study: Drug Administration

When we take a pill, it is absorbed from the stomach and the gut and is delivered to the liver through the portal vein. From the liver, the absorbed drug is delivered to different parts of our body in a bound form with the plasma. The drug then has its effect on the target tissue and it is then eliminated from our body through the liver or the kidney. In Pharmacokinetics, drug *Absorption, Distribution, Metabolism,* and *Elimination* (ADME) processes are often explained by the compartment model [34], [35]: Organs with similar drug absorption characteristics are modeled as a compartment and the drug molecules make transitions between these compartments. In the compartment model, the amount of drug leaving a compartment is proportional to the drug concentration of the compartment, thus satisfying the Markovian property.

We represent the compartment model as a DTMC such that each compartment is a state, and the transition probabilities between the states are the probability that a drug molecule is moving from one compartment to the other during a sampling time. Drug kinetics is described by the amount of drug or the number of drug molecules in certain compartments. Due to the large number of molecules, it might be impossible to model the state of the drug as a Cartesian product of all molecule locations. However, if we model the state of the drug as a pmf that a randomly picked molecule is in a certain compartment, then the large number of molecules will make the pmf an accurate representation of the actual fractions of the drug in certain compartments.

A drug is active if more than a certain level of the drug is concentrated at the target tissue, and the drug takes effect if the active state is maintained for a certain duration. However, because the drug concentration is decreasing exponentially as it is metabolized, it is not desirable to increase the dose in order to increase the active duration of the drug, especially for drugs with fast clearance rates. Also, certain drugs are toxic to some organs, so drug dosage should be limited. A well-known way to increase a drug's active duration is to coat the drug so that its absorption rate is reduced. However, this will also delay the first drug effect. Thus, mixed use of coated and uncoated drugs is a common practice. In our approach, we model the drug absorption process using a multiple DTMC model in order to find a dose by iLTL model checking. Specifically, we model the behaviors of the coated and the uncoated drugs in two different Markov chains and examine their combined effects using a multiple DTMC model.

Fig. 5 shows a state transition diagram for the drug absorption process. The concrete states in the model are the compartments and the status of the drug: *Sto, Gut, Liv, Kid,* and *Pla* are the representative compartments of the stomach, gut, liver, kidney, and plasma; *Dru, Clr,* and *Rem* are the drug status: intake, cleared, and remaining.

We introduce the state *Rem* to describe the specification in physical units, such as milligram, and find the dose in physical units from the model checking results. Note that as far as the prediction equation is concerned, a DTMC is a linear system. That is, if we scale the initial state, ignoring the fact that it is an array of pmfs, then the trailing states also scale by the same amount. In order to scale the pmfs we introduce the noninteracting state *Rem*: If we scale a pmf by u, then we put the rest of the probability $1-u$ in the *Rem* state. For example, let $\mathcal{M}$ be a DTMC with $\mathcal{T}_{\mathcal{M}} = (S, \mathbf{M})$, and let $\mathcal{M}'$ be a DTMC with $\mathcal{T}_{\mathcal{M}'} = (S \cup \{R\}, \mathbf{M}')$, where R is a fresh sink state. Now, suppose that $\mathbf{x}$ is a counterexample of $\mathcal{M} \models \Box\, \mathrm{P}[D = Kid] < 0.1$, then $[u \cdot \mathbf{x}^T, 1-u]^T$ is a counterexample of $\mathcal{M}' \models \Box\, \mathrm{P}[D = Kid] < 0.1 \cdot u$ for any $0 < u < 1$. Thus, we can use the physical units in model checking. In Fig. 5, *Rem* is introduced as the noninteracting sink node, and then it is drained to the other sink state *Clr* in order to ensure that the model has a unique final state.

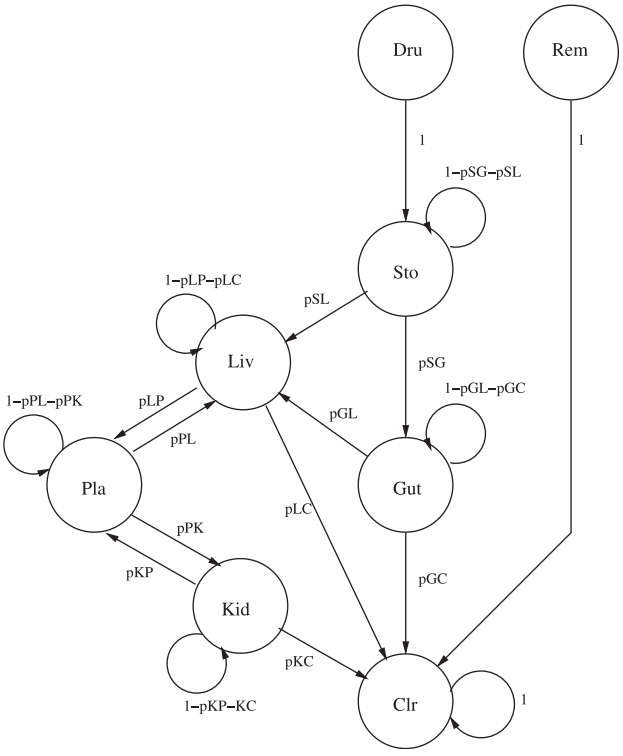


Fig. 5. A state transition diagram for the drug administration model.

We build the Markov chain model assuming that the sampling period is 30 (min). The transition probabilities during a sampling period are as follows: 30 percent of the drug in the stomach moves to the gut (pSG), 1 percent of the drug in the gut clears without being absorbed (pGC), 30 percent of the drug is absorbed from the stomach (pSL), and the same portion of the drug is absorbed from the gut (pGL). To distinguish the absorption rates of the coated drug and the uncoated drug, we suffixed them with c for the coated drug and u for the uncoated drug. The absorption rates for the coated drug are dropped by 90 percent in the stomach ($\mathrm{pSLc} = 0.1 \cdot \mathrm{pSLu}$) and 60 percent in the gut ($\mathrm{pGLc} = 0.4 \cdot \mathrm{pSLu}$). From the liver, 80 percent of the drug is cleared (pLC) during the sampling period and 10 percent is returned to the plasma (pLP), and from the kidney 80 percent is cleared (pKC) and 10 percent is returned to the plasma (pKP). Also, from the plasma, 40 percent is moved to the liver (pPL) and 40 percent is moved to the kidney (pPK). *Clr* is the only absorbing state in the DTMC model that does not have any outgoing links.

```
var:              # Transition probabilities
  pSG  = 0.3,     # stomach to gut
  pGC  = 0.01,    # unabsorbed
  pSLu = 0.3,     # stomach to liver
  pGLu = 0.3,     # gut to liver
  pSLc = pSLu*0.1, # stomach to liver when coated
  pGLc = pGLu*0.4, # gut to liver when coated
  pLP = 0.1,      # liver to plasma
  pLC = 0.8,      # cleared from liver
  pKP = 0.1,      # return from kidney
  pKC = 0.8,      # cleared from kidney
  pPL = 0.4,      # plasma to liver
  pPK = 0.4       # plasma to kidney

model:
  Markov chain D   # DTMC for uncoated drug
    has states :
      { Dru, Rem, Sto, Gut, Liv, Kid, Pla, Clr},
    transits by :
      [ 0, 0, 0,      0,      0,      0,     0,        0;
        0, 0, 0,      0,      0,      0,     0,        0;
        1, 0, 1-pSG-pSLu, 0,  0,      0,     0,        0;
        0, 0, pSG,    1-pGLu-pGC, 0,  0,     0,        0;
        0, 0, pSLu,   pGLu,   1-pLP-pLC, 0,  pPL,      0;
        0, 0, 0,      0,      0,      1-pKP-pKC, pPK,  0;
        0, 0, 0,      0,      pLP,    pKP,   1-pPL-pPK,0;
        0, 1, 0,      pGC,    pLC,    pKC,   0,        1
      ],
  Markov chain C   # DTMC for coated drug
    has states :
      { Dru, Rem, Sto, Gut, Liv, Kid, Pla, Clr},
    transits by :
      [ 0, 0, 0,      0,      0,      0,     0,        0;
        0, 0, 0,      0,      0,      0,     0,        0;
        1, 0, 1-pSG-pSLc, 0,  0,      0,     0,        0;
        0, 0, pSG,    1-pGLc-pGC, 0,  0,     0,        0;
        0, 0, pSLc,   pGLc,   1-pLP-pLC, 0,  pPL,      0;
        0, 0, 0,      0,      0,      1-pKP-pKC, pPK,  0;
        0, 0, 0,      0,      pLP,    pKP,   1-pPL-pPK,0;
        0, 1, 0,      pGC,    pLC,    pKC,   0,        1
      ]

specification:     # iLTL Specification
  id : P[D=Dru] + P[D=Rem] = 1,
  ic : P[C=Dru] + P[C=Rem] = 1,
  e  : P[D=Pla] + P[C=Pla] > 0.008,
  k  : P[D=Kid] + P[C=Kid] < 0.005

  ~ (  (id /\ ic)
    /\ (e \/ X e \/ X X e \/ X X X e)
    /\ <> (  e /\ X e /\ X X e /\ X X X e /\
             X X X X e /\ X X X X X e      )
    /\ [] k
    )
```

Fig. 6. iLTL description for the drug absorption example.

We assume that the drug is active if more than 8 (mg) of the drug is concentrated in the plasma; we also require that its concentration level in the kidney never exceeds 5 (mg). In order for the drug to be effective, we require that its active concentration level should be maintained for 3 hours (6 sampling periods), and we want to have the first drug effect no later than 2 hours (4 periods).

Fig. 6 shows the *iLTLChecker* description for the drug absorption model and specification [1]. The description begins with an optional variable description block that begins with the *var:* tag. Note that in an iLTLChecker description, any string from # to the end of the line is treated as comments and ignored. Next, a model description block is defined: It begins with *model:* tag, where a list of DTMCs are defined separated by commas. Each DTMC description has the same name as that of the DTMC, its set of states, and its probability transition matrix. Finally, an iLTL specification block follows the model description block. This block begins with a *specification:* tag. In this block, an optional list of atomic propositions is defined first and then an iLTL specification is described. In an iLTLChecker description, the logical connectives are `~, /\ , \/, ->, <->` for $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$, and the temporal connectives are `X, U, R, [], <>` for `X` , `U` , `R` , $\Box$, $\diamond$.

In Fig. 6, $\mathrm{P}[D = Dru] + \mathrm{P}[D = Rem] = 1$ means that all of the uncoated drug is either in the *Dru* state or in the *Rem* state. Similarly, $\mathrm{P}[C = Dru] + \mathrm{P}[C = Rem] = 1$ means that all of the coated drug is either in the *Dru* state or in the *Rem* state. Because the drug in the *Rem* state is cleared directly and does not interact with other states, the probabilities $\mathrm{P}[C = Dru]$ and $\mathrm{P}[D = Dru]$ are the doses for the coated and uncoated drug.

Note that, because subformula $id \wedge ic$ does not have any temporal operators, it describes an initial condition. The next subformula $e \vee \mathtt{X}\, e \vee \mathtt{X}\, \mathtt{X}\, e \vee \mathtt{X}\, \mathtt{X}\, \mathtt{X}\, e$ states that the drug concentration at the plasma should be larger than 8 (mg) within three steps. This subformula is about the fast drug effect. The next subformula $\diamond\, (e \wedge \mathtt{X}\, e \wedge \mathtt{X}\, \mathtt{X}\, e \wedge \mathtt{X}\, \mathtt{X}\, \mathtt{X}\, e \wedge \mathtt{X}\, \mathtt{X}\, \mathtt{X}\, \mathtt{X}\, e \wedge \mathtt{X}\, \mathtt{X}\, \mathtt{X}\, \mathtt{X}\, \mathtt{X}\, e)$ specifies a constraint on the duration of the active concentration level. The *eventually* operator ($\diamond$) ensures that this condition will occur. Finally, the last subformula $\Box\, k$ is about the drug concentration level at the kidney: It should never exceed 5 (mg).

The specification to our model checker is the negation of the desired goal. Thus, if the DTMC is not a model of the negated specification, then the counterexample is the prescription that would satisfy the original goal. The model checking result is

```
Depth: 15
Result: F
counterexample:
 pmf(D(0)): [ 0.213311 0.786689 0 0 0 0 0 0 ]
 pmf(C(0)): [ 0.533555 0.466445 0 0 0 0 0 0 ]
```

In the result above, *Depth: 15* means that the search depth N is 15. Note that the search depth is printed before the actual search begins. Thus, if this number is too large, we can modify the conditions to make the problem more tractable. The next line *Result: F* means that the DTMC is not a model of the specification. Because we are checking the negation of the desired formula, the counterexample is the dose that we need. The last two lines are the counterexample: the initial pmfs for the eight states. That is, if we take 0.213 (g) of the uncoated drug and 0.534 (g) of the coated drug, then within 2 hours more than 8 (mg) of the drug is concentrated at the plasma and the active level lasts for more than 3 hours. Also, the drug concentration in the kidney never exceeds 5 (mg).

Fig. 7 shows the drug concentration levels over time in the kidney and in the plasma, when the amount of drug specified in the counterexample is taken. The dashed lines on both graphs are the concentrations due to the uncoated drug, the dot-dashed lines are the concentrations due to the coated drug, and the solid lines are the total concentration by both types of the drug. Note that the uncoated drug causes a fast rise and a fast decline in the drug concentration level, whereas the coated drug has a slow rise and a

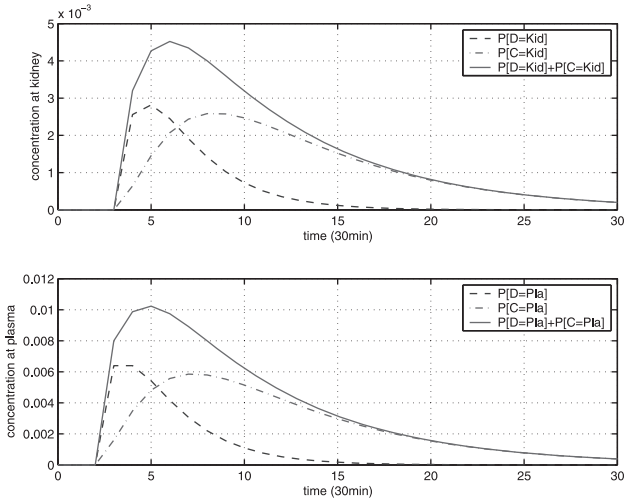


Fig. 7. Drug concentration at kidney and plasma.

slow decline. Thus, by mixing the two forms of the drug, we can get the desired result: a fast and long-lasting effect. The solid lines show how the specification is satisfied: The concentration level in the kidney never exceeds 0.005 and the desired active concentration level (more than 0.008) is maintained in the plasma for more than 6 hours.

Note that the desired effect could not be achieved if we did not mix the coated drug and the uncoated drug. For example, if we modify the initial conditions *id* or *ic* as id: $\mathrm{P}[D = Rem] = 1$ and ic: $\mathrm{P}[C = Rem] = 1$, then the model checking result becomes true, proving that the desired effect cannot be reached. That is, there is no way to achieve the goal with a single type of drug.

We now consider a slightly more difficult problem. In oral drug administration, we cannot assume a constant intake of the drug. Usually, the drug is taken at certain periods or intervals. Thus, the drug concentration level fluctuates following the drug administration period. Hence, it is important to maintain a concentration level near the desired level. Note that if we take more drug than the amount of drug that has cleared, the concentration level will increase gradually and may reach a dangerously toxic level. If we take less drug than the cleared amount, then the concentration level will decrease and the drug will not be effective.

We solve the oral drug administration problem as follows: We check whether there is a repeatable state with certain periodicity and find the amount of drug to take in order to maintain the repetition. Suppose that the drug administration period is 6 hours (12 samples). Also, because we do not need the fast effect during the steady period, we only use the coated drug. The repetition condition can be expressed as

$$\begin{aligned}
&\mathrm{P}[C = Sto] = \mathrm{P}[C(12) = Sto] \wedge \\
&\mathrm{P}[C = Gut] = \mathrm{P}[C(12) = Gut] \wedge \\
&\mathrm{P}[C = Liv] = \mathrm{P}[C(12) = Liv] \wedge \\
&\mathrm{P}[C = Kid] = \mathrm{P}[C(12) = Kid] \wedge \\
&\mathrm{P}[C = Pla] = \mathrm{P}[C(12) = Pla].
\end{aligned}$$

Note that $\mathrm{P}[C = Dru]$ and $\mathrm{P}[C = Rem]$ can be regarded as an input which we can freely control, and $\mathrm{P}[C = Clr]$ is the state for the cleared drug that we do not need to consider.

```
ssa: P[C=Sto]<P[C(12)=Sto]+eps,   ssb: P[C=Sto]>P[C(12)=Sto]-eps,
sga: P[C=Gut]<P[C(12)=Gut]+eps,   sgb: P[C=Gut]>P[C(12)=Gut]-eps,
sla: P[C=Liv]<P[C(12)=Liv]+eps,   slb: P[C=Liv]>P[C(12)=Liv]-eps,
ska: P[C=Kid]<P[C(12)=Kid]+eps,   skb: P[C=Kid]>P[C(12)=Kid]-eps,
spa: P[C=Pla]<P[C(12)=Pla]+eps,   spb: P[C=Pla]>P[C(12)=Pla]-eps,
id : P[D=Rem]=1

~ (  (  ssa /\ ssb /\ sga /\ sgb /\ sla /\ slb /\
        ska /\ skb /\ spa /\ spb )
  /\ <> (  e /\ X e /\ X X e /\ X X X e /\
           X X X X e /\ X X X X X e )
  /\ [] k
  /\ id
  )
```

Fig. 8. iLTL description for the steady state drug administration.

To express our constraints, we use the time offset operator: $\mathrm{P}[C(12) = Sto]$ is the 12-step future probability of $\mathrm{P}[C = Sto]$. One problem in using these constraints is that the equality holds at the limiting state, which violates the third condition of (4):

$$\lim_{t\to\infty} \mathrm{P}[C(t) = Sto] = \lim_{t\to\infty} \mathrm{P}[C(t+12) = Sto].$$

Thus, we use an approximation in Fig. 8, with *eps = 0.00001*. With this specification, we got the following result:

```
Depth: 86
Result: F
counterexample:
  pmf(D(0)): [ 0.000 1 0.000 0.000 0.000 0.000
  0.000 0 ]
  pmf(C(0)): [ 0.680 0 0.008 0.259 0.043 0.003
  0.007 0 ]
```

Note that with the use of small *eps*, the search depth is increased to 86. However, even with the increased depth, it takes less than a second to get the result on a Pentium III machine. The fast result is mainly due to the early pruning of infeasible subtrees in the model checking process. Also, in general, the model checker will finish more quickly if the result is false because it can avoid searching the rest of the search space once a counterexample is found.

The results show that $\mathrm{P}[C(0) = Dru] = 0.680$ is the amount of drug to take to maintain the repeating state. The probabilities $\mathrm{P}[C(0) = Sto] = 0.008$, $\mathrm{P}[C(0) = Gut] = 0.259$, $\mathrm{P}[C(0) = Liv] = 0.043$, $\mathrm{P}[C(0) = Kid] = 0.003$, and $\mathrm{P}[C(0) = Pla] = 0.007$ are the amount of drug in the stomach, the gut, the liver, the kidney, and the plasma when the repeating cycle begins. Also, we do not use the uncoated drug because $\mathrm{P}[D(0) = Rem] = 1$.

Fig. 9 shows the drug concentration level in the kidney and the plasma when 0.68 (g) of the coated drug is taken at 6 hour intervals. Note that the 3 hour concentration level constraint in the plasma is satisfied for each drug administration and the concentration level at the kidney never exceeds its toxic limit.

7 Conclusion

In this paper, we presented a new probabilistic temporal logic called iLTL and its model checking algorithm. Because iLTL specifies properties involving pmf transitions from all possible initial pmfs of DTMCs, it provides an effective and complete method to check the global properties of large-scale

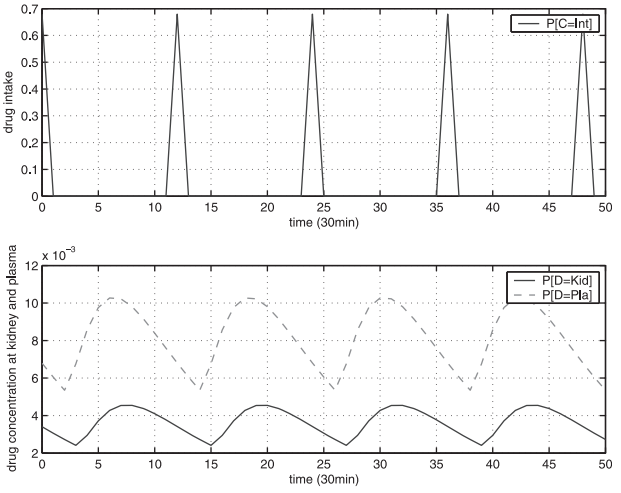


Fig. 9. Drug concentration level in the kidney and the plasma, when the drug is taken at a 6 hour interval.

systems. We extended the logic so that it can specify parallel transitions of multiple DTMCs. Using this extension, we can easily compare the performance of different systems or check their combined effects. As one of our case studies illustrates, by using model checking to find counterexamples of a negated goal, iLTL model checking can find desirable parameters.

We proved the *soundness* of iLTL model checking and showed the easy to satisfy conditions under which iLTL model checking is *complete*. If these conditions are satisfied, then there is a bound after which all atomic propositions of iLTL become constants. This enables the use of bounded model checking [6]. For a large bound, it might be interesting to compare the performance between a Büchi automaton-based checking and *Binary Decision Diagram* (BDD)-based symbolic model checking [30].

Performance can be a concern in some iLTL model checking if a large number of initial states must be checked or they must be checked to a great depth. One of the time-consuming aspects of iLTL model checking is the search for a feasible vector while traversing the graph of a Büchi automaton. Fortunately, feasibility checking in a subtree of the graph can be done independently of other subtrees, which allows the possibility of implementing distributed iLTL model checking to improve performance.

We believe that iLTL is a useful logic to specify and verify aggregate global properties of large-scale systems. The ability of our iLTL model checker to find desirable initial condition can make it a useful tool in the design process for concurrent systems.

ACKNOWLEDGMENTS

The authors thank Eunhee Kim and Kent Cheung for their helpful comments. The authors also thank the anonymous referees for numerous comments that have helped improve the exposition of this paper. This research has been supported in part by CNS 05-09321 and by the US Office of Naval Research under US Department of Defense MURI award N0014-020100715.

REFERENCES

[1] iLTLChecker: http://osl.cs.uiuc.edu/~ykwon4/cgi/iLTL.html, 2010.

[2] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Verifying Continuous Time Markov Chains," *Proc. Eighth Int'l Conf. Computer Aided Verification,* pp. 269-276, 1996.

[3] A. Aziz, V. Singhal, and F. Balarin, "It Usually Works: The Temporal Logic of Stochastic Systems," *Proc. Seventh Int'l Conf. Computer Aided Verification,* pp. 155-165, 1995.

[4] C. Baier, J. Katoen, and H. Hermanns, "Approximate Symbolic Model Checking of Continuous-Time Markov Chains," *Proc. 10th Int'l Conf. Concurrency Theory,* pp. 146-162, 1999.

[5] B.A. Berg, *Markov Chain Monte Carlo Simulation and Their Statistical Analysis.* World Scientific Publishing Company, 2004.

[6] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic Model Checking Without BDDs," *Proc. Fifth Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems,* pp. 193-207, 1999.

[7] J. Büchi, "On a Decision Method in Restricted Second Order Arithmetic," *Proc. Int'l Conf. Logic, Methodology and Philosophy of Science,* pp. 1-11, 1960.

[8] P. Bucholz, "Exact and Ordinary Lumpability in Finite Markov Chains," *J. Applied Probability,* vol. 31, pp. 59-74, 1994.

[9] M. Calder, A. Duguid, S. Gilmore, and J. Hillston, "Stronger Computational Modelling of Signalling Pathways Using Both Continuous and Discrete-State Methods," *Computational Methods in Systems Biology,* pp. 63-77, Springer-Verlag, 2006.

[10] G. Ciardo, R.A. Marie, B. Sericola, and K.S. Trivedi, "Performability Analysis Using Semi-Markov Reward Process," *IEEE Trans. Computers,* vol. 39, no. 10, pp. 1251-1264, Oct. 1990.

[11] G. Ciardo and A.S. Miner, "A Data Structure for the Efficient Kronecker Solution of GSPNs," *Proc. Int'l Workshop Petri Nets and Performance Models,* pp. 22-31, 1999.

[12] E. Clarke and E. Emerson, "Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic," *Proc. Workshop Logic of Programs,* 1981.

[13] E. Clarke, E. Emerson, and A. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logics Specification: A Practical Approach," *Proc. 10th Int'l ACM Symp. Principles of Programming Languages,* pp. 117-126, 1983.

[14] E. Clarke, O. Grumberg, and D. Peled, *Model Checking.* MIT Press, 2000.

[15] R. Grosu, E. Bartocci, F. Corradini, E. Entcheva, S. Smolka, and A. Wasilewska, "Learning and Detecting Emergent Behavior in Networks of Cardiac Myocytes," *Hybrid Systems: Computation and Control,* pp. 229-243, Springer-Verlag, 2008.

[16] H. Hansson and B. Jonsson, "A Logic for Reasoning about Time and Reliability," *Formal Aspects of Computing,* vol. 6, pp. 512-535, 1994.

[17] G.J. Holzmann, "The Model Checker Spin," *IEEE Trans. Software Eng.,* vol. 23, no. 5, pp. 279-295, May 1997.

[18] G. Hughes and M. Creswell, *Introduction to Modal Logic.* Methuen, 1997.

[19] J.G. Kemeny and J.L. Snell, *Finite Markov Chains.* Springer-Verlag, 1976.

[20] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 2.0: A Tool for Probabilistic Model Checking," *Proc. Int'l Conf. Quantitative Evaluation of Systems,* pp. 322-323, 2004.

[21] Y. Kwon and G. Agha, "Linear Inequality LTL (iLTL): A Model Checker for Discrete Time Markov Chains," *Proc. Int'l Conf. Formal Eng. Methods,* pp. 194-208, 2004.

[22] Y. Kwon and G. Agha, "iLTLChecker: A Probabilistic Model Checker for Multiple DTMCs," *Proc. Int'l Conf. Quantitative Evaluation of Systems,* 2005.

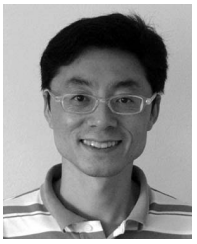
[23] Y. Kwon and G. Agha, "Scalable Modeling and Performance Evaluation of Wireless Sensor Networks," *Proc. IEEE Real-Time and Embedded Technology and Applications Symp.,* 2006.

[24] Y. Kwon and G. Agha, "A Markov Reward Model for Software Reliability," *Proc. Next Generation Software Workshop at the IEEE Int'l Parallel and Distributed Processing Symp.,* pp. 1-6, 2007.

[25] Y. Kwon and G. Agha, "LTLC: Linear Temporal Logic for Control," *Hybrid Systems: Computation and Control,* pp. 316-329, Springer-Verlag, 2008.

[26] V.V. Lam, P. Buchholz, and W.H. Sanders, "A Structured Path-Based Approach for Computing Transient Rewards for Large CTMCs," *Proc. Int'l Conf. Quantitative Evaluation of Systems,* pp. 136-145, 2004.

[27] O. Lichtenstein and A. Pnueli, "Checking That Finite State Concurrent Programs Satisfy Their Linear Specification," *Proc. 12th ACM Symp. Principles of Programming Languages*, pp. 97-107, 1985.
[28] D.G. Luenberger, *Linear and Nonlinear Programming*, second ed., Addison Wesley, 1989.
[29] J. Martyna, "Performance Modeling of Mobile Sensor Networks," *Ad-Hoc, Mobile, and Wireless Networks*, pp. 262-272, Springer-Verlag, 2007.
[30] A.S. Miner and G. Ciardo, "Efficient Reachability Set Generation and Storage Using Decision Diagrams," *Proc. Int'l Conf. Application and Theory of Petri Nets*, pp. 6-25, 1999.
[31] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, third ed. McGraw-Hill, 1991.
[32] H. Pham, *Software Reliability*. Springer, 2000.
[33] J. Rajgopal and M. Mazumdar, "Modular Operational Test Plans for Inference on Software Reliability Based on a Markov Model," *IEEE Trans. Software Eng.*, vol. 28, no. 4, pp. 358-363, Apr. 2002.
[34] D.M. Roberts and N.A. Buckley, "Pharmacokinetic Considerations in Clinical Toxicology," *Clinical Pharmacokinetics*, vol. 46, pp. 897-939, 2007.
[35] L. Shargel and A.B.C. Yu, *Applied Biopharmaceutics and Pharmacokinetics*. Appleton-Century-Crofts, 1985.
[36] H. Start and J.W. Woods, *Probability and Random Processes with Applications to Signal Processing*, third ed., Prentice-Hall, 2002.
[37] G. Strang, *Linear Algebra and Its Applications*, third ed., Harcourt Brace Jovanovich, 1988.
[38] A.S. Tanenbaum, *Computer Networks*, fourth ed., Prentice Hall, 2003.
[39] S.M. Yacoub, B. Cukic, and H.H. Ammar, "Scenario-Based Reliability Analysis of Component-Based Software," *Proc. 10th Int'l Symp. Software Reliability Eng.*, pp. 22-31, 1999.



YoungMin Kwon received the BE degree in electrical engineering and the ME degree in mechatronics from Korea University and the PhD degree in computer science from the University of Illinois at Urbana-Champaign. He is a software development engineer at Microsoft Corporation. His research interests include model checking physical systems described in differential equations and developing mobile agent services and middleware services for Wireless Sensor Networks.

Gul Agha received the BS degree from the California Institute of Technology and the AM degree in psychology and the MS and PhD degrees in computer and communication science from the University of Michigan at Ann Arbor. He is a professor of computer science at the University of Illinois at Urbana-Champaign. He is a fellow of the IEEE, a past editor-in-chief of *IEEE Concurrency: Parallel, Distributed and Mobile Computing*, and of *ACM Computing Surveys*. He has supervised 22 PhD dissertations, given more than 30 keynote lectures, and published more than 180 research articles. He is best known for his work on the Actor model of concurrent and distributed programming.