Master Thesis

Model-Checking Markov Chains using Interval Arithmetic

Enno Ruijters

Master Thesis 13-15

Thesis submitted in partial fulfillment
of the requirements for the degree of Master of Science
of Operations Research at the Department of Knowledge
Engineering of the Maastricht University

Thesis Committee:

Dr. Pieter Collins
Dr. Gijs Schoenmakers

Maastricht University
Faculty of Humanities and Sciences
Department of Knowledge Engineering
Master Operations Research

August 19, 2013

Abstract

Model-checking is the process of determining whether a mathematical model obeys certain properties. Traditional model-checkers use floating-point arithmetic to calculate probabilities, which adds unknown errors to the results. This thesis describes the modification of the Markov Reward Model Checker (MRMC) to use interval arithmetic, providing rigorous error bounds on the results. In addition, the program was modified to allow model-checking of some time-varying models. The modified program is tested against the MRMC testsuite and a testcase from systems biology.

Chapter 1

Introduction

Model-checking is the process of determining whether a mathematical model of a physical system obeys certain properties. For example, it may be important to know whether certain states of the model are reachable, and what the probability is of reaching such a state.

One important class of mathematical models are Markov Models. These models consist of a discrete set of states, and a set of transition probabilities or transition rates between these states. One of the defining properties of a Markov Model is that the transition probabilities or rates depend only on the current state, and not the path taken to reach that state.

Existing model-checking programs for Markov Chains, such as the Markov Reward Model Checker (MRMC)[8], Probabilistic Symbolic Model checker (PRISM)[9], and VESTA (Verification of Simulations for Timed Automata) [12] rely on floating-point arithmetic for computing probabilities. As such, their results have an unspecified error bound. This is undesirable when analysing safety-critical systems or other situations where a high degree of confidence in the results is needed.

One approach to overcome the potential inaccuracy of floating-point arithmetic is to use 'interval arithmetic'. In interval arithmetic, numbers are represented by intervals which contain the true value, and computations yield intervals which are guaranteed to contain the true value of the result.

This thesis describes the modification of the MRMC program to use interval arithmetic instead of traditional floating-point arithmetic. MRMC is capable of analysing the transient and steady-state behaviour of continuous-time and discrete-time Markov Models. It can also perform these analyses on Markov Models extended with rewards or impulse rewards [3].

The modified program is called the Interval Markov Reward Model Checker (IMRMC) and provides rigorous error bounds on its results. It contains all the functionality of MRMC except for the analysis of models with impulse rewards. In addition to the functionality of MRMC, IMRMC has been extended to allow models in which the transition probabilities or rates change in a stepwise fashion at predetermined moments in time.

Chapter 2 of this thesis describes some preliminary concepts and introduces the notation that will be used in the remainder of the thesis. Chapter 3 explains the algorithms used for calculating certain functions in interval arithmetic. These functions are later used as parts of the actual model-checking algorithms. Chapter 4 describes the algorithms used for performing the model-checking. Chapter 5 provides a brief description of the implementation. In Chapter 6 some experiments are described, and the results of these experiments provided and explained. Finally, Chapter 7 summarizes the most important findings, draws conclusions, and offers some avenues for further research.

Chapter 2

Preliminaries

This chapter describes the basic concepts needed to explain the remainder of this thesis. First, Markov models are introduced. Then, systems of Temporal logic are described, these are then combined in an explanation of model-checking. Finally, the method of Interval Arithmetic is explained.

2.1 Markov models

Markov models are a special case of stochastic models: A Markov model consists of a countable set of states, and a set of probabilities or rates for transitions between states. The special property of a Markov chain is that the transition probabilities or rates depend only on the current state and the state to be transitioned to.

Markov models can be discrete-time or continuous-time. In a discrete-time model, transition can occur only at fixed, regularly spaced moments in time. These models are characterized by a transition matrix Q, in which each element q_{ij} represents the probability of transitioning from state i to state j at one of these transition moments. It is convenient to define that the model undergoes exactly one transition each timestep, even if that transition is from a state to that same state. This implies that $\sum_j q_{ij} = 1$ for all i. This definition has the property that, if x_t is a vector of the probabilities of being in each of the states at time t, $x_{t+1} = x_t Q$.

Formally, a discrete-time Markov model is defined by:

- A state-space S of n states.
- A n-by-n transition probability matrix Q satisfying $\forall_i \sum_j q_{ij} = 1$ and $\forall_{i,j} q_{ij} \in [0, 1]$.

One run of a Markov model yields a 'path' of the states visited during the run. Such a path is defined as $S = s_0, s_1, ...$ where s_n denotes the state of the model after n transitions.

In a continuous-time model, transition can occur at any time. These models are characterized by a rate matrix R, in which each element r_{ij} represents the expected number of transitions from state i to state j in a unit time interval. The time between transitions is assumed to be exponentially distributed. Since transitioning from a state to itself is in indistinguishable from not transition at all, the diagonal entries of R can be arbitrarily defined. A convenient definition is to choose the diagonal entries such that $\sum_j r_{ij} = 0$. In this definition, the time-derivative of a probability distribution vector x_t is computed as $x_t R$, which will be used in Section 4.3.2.

Formally, a Markov (reward) model is defined by:

- A state-space S of n states.
- A n-by-n rate matrix Q satisfying $\forall_i \sum_j r_{ij} = 0$ and $\forall_{i \neq j} r_{ij} >= 0$.

Similar to a discrete-time model, we can define a path through the model. For a continuous-time model, the definition of such a path is $S = (s_0, \delta_0), (s_1, \delta_1), ...$, where s_n is the state of the model after n transitions, and δ_n is the time between the n^{th} transition and the transition after that.

Either model can also be extended to have 'rewards'. A reward is a state-dependent value which is accumulated for the time spent in that state. For a discrete-time model, the reward of the current state is added to the total reward each timestep. For a continuous-time model, the total reward is the time spent in each state, multiplied by the reward for that state.

2.1.1 Time-varying models

Some systems cannot be correctly modeled as a Markov model because the transition probabilities vary over time.

IMRMC has the ability to model some of these systems regardless: If the system can be modeled as a Markov-like model in which the transition probabilities or rates make stepwise changes at fixed, predetermined times. In the remainder of this thesis, such a model will be referred to as a 'Stepwise Markov Clock Model'.

Discrete-time models of this type can be specified by having multiple instances of each state, and forced transitions from each of instance of the state to the next at predefined times. This approach, however, can result in models with very large state spaces. A more specialized approach can therefore result in better performance.

In this case the system can be specified using the duration of each model, and the transition probabilities or rates during this time. If all the durations are finite, the model returns to the first set of transitions. If the last duration is infinite, the last set of probabilities is used indefinitely.

Formally, a Stepwise Markov Clock Model is defined as:

- A state-space S of n states.
- A sequence of m transition matrices $Q_0, Q_1, ..., Q_{m-1}$ or rate matrices $R_0, R_1, ..., R_{m-1}$
- A sequence of m corresponding times during which each of the matrices is active: $t_0, t_1, ..., t_{m-1}$ satisfying $\forall_i t_i >= 0$.

For discrete-time models, the times t_i are rounded up to integer values.

2.2 Temporal logics

Temporal logics are systems of formal logic that allow reasoning about propositions that include time and may have different truth values at different times. For example, the statement 'it is raining' is sometimes true, and sometimes false. The Linear Temporal Logic (LTL)[10] allows the formulation of statements such as 'it is *always* raining' or 'it will be raining tomorrow'.

Some temporal logics also allow statements about probabilities. An example of such a statement is 'the probability that it will be raining tomorrow is greater than 0.5'.

The logics used in IMRMC are Continuous Stochastic Logic (CLS) and Probabilistic Computational Tree Logic (PCTL)[1]. These are described below.

In general, these logics are defined for a time-evolving model M and a state s from the set of states S.

Atomic propositions have a truth value determined by the state of the model. This is described by a labeling function $\mathbb{L}$ where $\mathbb{L}(s)$ is the set of all atomic propositions that are true in state s. All other atomic propositions are false in state s.

In CSL, the set of statements (Ψ) and the set of probabilistic statements (Φ) consist of:

- For every atomic proposition p, "p" $\in \Psi$.

- For all statements $\alpha \in \Psi$ and $\beta \in \Psi$, and all intervals of time $T \subset \mathbb{R}_{\geq 0}$, the statements "$\alpha U \beta$" *(until)* and "$\alpha U^T \beta$" *(bounded until)* are elements of Φ.
- For all statements $\alpha \in \Psi$ and $\beta \in \Psi$, all intervals of probabilities $J \subset [0,1]$, and all statements $\phi \in \Phi$, the following statements are in Ψ: "$\neg\alpha$", "$\alpha \wedge \beta$", "$L_J(\alpha)$" *(long-run probability)*, and "$P_J(\phi)$" *(probability)*.

The interpretation of the statements in Ψ and Φ is as follows:

- $(M, s) \models p$ iff $p \in \mathbb{L}(s)$
- $(M, s) \models \neg\alpha$ iff $\neg((M, s) \models \alpha)$
- $(M, s) \models \alpha \wedge \beta$ iff $(M, s) \models \alpha$ and $(M, s) \models \beta$
- $(M, s) \models L_J(\alpha)$ iff the *long-run* probability of α being true given that the model starts in s lies within the interval J. That is, given a path s' starting with s, $P\left(\lim_{x\to\infty}(M, s'_x) \models \alpha\right) \in J$.
- $s \models P_J(\phi)$ iff the *probability* of ϕ being true given that the model starts in s lies within the interval J. That is, given the stochastic future evolutions M' of the model, $P((M', s) \models \phi) \in J$.
- $(M, s) \models \alpha U \beta$ iff α is true and remains true until β becomes true, and β will eventually become true. Formally, for some integer i and the path through the model $s_0 = s, S_1, ..., \forall_{j<i} : (M, s_j) \models \alpha$ and $(M, s_i) \models \beta$
- $(M, s) \models \alpha U^T \beta$ iff $\alpha U \beta$ is true and the transition from α to β occurs at some time within the interval T. Formally, given the path through the model $s_0 = s, s_1, ...$ and some integer $i \in T$, $\forall_{j<i} : (M, s_j) \models \alpha$ and $(M, s_i) \models \beta$.

The long-run operator L_J is only well-defined for models that eventually converge to a stationary distribution.

PCTL defines the same statements, except that it does not define the long-term operator, and has statements $P_J(\Box\alpha)$ defined such that $(M, s) \models P_J(\Box\alpha)$ iff for all paths $s_0 = s, s_1, ...$ through the model, $\forall_{i\geq 0} : s_i \models \alpha$.

For continuous-time models, only the bounded until-formula has a different interpretation, namely that the time interval does not refer to timesteps, but instead to the total elapsed time.

To illustrate these families of logic and their application to model-checking, some examples are provided in Section 2.4.

2.3 Model-Checking

The process of model-checking Markov model is determining whether a certain statement is true about the paths through the model. Since most Markov models are probabilistic, most of these statements are about the probabilities of these paths satisfying the requested property.

The property to be checked is typically phrased in some temporal logic. MRMC allows statements to be checked to be specified in Continuous Stochastic Logic (CSL), or Probabilistic Computational Tree Logic (PCTL).

MRMC determines the validity of statements for each initial state, and then displays the states in which the requested statement is true.

2.4 Examples of model-checking

To illustrate these families of logic, some examples are listed below. These are based on the simple example model in Figure 2.1. This example system consists of three states, named 1, 2, and 3. In states 1 and 2, the atomic proposition p is true. In state 3, the atomic proposition q is true. States 1 and 3 are *absorbing* (i.e. there are no transition from either to these states to any other state). If the system is in state 2 at a certain time, there is a probability of 0.1 that is will transition to state 1 after this timestep, a probability of 0.4 that it will transition to state 3, and a 50% chance that it will remain in state 2.

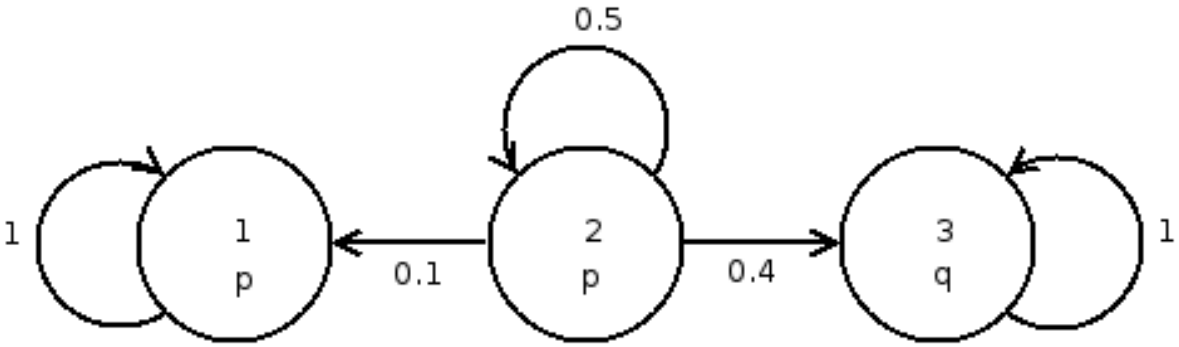


Figure 2.1: Example discrete-time Markov Model

Some example of logical statements about this system are:

- p: True for initial states 1 and 2, false for initial state 3.
- $P_{[0,0.5]}(p\ U^{[0,2]}\ q)$: This statement means 'The probability that q becomes true within two timesteps, and that p remains true until that time, is smaller than or equal to 0.5'. For initial state 1, it is impossible to reach a state in which q holds, so the probability of $p\ U^{[0,2]}\ q$ is 0, and the proposition is false. For initial state 3, q is immediately true, so $P(p\ U^{[0,2]}\ q) = 1$, and the statement is true. For initial state 2, within 2 timesteps there is a probability of 0.25 that the system remains in state 2, a probability of 0.6 that the system reaches state 3, and a probability of 0.15 that the system transitions to state 1. Since $p\ U^{[0,2]}\ q$ is only true if the system reaches state 3 within 2 timesteps, $P(p\ U^{[0,2]}\ q) = 0.6$, and the statement is false.
- $L_{[0.5,1]}(q)$: This proposition states that, in the long-run distribution, q holds with probability greater than or equal to 0.5. Starting from state 1, q will never be true, so the proposition is false. Starting from state 3, q will always be true and the proposition is true. Starting from state 2, in the long run there is a probability of 0.2 of being in state 1, and a probability of 0.8 of being in state 3. Since q is only true in state 3, $L(q) = 0.8$ and the proposition is true.
- $P_{[0,0.5]}(p\ U\ L_{[0.5,1]}(q))$: This statement is more complicated. In English, it says that the probability of the following statement is smaller than or equal to 0.5: 'Eventually the model reaches a state in which $L_{[0,0.5]}(q)$ is true, and p is true until that time.' As stated above, $L_{[0,0.5]}(q)$ is true for states 2 and 3, and false for state 1. Thus, the statement can be rephrased to 'the probability is smaller than or equal to 0.5 that the model reaches either state 2 or 3, and that p remains true until then'.

  Starting in state 1, neither state 2 nor state 3 can be reached, so $P(p\ U\ L_{[0.5,1]}(q))) = 0$, and the proposition is true. From states 2 and 3, $L_{[0.5,1]}(q)$ is immediately true, so the entire proposition is true for these initial states.

2.5 Interval Arithmetic

Interval arithmetic is a way of performing arithmetic on floating-point numbers without risking an undetected loss of accuracy. It provides a guaranteed bound on the accuracy of the final result.

In floating-point arithmetic, numbers are represented in finite precision. This finite precision implies that almost all real numbers are only approximately representable in the floating-point domain. When calculations are performed on floating-point numbers, if the result is not exactly representable, the nearest representable number is usually chosen. In this way, each calculation can introduce a slight error due to rounding, which can accumulate to give potentially misleading results.

An example of such a rounding error is called 'catastrophic cancellation', where two nearly equal numbers are subtracted, and the result has a much lower accuracy. For example, the value '$2 + 2^{-60}$' cannot be represented in normal floating-point arithmetic, and the closest representable value is 2. Thus, the result of $(2 + 2^{-60}) - 2$ is 0 instead of 2^{-60}.

Interval arithmetic provides a way for these errors to be tracked throughout a program, providing rigorous error bounds on the final result. This is achieved by representing a number as two floating-point numbers: A lower bound (denoted $\underline{x}$) and an upper bound (denoted $\bar{x}$). By manipulating the direction in which computations round their intermediate results, it is possible to perform computations with a guarantee that the result satisfies $\underline{f(x)} <= f(x) <= \overline{f(x)}$.

For example, given two intervals $a = [\underline{a}, \bar{a}]$ and $b = [\underline{b}, \bar{b}]$, their sum ($a + b = [\underline{a+b}, \overline{a+b}]$) can be computed by:

$$\underline{a+b} = \underline{a} + \underline{b} \text{ (rounded downwards)}$$
$$\overline{a+b} = \overline{a} + \overline{b} \text{ (rounded upwards)}$$

.

On systems with hardware compliant with the IEEE 754 standard for floating-point arithmetic, and supporting at least upward rounding and downward rounding, the following operations can be easily implemented in software:

- Addition and subtraction can be performed in two floating-point additions or subtractions, respectively.
- Square root can be performed in one comparison to ensure that the argument is nonnegative, and one floating-point square root.
- Multiplication and division can be performed, but the number of primitive operations varies depending on the intervals. If neither interval contains the value 0, they can be performed in two floating-point operations, with two to four comparisons needed to determine that neither interval straddles 0. If both intervals include both positive and negative values, four floating-point operations are needed, as well as six comparisons.

More complex algorithms such as exponentiation or trigonometry require special algorithms.

To provide accurate bounds on the final intervals, it is often necessary to modify the algorithms used. For example, algorithms computing approximations of Taylor series must also compute a bounds on the truncation error caused by using only a finite number of terms. Similarly, algorithms using iterative approximation such as Newton's method need to be replaced.

Chapter 3

Overview of primitive operations

This chapter describes the basic operations performed in the model-checker, and how they are implemented in interval arithmetic.

Section 3.1 explains the matrix exponential, which is used to check continuous-time models without discretization. Section 3.2 describes the exponential function of real-values intervals, which is used by IMRMC to perform the discretization of continuous-time models (see Section 4.3.1). Finally, Section 3.3 described the interval Gauss-Seidel method used for analyzing discrete-time models.

3.1 Matrix exponential

The matrix exponential is computed using a Taylor series approximation. For an n-by-n matrix A and an integer m, we define:

$$\exp(A, m) = \sum_{k=0}^{m} \frac{1}{k!} A^k$$

Now, if we choose $m > \|A\|_\infty$, correct interval matrix exponential can be obtained through:

$$e^A = \exp(A, m) + [-E, E] \frac{\|A\|_\infty^{m+1}}{(m+1)!(1 - \frac{\|A\|_\infty}{m+2})} \tag{3.1}$$

Where the matrix E has the value 1 at all entries[6].

When model-checking Markov chains, it is possible that specific states of the model are never reachable from certain other states. These entries of the matrix will therefore be exactly zero, and are called structural zeroes.

To determine structural zeroes, IMRMC chooses $m > n$. This way, all nonzero elements of e^A will have some nonzero interval in $\exp(A, m)$. All zero elements in $\exp(A, m)$ are therefore structural zeroes and not subject to inaccuracy of computation.

To ensure rapid convergence of the Taylor series, if $\|A\|_\infty > 0.25$, the exponential is computed by halving each element of the matrix, and squaring the exponential of the resulting matrix. This follows the identity $e^A = (e^{\frac{A}{2}})^2$.

3.2 Real exponential function

Thanks to the monotonicity of the exponential function for real-values arguments, a relatively straightforward implementation of the Taylor series expansion of the exponential function can be used for exponentials of positive arguments. For negative arguments, the identity $e^x = \frac{1}{e^{-x}}$ is used.

A lower bound of e^x can be computed as

$$e^x > \sum_{k=0}^{n} \frac{x^k}{k!}$$

where at each iteration x^k is rounded down, $k!$ is rounded up if needed, and the total sum is rounded down. For computing the lower bound of an interval, the Taylor series expansion is performed on the lower bound of the interval.

An upper bound can be calculated by first computing the Taylor series expansion above, but reversing the rounding modes and using the upper bound of the original interval. An overestimate of the remainder of the Taylor series can be computed using the bound equation for the matrix exponential in Equation 3.1 of a one-element matrix. Some algebraic rearrangement and the observation that only the upper bound is required, yields the equation:

$$e^x < \sum_{k=0}^{n} \frac{x^k}{k!} + \frac{(2+n)x^{n+1}}{(2+n-x)(n+1)!}$$

To ensure rapid convergence of the Taylor series, for values largen than 1, the exponential is computed using $e^x = e^{\lfloor x \rfloor} e^{x - \lfloor x \rfloor}$, where $e^{\lfloor x \rfloor}$ is computed by repeated squaring and multiplication, with a lookup table of exact values for $x \in \{1, 2, ..., 15\}$

3.3 Linear equation solving

For solving a linear system of equation, the original model-checker used either the Gauss-Seidel or Jacobi algorithm[2]. Both of these take an initial guess and repeatedly refine this guess until little improvement remains to be made.

For the interval model-checker, this form of repeated approximation is not suitable, as there is no guarantee that all valid results are contained in the final approximation. Instead, the interval Gauss-Seidel method is used[7]. In this method, the initial guess consists of a vector of intervals guaranteed to contain all solutions. Since the model-checker only solves linear equations involving probabilities, the initial guess consists of the interval $[0, 1]$ for all unknowns.

For solving the linear system $[A][x] = b$, the approximations $[x]^{(k)}$ are then repeatedly refined using:

$$[x]_i^{(k+1)} = [x]_i^{(k)} \cap \frac{1}{[a]_{ii}} \left(b_i - \sum_{j<i} [a]_{ij} [x]_j^{(k+1)} - \sum_{j>i} [a]_{ij} [x]_j^{(k)} \right)$$

This iteration is repeated until the width of the widest interval is smaller than a user-specified error bound, or until no further improvement is achieved.

The Gauss-Seidel algorithm is only guaranteed to converge if the matrix A is diagonally dominant (i.e. if $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$. Since most matrices will not be diagonally dominant, a preconditioning step is performed.

To precondition the system of equations, a matrix P is determined such that PA is diagonally dominant, and the system of equations is transformed from $Ax = b$ to $PAx = Pb$. This has no effect on the solutions x, except that it may have slightly wider intervals due to accumulated rounding errors.

As the matrix P, the inverse of A would be ideal, since $A^{-1}A$ is definitely diagonally dominant. However, since computing the inverse of an interval matrix very computationally intensive, an approximation is used: $P \approx (\frac{1}{2}(\bar{A} + \underline{A}))^{-1}$. This matrix is computing using normal floating-point arithmetic. Since the system $PAx = Pb$ has the same solutions as $Ax = b$ regardless of P, the accuracy of P is not critical.

Chapter 4

Model-checking algorithm description

In this chapter the algorithms used for the model-checker are described. Particular emphasis is placed on the algorithms that have been modified for the interval model-checker.

4.1 Discrete-time transient analysis

For discrete-time Markov chain with transition matrix Q, the probability distribution at timestep t can be determined by $x_t = x_0 Q^t$ where x_0 is the initial probability distribution.

In most cases, the value of interest is the probability of reaching any of a certain set of final states starting from each initial state. This can be easily computed as $r = Q^t y$ where y is 1 for each desired final state and 0 for all other entries. Solving this equation for r gives $r_i = \sum_j q_{ij}^t y_j$. In other words, r_i is the sum of the probabilities going from state i to a final state in t transitions.

MRMC conserves memory by computing r or x_t by repeatedly multiplying by Q (e.g.. $x_{i+1} = Q x_i$). For an n-by-n matrix Q, the computational complexity of this method is $O(n^2 t)$.

In many cases, Q will be a sparse matrix. If Q has m nonzero entries, the computational complexity of repeated multiplication is $O(mt)$.

To obtain tighter bounds, IMRMC computes Q^t by repeated squaring. While this is often faster and produces tighter bounds, that fact that Q^t is generally nonsparse may result in excessive memory consumption. It also makes the worst-case computational complexity independent of m. If this is a problem, the original method can still be used. The worst-case computational complexity of this method is $O(n^3 \log(t))$ regardless of m.

4.1.1 The until-formula

Formulae of the form $P(\phi\ U^{[0,t]}\ \psi)$ are solved by making all states satisfying ψ or $\neg\phi$ absorbing to obtain a matrix Q', and then computing the probability distribution at time t using $x_t = (Q')^t x_0$. The probability of $\phi\ U^{[0,t]}\ \psi$ is then the sum of the probabilities of having reached the states satisfying ψ. This algorithm is unchanged from MRMC to IMRMC.

If a lower bound is also provided, as in $P(\phi\ U^{[s,t]}\ \psi)$, in addition to the matrix Q', a matrix Q'' is created in which only states satisfying $\neg\phi$ are absorbing. The probability distribution at time t is then analyzed using $(Q'')^s (Q')^{t-s}$. MRMC did not implement the until-formula with a lower bound for discrete-time models.

If neither bound is provided, the steady-state distribution of Q' is determined, and the steady-state probability of being in any state satisfying ψ is the probability of $\phi\ U\ \psi$. Again, this algorithm was not changed in IMRMC.

4.1.2 Stepwise Markov clock models

For models which have multiple transition matrices at different times, the resulting model's distribution at time t may be determined by multiple matrices. In this case, the t-transition matrix is computed as $Q_1^{t_1}Q_2^{t_2}...Q_n^{t-t_1-t_2-\cdots}$ where $Q_1, Q_2, ...$ are the different transition matrices, and $t_1, t_2, ...$ are the times each of these matrices is active.

In addition, for the bounded-until formula with a lower bound, instead of computing $Q''^s Q'^{t-s}$, the product of the active matrices between time s and time t is computed:
$(Q'')_1^{t_1}(Q'')_2^{t_2}...(Q'')_n^{s-t_1-t_2-\cdots}(Q')_n^{t_n-(s-t_1-t_2)}(Q')_{n+1}^{t_{n+1}}...(Q')_m^{t-t_1-t_2-\cdots}$.

4.1.3 Reward models

When models with rewards are analyzed, and the rewards are relevant to the question, it is no longer sufficient to analyze the probability distributions at certain times. The entire path through the system is required.

In this case, the program enumerates all paths through the model in a breadth-first fashion, collapsing path prefixes ending in the same state with the same probability and accumulated reward. When an absorbing state is reached or it is determined that the path definitely satisfies the desired property or not, the remainder of the path is not enumerated.

This part of the MRMC was not significantly modified, except to use interval data types where appropriate.

4.2 Steady-state analysis

The steady-state probabilities for discrete-time and continuous-time Markov chains can be computed using the same method, by treating a discrete-time model as if it were a continuous-time model with rates equal to the transition probabilities.

To convert a discrete-time probability matrix to a rate matrix, the diagonal elements of the transition or rate matrix Q are replaced by the negation sum of the other elements in the row:

$$q_{ii} = -\sum_{j \neq i} q_{ij}$$

For a continuous-time rate matrix, this property is already true and the rate matrix can be used unmodified.

Now, since for a distribution x we can obtain the derivative of the distribution by $x' = xQ$, and since in the steady-state the derivative will be zero, the steady-state distribution can be found by solving the linear system of equations $xQ = 0$ or, equivalently, $Q^T x = 0$.

This system of equations has many solutions, because if $xQ = 0$, then $(kx)Q = 0$ also holds for any scalar k. In addition, the matrix Q is generally singular. This makes it difficult to solve using the Gauss-Seidel algorithm since preconditioning with the approximate inverse of the matrix does not work.

Both problems are solved by replacing all elements in the first row of Q by 1, as well as the corresponding element in the right-hand side of the equation. The new matrix is nonsingular and the only solution will be correctly normalized to sum to 1.

If the Markov chain in question has states which cannot be reached from certain other states (i.e. it is not ergodic), $xQ = 0$ no longer has a unique solution. this model is divided into several smaller models, each of which is ergodic. The steady-state distributions of the smaller models are then combined into one distribution, according to the desired initial probability distribution.

The steady-state analysis algorithm is mostly unchanged from MRMC to IMRMC, except that the linear equation solver has been replaced by the Gauss-Seidel iteration, as described in Section 3.3.

4.2.1 Stepwise Markov clock models

If the model changes over time, multiple definitions of the steady-state distribution exist. If the last transition matrix is active for an infinite period, the steady-state probability for this matrix can be computed for each of the initial states, and premultiplied by the transient probability of being in said state at the time when the last matrix becomes active.

If the model is cyclic, IMRMC computes the distribution the model is in after each cycle. In this case, the 1-cycle transition matrix is computed using either $Q_c = Q_1^{t_1} Q_2^{t_2} ... Q_n^{t_n}$ for discrete-time models, or $Q_c = e^{t_1 Q_1} e^{t_2 Q_2} ... e^{t_n Q_n}$ for discrete-time models. The steady-state distribution of this 1-cycle matrix is then computed as usual.

4.3 Continuous-time transient analysis

The transient analysis of a continuous-time Markov chain can be performed in several ways: by converting the model into a discrete-time model and performing the corresponding discrete-time analysis, by using the matrix exponential, or by enumerating all possible paths through the model. IMRMC can use either the discretization method or the matrix exponential. MRMC used path enumeration, but this was not included in IMRMC due to time constraints.

4.3.1 Discretization

A simple method for approximating a continuous-time model by a discrete-time model involves choosing a suitably small timestep h, and converting the transition rates into transition probabilities according to a Poisson distribution. The resulting approximation then has the same set of states, and the transition matrix $Q = (I + hR)$.

Unfortunately, this method ignores the possibility that two or more transition occur during one timestep. In addition, if the question to be answered concerns a time inbetween two timesteps, the result will be inexact. Although both of these problems can be mitigated by reducing the timestep length, they cannot be completely eliminated.

An alternative discretization system can be applied using interval arithmetic to give a discrete-time system of which the probability distribution at any time is guaranteed to contain the distribution of the continuous-time model at the corresponding time. For the remainder of this section, the rate matrix of the continuous-time model will be called R, and the transition probability matrix of the discretized model will be called Q. The discretization timestep will be h.

This discretization scheme converts an N-state continuous-time model into a (2N+1)-state discrete-time model (two additional states are added for solving the bounded until formula). In this new model, each state is defined not only by the corresponding continuous-time state, but also by whether or not a transition occurs from each state during this timestep. One additional state represents the condition that more than one transition has occurred during a timestep.

In the discretized model, the states $i = 1, ..., N$ correspond to remaining in a particular continuous-time state for an entire timestep. Each of these states has at most three transitions: the self-transition probability, the probability of exactly one outgoing transition out of the current state, and the probability of more than one transition. probability, the probability of one outgoing transition, and the probability of more than one transition.

The self-transition rate for state N is easily computed. Since the combined outgoing rate in state i is $r_i = \sum_{i \neq j} R_{ij}$, the probability of no transition during one timestep is $Q_{ii} = e^{-hr_i}$.

The states $i = N + 1, ..., 2N$ represent the states in which exactly one transition occurred during the timestep. The calculation of this probability is more difficult. It depends on the outgoing rates of the current state, as well as the outgoing rates of the destination states. State $N + i$ represents the state of one transition occuring out of state i in the current timestep, so the probability of one transition occuring during the next timestep is placed in $Q_{i,(N+i)}$.

The probability of one transition occurring during a timestep going from state i to state k can be computed as follows:

The probability can be conditioned on the transition time. Let the transition occurs at time t. It is then necessary that no other transition occurred before t, that the transition of interest occurred at time t, and that no transitions occur out of the new state between times t and h. Let r_i be the total outgoing rate from state i, and r_k the total outgoing rate from state k. By integrating over all possible possible transition times we obtain:

$$P(i \to k) = \int_0^h e^{-(r_i - R_{ik})t} R_{ik} e^{-R_{ik}t} e^{-r_k(h-t)} dt$$

Simplifying and summing over all destination states, we obtain the 1-transition probability:

$$Q_{i(N+i)} = \sum_{k \neq i} \frac{R_{ik}\left(e^{-r_k h} - e^{-r_i h}\right)}{r_i - r_k}$$

The probability of more than one transition occurring is simply taken to be $1 - Q_{ii} - Q_{i(N+i)}$.

The more-than-one-transition state (or 'fail state') is absorbing. To obtain correct intervals on the desired probabilities, the probability of reaching this state is added to the upper bounds (assuming the fail state does not satisfy the property of interest).

The outgoing probabilities from a state do not depend on whether the preceding timestep was a 0-transition or a 1-transition timestep. Therefore, the outgoing probabilities from the 1-transition state $N+i$ are the weighted sum of the outgoing probabilities from the 0-transition states of the destination states:

$$Q_{(N+i)x} = \frac{R_{ik}\left(e^{-r_k h} - e^{-r_i h}\right)}{Q_{kx}(r_i + r_k)}$$

This method could be generalized to include two-transition or N-transition probabilities if desired, but this functionality is not included in the current program.

To illustrate the discretization process, an example model is shown in Figure 4.1. The continuous-time model is given in Figure 4.1a, and the discretized model in Figure 4.1b. The discretization timestep is 0.1.

In the discretized model of the example, states 1, 2, and 3 correspond to remaining in their counterpart of the original model for an entire timestep. States 1a, 2a, and 3a are the transition states out of states 1, 2, and 3 respectively. State 'F' is the more-than-one-transition state.

All probabilities in the example are rounded to two significant figures. A derivation of the probabilities is shown below:

- Self-transition probability from state 1: $Q_{1,1} = \exp(-0.1 \cdot (2+1)) = 0.74[0;1]$.
- Self-transition probability from state 2: $Q_{2,2} = \exp(-0.1 \cdot (0.5+0.5)) = 0.90[4;5]$.
- State 3 has no outgoing transitions, and therefore remains absorbing in the discretized model ($Q_{3,3} = 1$).
- 1-transition probability from state 1: There are two outgoing transitions. For the transition to state 2, we obtain the 1-transition probability $P(1 \to 2) = \frac{2(\exp(-0.1\cdot(0.5+0.5)) - \exp(-0.1\cdot(1+2)))}{(1+2)-(0.5+0.5)} = 0.16[4;5]$. Similarly, for the transiton to state 3 we obtain $P(1 \to 3) = \frac{1(\exp(0) - \exp(-0.1\cdot(1+2)))}{(1+2)-0} = 0.08[6;7]$. Summing these, we obtain $Q_{1,1a} = 0.16[4;5] + 0.08[6;7] = 0.2[5;6]$.
- 1-transition probability from state 2: Again there are two outgoing probabilities, giving $P(2 \to 1) = \frac{0.5(\exp(-0.1\cdot(2+1)) - \exp(-0.1\cdot(0.5+0.5)))}{(0.5+0.5)-(1+2)} = 0.041[0;1]$ and $P(2 \to 3) = \frac{0.5(\exp(0) - \exp(-0.1\cdot(0.5+0.5)))}{(0.5+0.5)-0} = 0.047[5;6]$. Together these give $Q_{2,2a} = 0.041[0;1] + 0.047[5;6] = 0.08[8;9]$.

- Failure probability from state 1: $Q_{1,F} = 1 - Q_{1,1} - Q_{1,1a} = 1 - 0.7[4;5] - 0.2[5;6] = [-0.01; 0.01]$. Since we know the transition probability cannot be below 0, we set $Q_{1,F} = 0.0[0;1]$.
- Failure probability from state 2: $Q_{2,F} = 1 - Q_{2,2} - Q_{2,2a} = 1 - 0.9[0;1] - 0.08[8;9] = 0.0[01;12]$.
- During a timestep in state 1a, the continuous-time system will have experienced a transition to either state 2 or state 3. By dividing the previously calculated probabilities, we can determine that the probability of having transitioned to state 2 is $P(1 \to 2 \mid 1 \text{ transition}) = \frac{P(1\to 2)}{P(1 \text{ transition})} = \frac{0.16[4;5]}{0.2[5;6]} = 0.6[3;6]$. Similarly, $P(1 \to 3 \mid 1 \text{ transition}) = \frac{P(1\to 3)}{P(1 \text{ transition})} = \frac{0.08[6;7]}{0.2[5;6]} = 0.3[3;5]$. The outgoing probabilities from state 1a are now the weighted sum of the probabilities from states 2 and 3, since the probabilities are the same whether the system has just transitioned to a state or spent the entire previous timestep in that state. Therefore, $Q_{1a,2} = 0.6[3;6] \cdot Q_{2,2} + 0.3[3;5] \cdot Q_{3,2} = 0.6[3;6] \cdot 0.90[4;5] = 0.05[5;9]$. Similarly, $Q_{1a,2a} = 0.6[3;6] \cdot Q_{2,2a} + 0.3[3;5] \cdot Q_{3,2a} = 0.[59;60]$, $Q_{1a,3} = 0.6[3;6] \cdot Q_{2,3} + 0.3[3;5] \cdot Q_{3,3} = 0.3[3;5]$, and $Q_{1a,F} = 0.00[4;5]$.
- The outgoing probabilities from state 2a are calculated like for state 1a.
- The fail state is absorbing.
- Since state 3 is absorbing, state 3a is not needed.

If the formula to be solved requires the determination whether a property is satisfied in a given time interval, as opposed to at a given time, a slight modification is made to the discretizetion: Two additional states are added. One of these states is absorbing and represents that the desired property has been satisfied. The other represents that a transition will be made in the current timestep to the 'satisfied' state. The probability of being in the latter state is used to widen the final probability if the time of interest is not an exact multiple of the discretization timestep.

To obtain the probability distribution at time t from an initial state s_0, take the initial distribution x_0 to be row s_0 of Q. This includes the probabilities of one or more transition occurring in the first time step. Then compute $x_t = x_0 Q^{t/h}$. Finally, an accurate bound on the probability distribution y (with N states) can be obtained by $y_t(i) = x_t(i) \pm \sum_{j \geq N+1} x_t(j)$. It is possible to obtain better bounds by only including transition states with transition to or from state i, but this is not implemented in IMRMC.

It should be noted that, for models with very large outgoing rates, the discretization timestep required to obtain usefully narrow intervals may be very small. In these cases, the time requires to perform the subsequent discrete-time analysis may be unaccaptably long. In addition, because the resulting interval would be the result of many more calculations in the discrete-time algorithms, the resulting bounds may still be too wide.

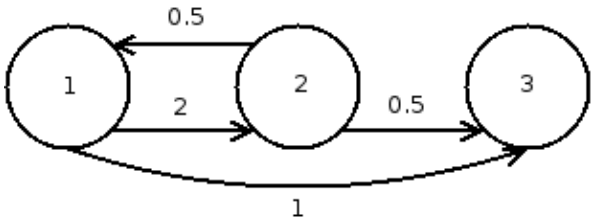
For time-varying models, each of the submodels is discretized separately. IMRMC only supports discretization of these models if the durations of each of the models is exact and divisible by the discretization timestep.

4.3.2 Exponentiation

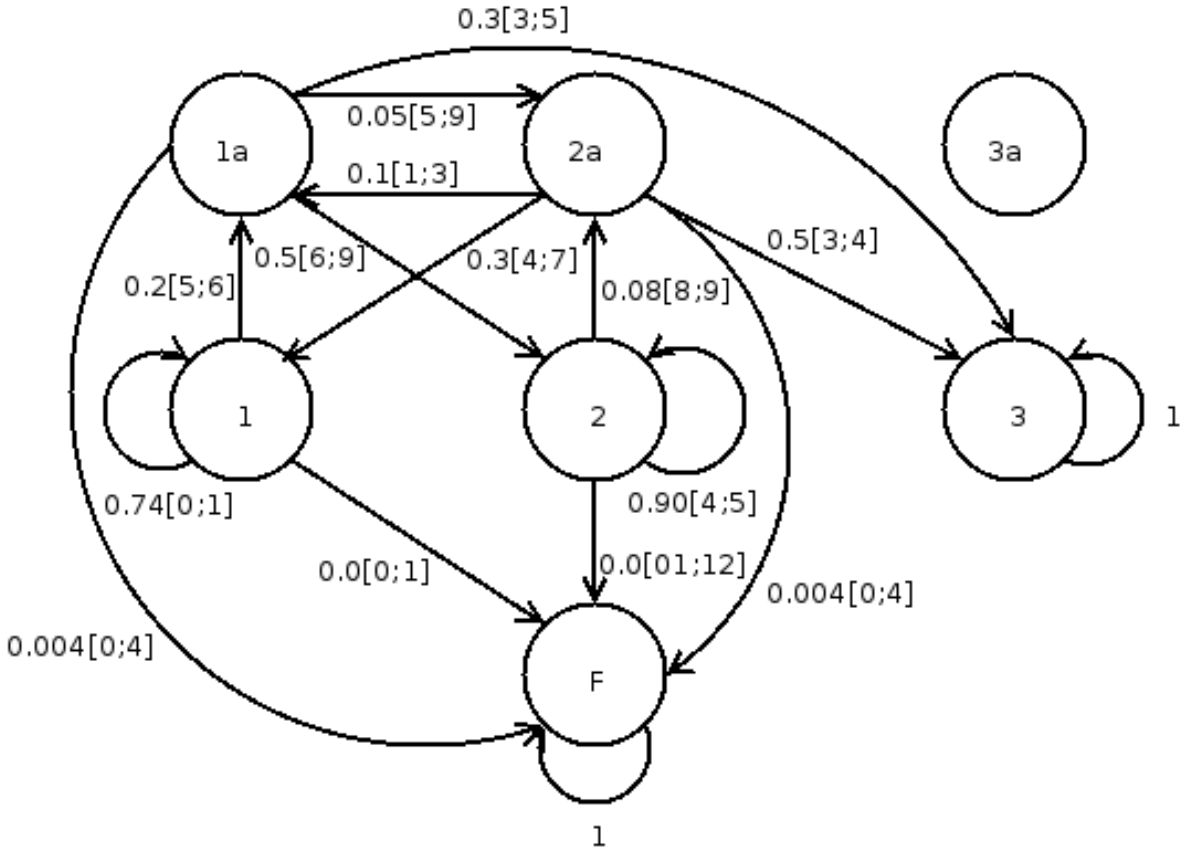
It is also possible to compute the probability distribution at time t using the matrix exponential of the rate matrix. In this case, the probability distribution at time t is $x_t = e^{Qt} x_0$. This method is used by default in IMRMC whenever possible since it usually yields tighter bounds.

This has the benefit that the matrix exponential can ofter be computed faster and with tighter bounds than the results of the discretized model, but it has the disadvantage that e^{At} is generally not sparse, and may not fit in memory.

For questions involving rewards, not only the distribution at time t is required, but also the path taken to reach the relevant states at that time. Since the matrix exponential loses this information, this approach is not suitable for these questions.



(a) Example model to be discretized



(b) Discretized model with timestep 0.1

Figure 4.1: Example of discretization

Chapter 5

Implementation

MRMC is written in C. To make the conversion to using interval arithmetic easier, IMRMC has been converted to C++. This allows the use of operator overloading to use the interval class as if it were a normal arithmetic type in most cases. Other than the interval class and some adaptation to C++ syntax, most of the code remains compatible with C.

The interval class uses two variables of type 'double' to represent the endpoints of an interval. Simple operations such as addition and multiplication are written using inline methods for best performance. Some assembly code is used to change the rounding mode of the computer's Floating Point Unit (FPU). When compiling IMRMC, it is possible to use 'long double' as the type for the interval bounds to increase accuracy, though this will reduce performance.

For most operations the rounding mode is only changed to upward rounding. Downward rounding is achieved by instead calculating the negative of the desired result using upward rounding, and then inverting the sign of the result. Since sign inversion is an exact operation, this has the same effect as computing value while rounding downward. Due to some incorrect optimization of the GCC compiler, it was necessary to make some intermediate variables 'volatile', inhibiting optimization of these variables.

The code structure of MRMC has mostly been maintained. Parsing and model-checking are largely separated, and complex operations such as linear equation solving and matrix exponentials have been placed in separate files. The approximate matrix inverse for the Gauss-Seidel algorithm is performed using the GNU Scientific Library[5].

Chapter 6

Experiments and Results

This chapter describes some of the tests performed to validate the performance and correctness of the modified model-checker. First some basic tests are described to explain the format of the results and give some indication of correct operation. After this several cases from the testsuite provided with the MRMC are described, and finally a real-life testcase from systems biology is presented.

6.1 Basic tests

One of the simplest tests concerns the three-state discrete-time model shown in Figure 2.1.

Models in IMRMC are represented by two files: a transition file which defined the markov model, and a label file which defines the atomic propositions. The transition file corresponding to this model is as follows (comments follow '%' signs):

```
STATES 3      % Number of states in model
TRANSITIONS 5 % Number of transitions in model
1 1 1         % Transition probability from state 1 to state 1
2 1 0.1       % Transition probability from state 2 to state 1
2 2 0.5       % ...
2 3 0.4
3 3 1
```

The label file is:

```
#DECLARATION
p q  % Which labels are used in the model.
#END
1 p  % Which labels apply to state 1
2 p  % ...
3 q
```

If the statement "$P\{>0.6\}[p\ U[0,2]\ q]$" is input to the IMRMC, it gives the following output:

```
$ERROR_BOUND: [1.000000e-20, 1.000000e-20]
$RESULT: ( [0.000000000000000000000000, 0.000000000000000000000000],
[0.599999999999999999132638, 0.600000000000000000758942],
[1.000000000000000000000000, 1.000000000000000000000000] )
$STATE: { (2), 3 }
```

The indicated error bound is the interval width at which the Gauss-Seidel iteration ends, and is not relevant for this particular question.

In this particular problem, states 1 and 3 have an exact value. The probability for state 2 is computed as an interval containing the correct solution (0.6), with a precision better than 10^{-19}. Since this is a very simple model and a question about a very short period of time, the precision is better than it would be for more common problems.

The output of the original MRMC for the same problem is:

```
$ERROR_BOUND: 1.000000e-20
$RESULT: ( 0.00000000000000000000, 0.60000000000000088818,
1.00000000000000000000 )
$STATE: { 2, 3 }
```

As can be seen, the original MRMC computes an exact, but slightly incorrect probability.

Finally, the program displays the initial states for which the proposition is true. In this case, it is definitely true for state 3, and possibly true for state 2. The parentheses around state 2 indicate that the statement has neither been proven correct nor incorrect from this state. The original MRMC incorrectly indicates that the proposition is true.

For testing the long-term operator, the proposition $L_{>0.8}[q]$ is input. The resulting probability from state 2 is given as [0.79999999999999993, 0.8000000000000001], matching the correct answer of 0.8.

To test the continuous-time solver, the same model with the probabilities interpreted as rates, except that the self-transitions were removed (as self-transitions are implicit in continuous-time models).

The computed result for initial state 2 of the proposition $P_{<0.6}(p\ U^{[0,1]}\ q)$ with the discretization method is approximately 0.3147[75;83], containing the correct value $0.8(1 - e^{-0.5})$. The exponential method gives the result
0.3147754722298932[35;87], again agreeing with the correct value.

6.2 Effect of discretization timestep length

Table 6.1 shows the effect of changing the discretization timestep size on the accuracy of the computed probabilities. The model used is again the 3-state example model from Figure 2.1, and the query is $P(p\ U^{[0,1]}\ q)$.

Timestep length	Interval	Interval width
10^{-1}	0.[29;36]	$7.1 \cdot 10^{-2}$
$5 \cdot 10^{-2}$	0.3[02;38]	$3.6 \cdot 10^{-2}$
$2.5 \cdot 10^{-2}$	0.3[08;27]	$1.8 \cdot 10^{-2}$
10^{-2}	0.31[23;96]	$7.3 \cdot 10^{-3}$
$5 \cdot 10^{-3}$	0.31[36;72]	$3.6 \cdot 10^{-3}$
10^{-3}	0.31[45;53]	$7.3 \cdot 10^{-4}$
10^{-4}	0.314[75;82]	$7.3 \cdot 10^{-5}$
10^{-5}	0.3147[73;80]	$7.3 \cdot 10^{-6}$
10^{-6}	0.314775[23;96]	$7.3 \cdot 10^{-7}$
10^{-7}	0.314775[45;52]	$7.9 \cdot 10^{-8}$
10^{-8}	0.314775[46;53]	$6.1 \cdot 10^{-8}$
10^{-9}	0.31477[52;60]	$8.0 \cdot 10^{-7}$

Table 6.1: Effect of discretization timestep on CTMC model-checking precision

As can be seen, the precision increases approximately linearly with the discretization timestep until the timestep becomes very small. Eventually, a balance is reached between the discretization error and the accumulated error of the larger number of calculations required for the DTMC checker.

6.3 Performance tests on discrete-time models

To compare the performance of IMRMC to that of MRMC, several testcases were executed in both programs. The results are shown in the tables below. The tables show the running times of the programs, as well as the factor by which IMRMC is slower than MRMC, and the width of the interval produced by IMRMC. This width is specified in *units in last place* or 'ulp'. One ulp is the difference between the upper bound of the interval and the largest number smaller than this upper bound, that is still representable by a double-precision floating point value.

First, a 500-state model was analyzed in which every state has an equal probability of transitioning to every other state. The running times are shown in table 6.2. For the purposes of this test, MRMC was modified to also allow the calculation of matrix powers by repeated squaring.

Proposition	MRMC	IMRMC	Slowdown factor	Interval width
$P_{[0,0.5]}(\phi\ U^{[0,10]}\ \psi)$ (matrix power)	1854 ms	10674 ms	5.8	977 ulp
$P_{[0,0.5]}(\phi\ U^{[0,10]}\ \psi)$ (premultipliation)	8 ms	29 ms	4	927 ulp
$P_{[0,0.5]}(\phi\ U^{[0,20000]}\ \psi)$ (matrix power)	8284 ms	48272 ms	5.8	71471 ulp
$P_{[0,0.5]}(\phi\ U^{[0,20000]}\ \psi)$ (premultiplication)	15345 ms	65584 ms	4.3	166492 ulp
$L_{[0,0.5]}(\psi)$	2.6 ms	1017 ms	400	178 ulp

Table 6.2: Running times of MRMC and IMRMC on DTMC models

Clearly, IMRMC is approximately a factor of five slower than the unmodified MRMC. A significant portion of this is due to the slow implementation of interval arithmetic. A matrix-matrix multiplication of a full, 100-by-100 matrix consumes 4.5 ms without interval arithmetic and 18.2 ms with interval arithmetic, giving a slowdown factor of approximately 4. Considerable improvement can probably be achieved here, but erroneous optimizations by the gcc compiler have prevented a more efficient implementation from being included in IMRMC.

A more significant slowdown can be seen in the long-run operator. This is likely due to the use of a different algorithm. In addition, the Gauss-Seidel implementation in IMRMC is not very optimized. It is therefore likely this performance can be improved.

The accuracy of the two different methods for computing the probability distribution do not differ much for short times, but the matrix power algorithm is clearly more accurate for longer times.

To test performance on models with a high degree of sparsity, a model was constructed with 501 states: state 1 has a random transition probability to every state. The remaining states are divided into 5 components of 100 states each. Each state has a random transition probability to each other state within the same component, and a probability of 0 to transition to any other component. Running times are listed in table 6.3.

Proposition	MRMC	IMRMC	Slowdown factor	Interval width
$P_{[0,0.5]}(\phi\ U^{[0,10]}\ \psi)$ (matrix power)	79 ms	405 ms	5.1	254 ulp
$P_{[0,0.5]}(\phi\ U^{[0,10]}\ \psi)$ (premultipliation)	0.4 ms	1.5 ms	3.8	249 ulp
$P_{[0,0.5]}(\phi\ U^{[0,20000]}\ \psi)$ (matrix power)	349 ms	2021 ms	5.8	7122 ulp
$P_{[0,0.5]}(\phi\ U^{[0,20000]}\ \psi)$ (premultiplication)	636 ms	3022 ms	4.8	1706 ulp
$L_{[0,0.5]}(\psi)$	0.26 ms	295 ms	590	172 ulp

Table 6.3: Running times of MRMC and IMRMC on a sparse DTMC

Clearly, a sparse model results in a considerably lower running time than a full model with the same number of states.

Finally, a model was constructed with 500 states in which every state has as transition probability of 0.5 to remain in that state, and a probability of 0.5 to transition to the state numbered one higher (or from state 500 back to state 1). This model has the property that, although the transition matrix

is very sparse, powers of that matrix gradually become very nonsparse. The running times are shown in Table 6.4

Proposition	MRMC	IMRMC	Slowdown factor	Interval width
$P_{[0,0.5]}(\phi\ U^{[0,10]}\ \psi)$ (matrix power)	21 ms	104 ms	5	exact
$P_{[0,0.5]}(\phi\ U^{[0,10]}\ \psi)$ (multiplication)	0.1 ms	0.3 ms	3	exact
$P_{[0,0.5]}(\phi\ U^{[0,10000]}\ \psi)$ (matrix power)	1199 ms	5975 ms	5	1 ulp
$P_{[0,0.5]}(\phi\ U^{[0,10000]}\ \psi)$ (multiplication)	50 ms	276 ms	5.5	1 ulp
$L_{[0,0.5]}(\psi)$	0.1 ms	497 ms	5000	exact

Table 6.4: Running times of MRMC and IMRMC on a sparse, fully connected DTMC

This shows that for these types of model, repeatedly multiplying a vector by the matrix is much faster than constructing the full 10000-transition matrix.

6.4 Performance tests on continuous-time models

The continuous-time algorithms take more time than the discrete-time case, so for these tests a model was used with 100 states, with a rate of 1 from every state to every state. The running times of the analyses are shown in table 6.5. The unmodified MRMC uses the uniformization algorithm[11].

Proposition	MRMC (uniformization)	IMRMC (exponential)	IMRMC (discretization)
$P_{[0,0.5]}(\phi\ U^{[0,1]}\ \psi)$	3 ms	2108 ms	8167 ms
$P_{[0,0.5]}(\phi\ U^{[0,10000]}\ \psi)$	10824 ms	2298 ms	14766 ms
$L_{[0,0.5]}(\psi)$	0.2 ms	12 ms	12 ms

Table 6.5: Running times of MRMC and IMRMC on CTMC models

It should be noted that discretization yielded uselessly large intervals for this model ([0.0028; 1]), however this does not affect the running time of the algorithms.

Again, IMRMC is considerably slower than MRMC. In addition, the discretization mode is much slower than analysis by matrix exponentials. Since discretization generally also produces wider intervals, the exponential algorithm is preferred whenever it can be used.

To determine how much of the slowdown is the result of using interval arithmetic, and how much is due to the algorithms used, the matrix exponential and discretization algorithms were also implemented without using intervals. A comparison of the running time of the interval version to that of the non-interval version is shown in table 6.6.

Proposition	non-interval	interval	slowdown factor	interval width
$P_{[0,0.5]}(\phi\ U^{[0,1]}\ \psi)$ [exponential]	484 ms	2108 ms	4.3	35973 ulp
$P_{[0,0.5]}(\phi\ U^{[0,10000]}\ \psi)$ [exponential]	529 ms	2298 ms	4.3	8464865 ulp
$P_{[0,0.5]}(\phi\ U^{[0,1]}\ \psi)$ [discretization]	268 ms	8167 ms	30	very large
$P_{[0,0.5]}(\phi\ U^{[0,10000]}\ \psi)$ [discretization]	702 ms	14766 ms	21	very large

Table 6.6: Running times of algorithms on CTMC models

As noted earlier, the discretization produces intervals which are completely useless in this case. In addition, the running times are considerably longer. The large increase in running time compared to the non-interval version is likely due to the ability of the non-interval version to compute scalar exponentials using one FPU instruction, while the interval version requires a special algorithm.

Like for the discrete-time, the CTMC algorithms were applied to a sparse model with 101 states and 5 ergodic components, with random transition probabilities. The running times are shown in Table 6.7.

Proposition	non-interval	interval	slowdown factor	interval width
$P_{[0,0.5]}(\phi\ U^{[0,1]}\ \psi)$ [exponential]	101 ms	433 ms	4.3	4998 ulp
$P_{[0,0.5]}(\phi\ U^{[0,10000]}\ \psi)$ [exponential]	108 ms	480 ms	4.4	4203958 ulp
$P_{[0,0.5]}(\phi\ U^{[0,1]}\ \psi)$ [discretization]	37 ms	1195 ms	33	very large
$P_{[0,0.5]}(\phi\ U^{[0,10000]}\ \psi)$ [discretization]	132 ms	3970 ms	30	very large
$L_{[0,0.5]}(\phi)$	0.06 ms	3 ms	50	52 ulp

Table 6.7: Running times of algorithms on a sparse CTMC model

Also, the algorithms were executed on a 100-state model where each state has a transition rate of 1 to the next state (and from the final state back to state 1). The results of this test are displayed in Table 6.8.

Proposition	non-interval	interval	slowdown factor	interval width
$P_{[0,0.5]}(\phi\ U^{[0,1]}\ \psi)$ [exponential]	60 ms	181 ms	3	40 ulp
$P_{[0,0.5]}(\phi\ U^{[0,10000]}\ \psi)$ [exponential]	86 ms	288 ms	3.3	131438 ulp
$P_{[0,0.5]}(\phi\ U^{[0,1]}\ \psi)$ [discretization]	24 ms	2171 ms	45	very large
$P_{[0,0.5]}(\phi\ U^{[0,10000]}\ \psi)$ [discretization]	385 ms	9671 ms	25	very large
$L_{[0,0.5]}(\phi)$	0.1 ms	7 ms	70	exact

Table 6.8: Running times of algorithms on a sparse, fully connected CTMC model

For all three testcases, discretization is slower than the matrix exponential. In addition, the matrix exponential algorithm yielded tighter bounds. This supports the recommendation to always use the matrix exponential algorithm if possible.

6.5 Systems biology testcase

To apply the modified program to a real-life testcase, a Markov model of a part of a human heartcell was simulated. This system models the slowly activating delayed-rectifier K^+ (I_{Ks}) channel. It can be open or closed depending on certain molecules that can attach or detach to the channel.

The channel has four receptors. Each of these receptors can bind one signaling molecule, and this signaling molecule in turn can bind a different molecule. The channel opens only when all eight molecules are attached. A diagram of the model is displayed in figure 6.1.

The probabilities of a molecule attaching or detaching depend on the voltage across the channel. This voltage in turn depends on several factors, including the history of the model. However, since the voltage in typical cases behaves in a known, cyclical pattern, the system can be approximated by only making the probabilities time-dependent.

The voltage profile across the channel behaves as shown in figure 6.2a. This figure shows 1000 ms of voltage, which is approximately a full cycle.

Several approximations of this profile were used to test IMRMC. First, a two-step average was computed. This voltage profile is shown in Figure 6.2b. Then, a more accurate 5-point average was determined as shown in Figure 6.2c. Finally, the minimum and maximum of the profile in each of 30 segments was determined to find intervals bounding the entire profile. This profile is shown in Figure 6.2d.

This model is so small none of the algorithms have a significant running time.

The steady-state distribution does not appear very interesting. State 1 has a very large difference between incoming and outgoing rates (more than 0.001 per millisecond incoming from state 2, but no

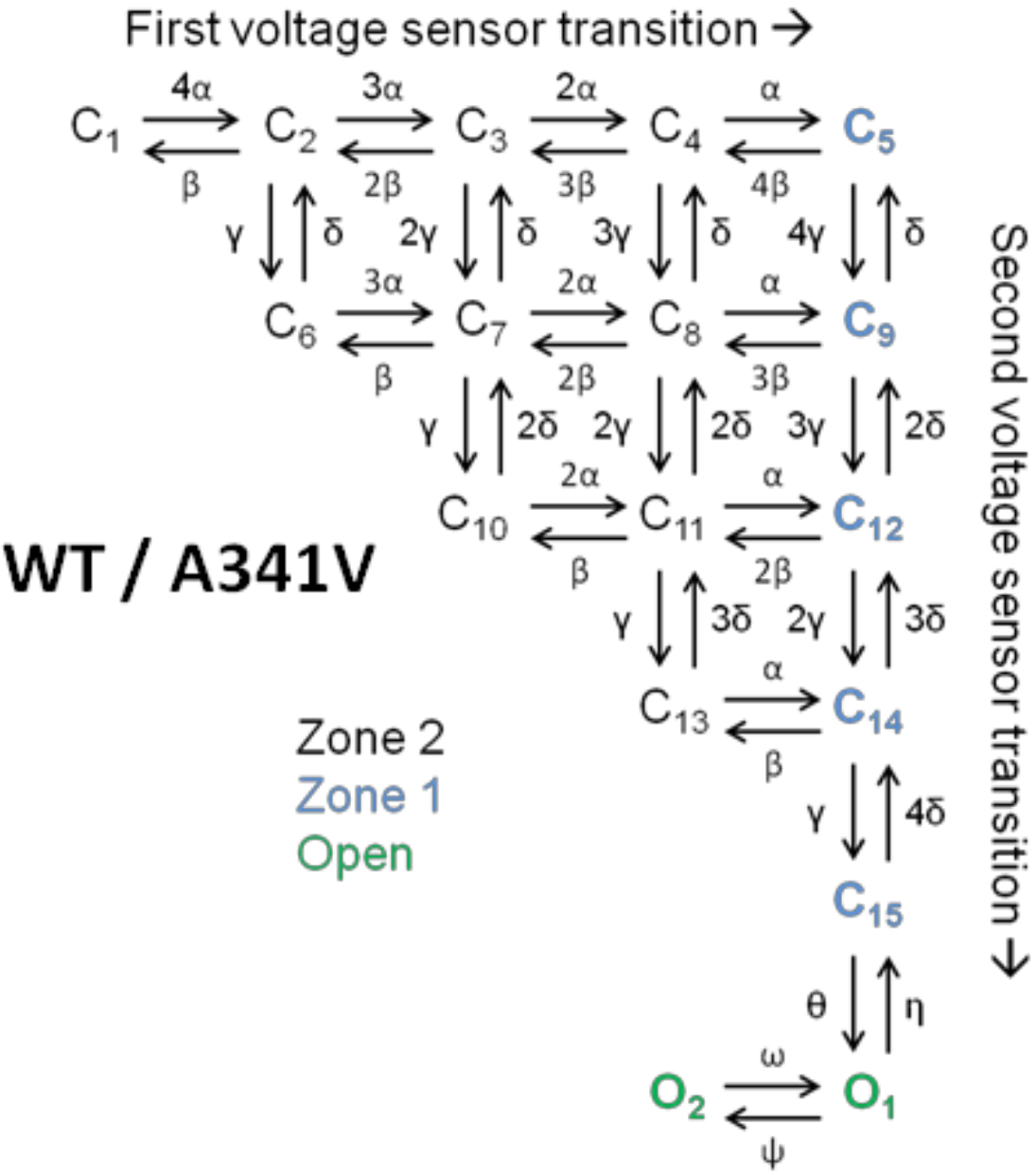
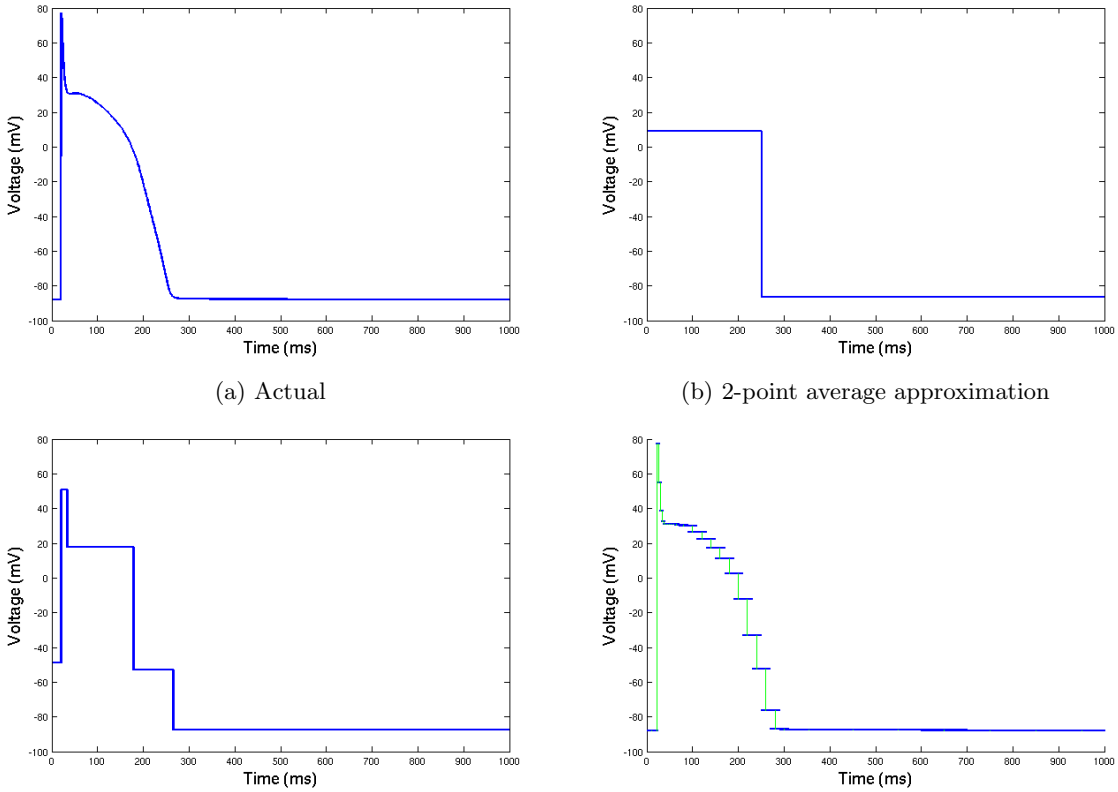


Figure 6.1: Structure of the model in the systems biology testcase[4]

more than $1 \cdot 10^{-16}$ outgoing). In the five-point average, this leads to a steady-state distribution where the probability of being in state 1 is almost 1, and the next most likely state is state 6 with a probability of $1.20[30; 47] \cdot 10^{-13}$. The two-point approximation raises this number to $2.604[36; 86] \cdot 10^{-12}$, indicating that the model is sensitive to the quality of the approximation of the voltage profile. The 30-segment interval model places the correct value for this state between $4.2 \cdot 10^{-14}$ and $4.6 \cdot 10^{-8}$.

The results of some analyses of the different approximations is shown in Table 6.9.

The steady-state analysis shows that the channel will almost always be closed, this is due to the almost-absorbing nature of state 1. It is notable that the 2-segment and the 5-segment models closely agree, while the 30-segment model allows a much higher probability of being open. This suggests



(a) Actual

(b) 2-point average approximation

(c) 5-point average approximation

(d) Interval approximation

Figure 6.2: Voltage profile across the modeled calcium channel

that either the precise value of the initial voltage spike may have a large effect on the model, since the subsequent downward slope is captured considerably better by the 5-segment model than by the 2-segment model.

The transient analysis of reaching open states from closed states in one cycle indicates that it is unlikely to reach an open state from state C_1. If the channel is in a state closer to an open state, switching between closed and open is much more likely.

Finally, the last two lines indicate that although a channel in an open state will almost certainly close within one cycle, there is a reasonable chance that at the end of the cycle the channel will be open again.

Proposition	2-segment	5-segment	30-segment interval
S[open]	$[0; 1.03 \cdot 10^{-22}]$	$[0; 1.04 \cdot 10^{-22}]$	$[0; 1.46 \cdot 10^{-11}]$
P[closed $U^{[0,1000]}$ open] from state C_1	$[0; 1.45 \cdot 10^{-23}$	$[0; 3.01 \cdot 10^{-23}]$	$[0; 1.29 \cdot 10^{-22}]$
P[closed $U^{[0,1000]}$ open] from state C_{15}	$0.93[73; 74]$	$0.92[93; 94]$	$0.9[08; 49]$
P[open $U^{[0,1000]}$ closed] from state O_2	$0.99992918435[37; 42]$	$0.999936127[29; 31]$	$[0.997; 1]$
P[True $U^{[1000,1000]}$ open] from state O_2	$0.2380451421[14; 55]$	$0.231121007[46; 53]$	$0.2[25; 40]$

Table 6.9: Results of analysis of the I_{K_s} potassium channel model

.

Chapter 7

Conclusions

The previous chapters have described the modification of the Markov Reward Model Checker to use interval arithmetic to obtain accurate error bounds on the obtained results.

The new program can answer probabilistic questions in Continuous Stochastic Logic or Probabilistic Computational Tree Logic for continuous-time and discrete-time Markov models.

For discrete-time systems, IMRMC can often compute the transient analysis faster and with better accuracy by calculating a matrix power using repeated squaring instead of repeated multiplication.

To analyze continuous-time models, the program can use two different algorithms: a discretization followed by a discrete-time analysis, or an analysis based on matrix exponentials. The exponential approach is generally faster and more accurate than the discretization method, but discretization can still be used to answer questions about reward models.

The choice of algorithm is largely left to the user. Some algorithms, such as the matrix power by repeated squaring and the continuous-time analysis by matrix exponential, are most suited for nonsparse models. Others are considerably more efficient for sparse models.

In addition, the program can analyze models in which the probabilities of the model vary over time in a stepwise fashion at fixed times. This allows the modeling of many systems that cannot otherwise be correctly modeled as a Markov chain.

Tests give confidence that the results computed by the modified program are correct, and the bounds are reasonably tight. Computation time has increased compared to the original program, but for most models not to the point of being unacceptable. Further, it is likely that the computation time can be significantly reduced without affecting accuracy.

7.1 Further Research

The original Markov Reward Model Checker could also analyse models which have impulse rewards (that is, a state-dependent reward is given each time the model transitions to that state). The modified program does not contain this functionality due to time constraints. It should be possible to modify the discretization method to incorporate impulse rewards.

The modified program also consumes much more time than the original program when analysing continuous-time models where rewards are involved. This is because the program then performs a discretization, followed by a path enumeration. Since the discretization doubles the number of states, and the number of paths grows quadratically with the number of states, this is no surprise. A better approach might allow these analyses to be performed within an acceptable time.

Because the choice of algorithm can affect accuracy and running time, it would be useful if IMRMC could automatically determine what class of model is being analyzed, and select the appropriate algorithm.

Several approaches can be used to optimize the interval methods used further. For example, many

propositions only require either the upper or the lower bound. For these propositions it is not necessary to compute the other bound. Furthermore, in many cases it is known that values are positive, leading to additional optimizations.

Although the interval arithmetic in IMRMC gives guaranteed bounds, bugs in the program cannot be entirely ruled out. These bugs may result in incorrect bounds. It may be useful to use algorithmic software verification to ensure that the program does not contain any errors.

Bibliography

[1] C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[2] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[3] L. Cloth, J.-P. Katoen, M. Khattri, and R. Pulungan. Model-checking Markov reward models with impulse rewards. In *Dependable Systems and Networks (DSN)*, Yokohama, Japan, 2005.

[4] J. Heijman et al. Dominant-negative control of cAMP-dependent IKs upregulation in human long-QT syndrome type 1novelty and significance. *Circulation Research*, 2013.

[5] M. Galassi et al. *GNU Scientific Library Reference Manual (3rd edition)*. Network Theory Ltd., 2009.

[6] A. Goldsztejn. On the exponentiation of interval matrices. Technical report, Laboratoire d'Informatique de Nantes Atlantique, 2009.

[7] E. R. Hansen. Bounding the solution of interval linear equations. *SIAM Journal on Numerical Analysis*, 29(5):1493–1503, 1992.

[8] J.-P. Katoen, M. Khattri, and I. S. Zapreev. A markov reward model checker. In *in QEST05, Proc. 2nd Intl. Conf. on the Quantitative Evaluation of Systems*, pages 243–244. IEEE CS Press, 2005.

[9] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[10] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on the Foundations of Computer Science*, pages 46–57, 1977.

[11] S. M. Ross. *Introduction to probability models*. Academic Press, 2007.

[12] K. Sen, M. Viswanathan, and G. Agha. VESTA: A statistical model-checker and analyzer for probabilistic systems. In *QEST05, Proc. 2nd Intl. Conf. on the Quantitative Evaluation of Systems*, pages 251–252. IEEE CS Press, 2005.

Appendix A

Parameters of systems biology testcase

The parameters for the model of the I_{Ks} potassium channel described in section 6.5 are listed below. In the following equations, V_m denotes the membrane voltage in mV, F is Faraday constant ($9.648534 \cdot 10^4$ $\frac{\mathrm{C}}{\mathrm{mol}}$), R is the universal gas constant ($8.314462 \frac{\mathrm{J}}{\mathrm{mol\ K}}$), and T is the temperature in Kelvin (taken to be 310 K).

$$\alpha = \frac{8.7313 \cdot 10^{-3}}{1+\exp\left(-\frac{V_m - 33.032}{1.1946} \cdot \frac{F}{R \cdot T}\right)}$$

$$\beta = \frac{1.1068 \cdot 10^{-2}}{1+\exp\left(\frac{V_m - 1.9519 \cdot 10^{-2}}{1.4447} \cdot \frac{F}{R \cdot T}\right)}$$

$$\gamma = \frac{1.5226 \cdot 10^{-1}}{1+\exp\left(-\frac{V_m + 5.3866 \cdot 10^{-2}}{5.3191 \cdot 10^{-1}} \cdot \frac{F}{R \cdot T}\right)}$$

$$\delta = 1.0372 \cdot 10^{-2} \cdot \exp\left(4.7872 \cdot 10^{-3} \cdot \frac{V_m \cdot F}{R \cdot T}\right)$$

$$\eta = 1.5447 \cdot 10^{-2} + \frac{5.453 \cdot 10^{-}2}{1+\exp\left(\frac{V_m + 103.18}{5.2369 \cdot 10^{-1}} \cdot \frac{F}{R \cdot T}\right)}$$

$$\theta = 1.7695 \cdot 10^{-3}$$

$$\omega = 5.6517 \cdot 10^{-4} \cdot \exp\left(-1.5643 \cdot \frac{V_m \cdot F}{R \cdot T}\right)$$

$$\psi = 2.8148 \cdot 10^{-5} \cdot \exp\left(1.9762 \cdot 10^{-4} \cdot \frac{V_m \cdot F}{R \cdot T}\right)$$