GPA – Tool for rapid analysis of very large scale PEPA models

Anton Stefanek Richard A. Hayden Jeremy T. Bradley
Department of Computing, Imperial College London

***Abstract*—We present a tool called *Grouped PEPA Analyser* (GPA) that allows fast analysis of large scale models described in the stochastic process algebra PEPA. GPA employs the techniques for approximations of transient moments in PEPA models with ordinary differential equations (ODEs), which allow analysis of systems with state spaces far beyond the limits of standard techniques. These moments provide useful information about the evolution of the model over time. Additionally, GPA implements a recently developed extension for moments of accumulated rewards, giving access to measures representing important factors in performance modeling such as energy consumption. GPA is also able to use these moments to calculate bounds on various passage time probabilities as well as completion times of the accumulated rewards. We describe the features of GPA in detail and briefly mention some design issues. We also present a technique that GPA implements for analysing more complex PEPA models with a higher degree of accuracy.**

I. Introduction

Quantitative analysis of stochastic systems by means of ordinary differential equations (ODEs) or fluid techniques aims to produce approximations of transient measures for models with state spaces beyond the limits of direct analysis of the underlying stochastic process, where otherwise stochastic simulation is the only alternative.

The recently developed technique [1, 2] shows how to derive a set of ODEs for approximations of certain transient measures of models described in the Markovian stochastic process algebra PEPA. The complexity of this technique doesn't depend on the number of instances of the same component in the model. This is crucial for analysing the class of so-called *massively parallel* systems, which consist of large numbers of identically behaved components cooperating in parallel, such as the servers within cloud computing clusters or devices in large wireless networks. Such systems suffer from the *state space explosion* problem, producing models with numbers of states far beyond the current limits of methods for directly analysing the underlying continuous time Markov chain (CTMC). The derived ODEs can be readily numerically solved, often in time that is multiple orders of magnitude smaller than what is needed for sufficiently many replications of the stochastic simulation of the system.

This approach directly benefits from the succinctness and compositionality of the stochastic process algebra PEPA. However, the structure of the resulting ODEs, especially those for higher moments of component counts [2] or moments of accumulated rewards [3], is much more complicated than the corresponding model description. We present a tool, called Grouped PEPA Analyser (GPA), that hides this complexity from the modeller and gives access to methods using the derived ODEs to produce useful measures of the system.

GPA for the first time implements the derivation of ODEs for higher moments of component counts, as described in [2], and accumulated rewards, as described in [3], in a stochastic process algebra. It also provides bounds on passage time probabilities, improving the techniques introduced in [4] by employing bounds based on higher moments from [5]. GPA provides means to estimate the accuracy of the resulting ODEs by producing so-called *switch point* plots, as described in [6], where an earlier version of the tool was introduced. Finally, to allow further experiments with the underlying theory, GPA provides efficient stochastic simulation based on the Gillespie algorithm.

After briefly mentioning the related work, we quickly introduce *Grouped PEPA*, a variant of the PEPA stochastic process algebra. We will give an overview of the techniques for deriving ODEs for moments of counts [2] and accumulated rewards [3] and demonstrate them on a simple client/server model. We also mention the results on deriving passage time probabilities from these measures [4] and error analysis using the switch points [6]. The next section introduces the GPA tool by demonstrating how it can produce the previously mentioned results and also highlights some interesting design issues. The final section describes an improved approximation GPA uses to produce ODEs for a more complex class of PEPA models giving rise to rates containing fractions.

II. Related tools

There are many tools that support analysis of very large state spaces in performance modelling. Two such popular tools which have good support for explicit state-space analysis are Möbius and PRISM.

The Möbius [7] framework has perhaps the widest user base with implementations of many formalisms, including stochastic process algebras (SPAs), stochastic automata networks (SANs) and generalised stochastic Petri nets (GSPNs). Möbius supports a distributed simulation environment and numerical solvers for models of up to tens of millions of states.

PRISM [8] is a probabilistic model checker which supports low level formalisms such as DTMCs, CTMCs and Markov Decision Processes (MDPs) with an analysis engine based on Binary Decision Diagrams (BDDs) and Multi-Terminal Binary Decision Diagrams (MTBDDs).

PRISM can analyse models of up to 10^{11} states, however this can depend heavily on the model being studied and on detailed considerations such as the exact variable ordering in the underlying MTBDD.

Performance tools that support differential-equation based analysis have been primarily designed around stochastic process algebras such as stochastic π-calculus and PEPA. For π-calculus SPiM [9, 10] has long been the standard tool for simulating stochastic π calculus models, but being a simulator it suffers from scalability issues for models with very large populations of components. A recent tool, JSPiM [11], allows for the *chemical ground form* subset of stochastic π-calculus to be analysed via differential equations. For the stochastic process algebra PEPA, the tools ipc [12, 13] and the Eclipse PEPA plug-in [14] implement the so-called *fluid translation* [1] to produce sets of differential equations for PEPA.

In the field of biological modelling, tools such as Dizzy [15] and SPiM have been used to capture first order approximations to system dynamics using a combination of stochastic simulation [16] and differential equation approximation. A recent tool from [17] generates ODEs approximating higher moments of models using the mass-action kinetics and described in the *Systems Biology mark-up language*. However for the reasons discussed, these techniques do not extend to computer and communication system modelling.

In [6] an earlier version of the GPA tool was used to analyse empirically the accuracy of the ODE approximations to the moments of component counts. In this paper we concentrate on GPA and present substantial extensions since [6]. These include implementation of the techniques for analysing moments of accumulated rewards, extensions of the bounds on passage and completion times using moments of orders higher than two and the improved approximation for models giving rise to fractional rates. We also look more closely at the implementation details of GPA.

III. Grouped PEPA

Grouped PEPA or GPEPA [2] is a simple syntactic extension of the stochastic process algebra PEPA defined to provide a more elegant treatment of the ODE moment approximation. Formally, the extension introduces a further level in the syntax of PEPA, the *Grouped PEPA models*, defined as:

$$G ::= G \underset{L}{\bowtie} G \;\mid \mathbf{Y}\{P \parallel \cdots \parallel P\}$$

This defines a GPEPA model to be either a PEPA cooperation between two GPEPA models (over the set of actions L) or alternatively a grouping of PEPA components P, in parallel with each other, labelled by the *group label* $\mathbf{Y}$. The Grouped PEPA model is nothing more than a standard PEPA model with a label to define the components involved in parallel grouping. These labels are used to define the level at which the ODE approximation to the system is made.

As an example, we can consider a simple client/server model. The clients communicate with the servers in two stages – they first request some data and then receive the data, after which they perform an independent action and return to the initial state where the cycle repeats. The servers, in addition to answering and serving the requests, can break and need to be reset. The following GPEPA description models this system:

$$\begin{aligned}
Client &\stackrel{def}{=} (req, r_{req}).Client_waiting \\
Client_waiting &\stackrel{def}{=} (data, r_{data}).Client_think \\
Client_think &\stackrel{def}{=} (think, r_{think}).Client \\
\\
Server &\stackrel{def}{=} (req, r_{req}).Server_get \\
&\qquad + (break, r_{break}).Server_broken \\
Server_get &\stackrel{def}{=} (data, r_{data}).Server \\
Server_broken &\stackrel{def}{=} (reset, r_{reset}).Server
\end{aligned}$$

$$\mathbf{Clients}\{Client[c]\} \underset{\{req,\,data\}}{\bowtie} \mathbf{Servers}\{Server[s]\}$$

The labels **Clients** and **Servers** allow unique identification of the components in the model. The system starts with c copies of the $Client$ components in the group **Clients**, which can evolve into $Client\_waiting$ and $Client\_think$ components respectively and with s copies of the $Server$ components in the group **Servers** that can evolve into $Server\_get$ and $Server\_broken$ components. In general, both the group label and the PEPA component are needed to uniquely identify a system component. However, since in the above model only client components can be in the group **Clients** and server components in the group **Servers**, we will use the shorthands C, C_w, C_t, S, S_g, S_b for the respective group/component pairs.

A. ODE Moment Analysis

Traditionally, the system states in PEPA models keep track of the state of each individual sequential component, which can lead to explosion of the state space resulting in the model being unamenable to standard analysis techniques other than the computationally expensive stochastic simulation. A way to tackle this in case of groups consisting of many identical components is by aggregating the state space by keeping track of *counts* of the individual components [1].

In the context of Grouped PEPA models, it is sufficient to represent each state of the underlying CTMC by a numerical vector $\mathbf{N}(t)$ consisting of counts $N_{\mathbf{G},P}(t)$ for each possible pair of a group label $\mathbf{G}$ and a component P. Using the shorthands above, such as $C(t)$ for $N_{\mathbf{Clients},Client}(t)$, we get vector coordinates $C(t), S(t)$, and so on.

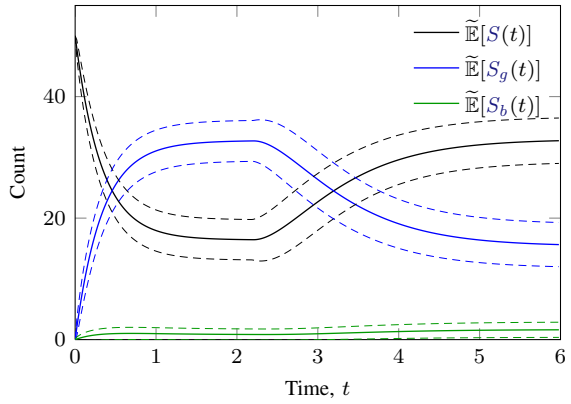
It has been shown in [2] how to derive approximations to the differential equations governing the expectations of these counts $\mathbb{E}[N_{G,P}(t)]$. For example, the method

from [2] gives the approximations $\widetilde{\mathbb{E}}[\cdot]$ to the exact means $\mathbb{E}[\cdot]$, such as

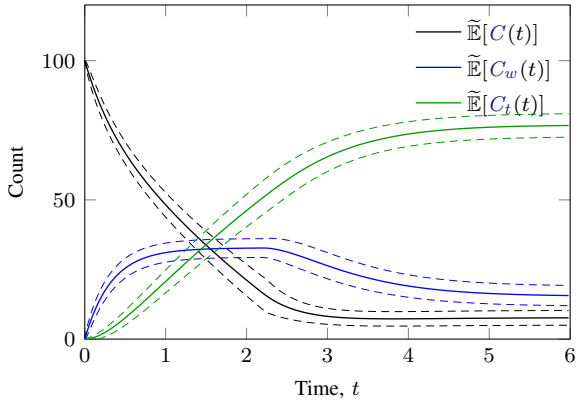
$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}\widetilde{\mathbb{E}}[C(t)] &= r_{think}\widetilde{\mathbb{E}}[C_t(t)] - r_{req}\min(\widetilde{\mathbb{E}}[C(t)], \widetilde{\mathbb{E}}[S(t)])\\ \frac{\mathrm{d}}{\mathrm{d}t}\widetilde{\mathbb{E}}[S(t)] &= r_{reset}\widetilde{\mathbb{E}}[S_b(t)] + r_{data}\min(\widetilde{\mathbb{E}}[C_w(t), S_g(t)])\\ &\quad - r_{break}\widetilde{\mathbb{E}}[S(t)] - r_{req}\min(\widetilde{\mathbb{E}}[C(t)], \widetilde{\mathbb{E}}[S(t)])\end{aligned} \tag{III.1}$$

The initial values of the approximations to the means are given by the counts in the initial grouped model, e.g. in the client/server example, $C(0) = c$, $S(0) = s$ and all other means are 0. The ODEs with the initial values can be numerically solved to give the respective approximations. In most cases, this process is computationally much less expensive than the corresponding stochastic simulation.

The method also derives ODEs for approximations of higher moments, such as $\widetilde{\mathbb{E}}[C(t)^2]$, $\widetilde{\mathbb{E}}[S_g(t)\,C_w(t)]$, and so on, that can be used to produce useful quantities such as variances and covariances. Figure 1 shows the numerical solution to the equations for means of all the components in the client/server example.



(a) servers



(b) clients

Figure 1: Numerical solution to the means of the client and server components in the client/server models. The dashed lines are one standard deviation above/below the respective means.

It also demonstrates how the higher moments can show variability of the model, by displaying the bounds by one standard deviation around the means. To calculate the approximation of standard deviations, for example those of the count of clients, $\sqrt{\widetilde{\mathrm{Var}}[C(t)]}$, the approximation for the variance is needed. Therefore an ODE for the second order moment $\widetilde{\mathbb{E}}[C(t)^2]$ has to be included in the system, since $\mathrm{Var}[C(t)] = \mathbb{E}[C(t)^2] - \mathbb{E}[C(t)]^2$.

B. Rewards

Accumulated quantities such as energy consumption or total cost are further critical factors determining the practical operation of massively parallel systems. Traditionally, these have been captured by using Markov Reward Models, where models have been augmented with random variables that accumulate a reward for being in a particular state. Similar to the case of component counts, analysing these quantities is sensitive to the size of the underlying state space.

The method in [3] extends the set of ODEs for component counts with further ODEs that capture means and higher moments of accumulated component counts. For example, it provides ODEs approximating the means of integrals of component counts, $\mathbb{E}[\int_0^t N_{\mathbf{G},P}(u)\mathrm{d}u]$. These ODEs can be derived simply by interchanging the order of integration and expectation, for example

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbb{E}\left[\int_0^t C(u)\mathrm{d}u\right] = \frac{\mathrm{d}}{\mathrm{d}t}\int_0^t \mathbb{E}\left[C(u)\right]\mathrm{d}u = \mathbb{E}[C(t)] \tag{III.2}$$

A theorem proved in [3] further derives approximations to ODEs governing higher moments of the accumulated rewards, such as $\mathbb{E}[(\int_0^t S(u)\mathrm{d}u)^2]$. These use auxiliary ODEs for combined moments such as $\widetilde{\mathbb{E}}[C(t)\int_0^t S(u)\mathrm{d}u]$.

Figure 2 demonstrates the accumulated rewards on the client/server example. It shows the accumulated quantity $\mathbb{E}[\int_0^t S_g(u)\mathrm{d}u]$ that could represent the energy consumption of the servers in arguably their most active state when serving clients with the requested data.

The technique also provides ODEs for moments of accumulations of *products* of component counts. For example the measure $\int_0^t C(u)S(u)\mathrm{d}u$ could represent the total accumulated income if each client is charged for access to each server at every time instant.

C. Passage Times

Answers to questions of the form "what is the probability of the system entering a specific target state within a given time t?" can provide practically useful information about parallel systems. It has been shown in [4] how to use the ODE approximations to moments of component counts to answer some of these questions. Often, the components are able to return to previous states from the target state. If we are interested in the time by which a component has entered the target state at least once, also know as the *first passage time*, we need to make sure that the set of states reachable from the target state is absorbing. For

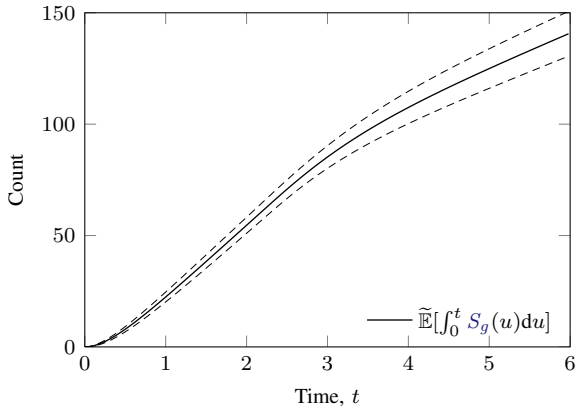


Figure 2: Numerical solution to the ODE representing the mean energy consumption of the most active server state. The dashed lines are one standard deviation from the mean.

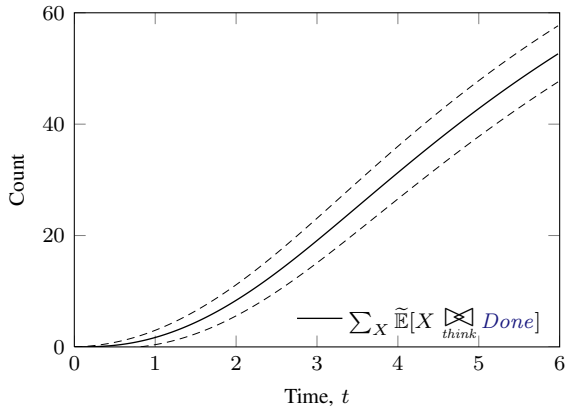


Figure 3: The mean number of clients performing their first $think$ action by the time t.

example, we can be interested in the time by which a $Client$ component has done its first $think$ action. This is the same as the component entering the $Client$ state for the first time after the start of the system.

In order to make all the subsequent states of the component distinguishable from the states where the component has yet to perform the $think$ action, we can attach a *probe* to the component. The probe will remember whether the $Client$ component performed the $think$ action – the $Done$ state, or not – the $NotDone$ state, and can be defined as

$$NotDone \stackrel{def}{=} (think, r_{think}).Done$$
$$Done \stackrel{def}{=} (think, r_{think}).NotDone$$

We can then replace the $Client$ component in the system by the synchronised component $Client \underset{think}{\bowtie} NotDone$, that has the same behaviour but also maintains the additional information. Summing the counts of all client components of the form $* \underset{think}{\bowtie} Done$, we can get the count of the client components that have performed the $think$ action. Figure 3 shows the ODE approximation to the mean of this sum and also the bounds based on the respective standard deviation.

As mentioned in [4], by dividing the mean count by the total client population c, we get an approximation to the cumulative density function (CDF) for the time it takes for an *individual* $Client$ component to perform its first $think$ action, a so-called *individual passage time* measurement.

Further, [4] also described how to get approximations to the probability that a certain fixed proportion of the clients performs their first $think$ action by the time t, a so-called *global passage time* measurement. In this case, [4] presents a method for getting *bounds* on the CDF of the passage time. These bounds use the Chebyshev inequality that uses second order moments of the component counts. To use higher moments for getting tighter bounds, we can employ the techniques from [5]. Figure 4 shows the bounds based on moments of order up to 6 on the CDF for 30 clients to perform their first $think$ action, compared to the exact CDF obtained using stochastic simulation.

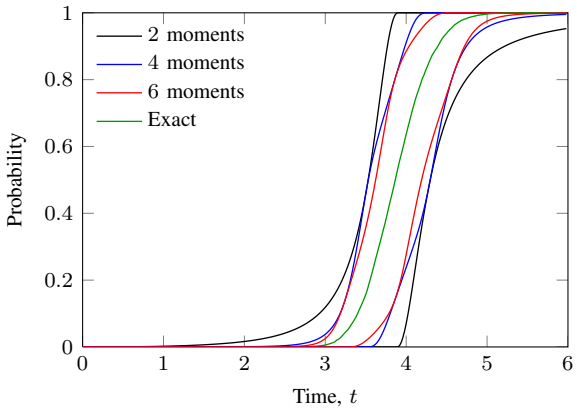


Figure 4: Bounds on the CDF of the global passage time of 30 clients performing the first $think$ action.

The techniques from [4] and [5] can be also used to provide bounds on the CDF of the time an accumulated reward takes to reach a certain target value, the so-called *completion time*. For example, Figure 5 shows the bounds on the CDF of the time it takes for the accumulated server energy consumption to reach the value 100.0.

D. Switch Points

The ODEs for component counts from [2] as well as the extension from [3] for accumulated rewards provide only approximations to the respective moments. This is because in the exact ODEs, terms containing the min function, such as

$$r_{req}\mathbb{E}[\min(C(t), S(t))] \qquad \text{(III.3)}$$

and also terms containing rational functions (fractions) can appear. In case of the former, the approximation

$$\mathbb{E}[\min(X, Y)] \approx \min(\mathbb{E}[X], \mathbb{E}[Y]) \qquad \text{(III.4)}$$

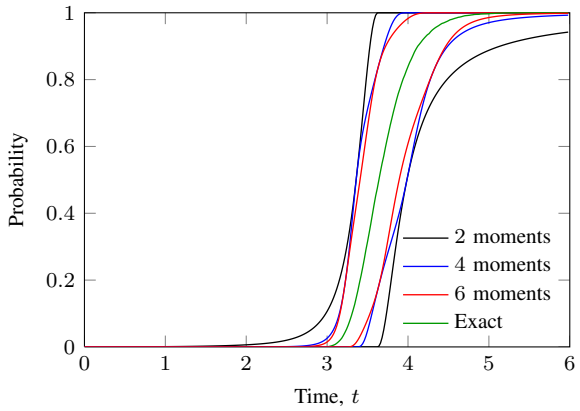
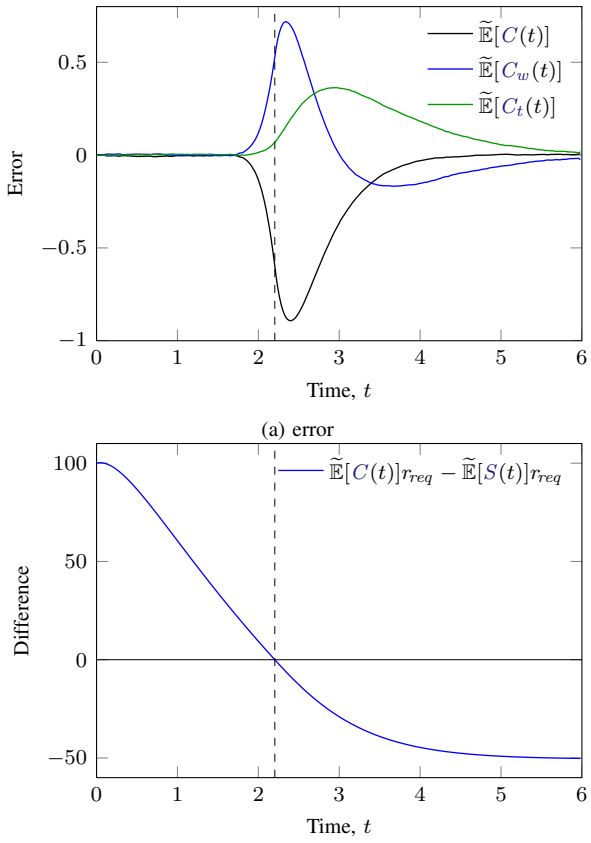


Figure 5: Bounds on the CDF of the completion time of the accumulated server energy consumption $\int_0^t S_g(u)\mathbf{d}u$ reaching the target value 100.0.

can be repeatedly used to obtain a closed system of ODEs. It can be shown that if in a GPEPA model each synchronised action can be performed only by one type of component in each cooperating group, all the rates of the synchronised actions will only contain $\min$ functions and no fractions.

For such models, called *split-free*, it has been shown in [2] how the approximations relate to the exact values. More specifically, implied by a well known theorem of Kurtz [18], the exact means converge to the ODE solutions as the initial populations of the components are scaled up. For example, in the client/server model, if we start with $c \times n$ copies of the *Client* component and $s \times n$ copies of the *Server* component, the exact means get closer to the ODE solutions as we increase n. However, there isn't much formal understanding of the nature of this convergence. To bring more practical insight into the error of the ODE approximations, [6] investigates the error with respect to so-called *switch points*. These occur at times when the arguments to the $\min$ functions in rates of synchronisation, such as those in (III.1), are equal, representing points of time where the system switches between two different modes of behaviour. For example the rate $r_{req} \min(C(t), S(t))$ of the synchronised *req* action is equal to $r_{req} C(t)$ when the speed of cooperation is restricted by the number of *Client* components that is smaller than the number of *Server* components. At the time when the arguments are equal, the system can keep switching between the two modes and so the averaging approximation (III.4) is the least accurate. Plotting the distance between approximations to the expectations of the two arguments, also called the *switch point distance*, can serve as a rough indicator warning the modeler where inaccuracies can occur. Figure 6 shows the absolute error of the numerical solution to the ODEs for the means in the client/server model with relation to the distance between the two arguments in the rate (III.3).

It can be seen that the error is the highest around the point



(a) error

(b) switch point distance

Figure 6: Error in the ODEs approximating means in relation to the switch point distance.

where the switch point distance is zero.

Models that are not split-free, also called *splitting* due to the fact that the behaviour on a synchronised action can be "split" between the multiple components on one side of the cooperation, give rise to rates containing rational functions. It then becomes more complicated to choose approximations that give sufficient accuracy and imply convergence properties similar to the split-free case. Section V describes the approximations GPA is using and empirically investigates the error.

IV. GPA: The GPEPA Analyser

In this section, we introduce the *Grouped PEPA Analyser* (GPA), a tool for analysing large scale models specified in the Grouped PEPA stochastic process algebra. GPA implements the above mentioned techniques for generating ODEs for moments of component counts and accumulated rewards and for using these to produce bounds on the passage and completion time distributions. GPA supports practical applications of the techniques by providing convenient syntax that is very close to the formal definition of GPEPA. Additionally, GPA can be used for experiments giving further insight into the underlying theory, by implementing efficient stochastic simulation and allowing exports to other frameworks such as MATLAB.

A. Overview

GPA is a portable command line tool implemented in the Java 1.6 framework. GPA runs on an input file describing the GPEPA *model* and the used components as well as the different *analyses*, such as the ODE approximation or stochastic simulation, that are to be executed on this model. Each analysis calculates the resulting data set – values of the various moments at different points in time. The data sets can be further manipulated by *commands*, such as plots of specific arithmetic expressions on the moments or plots of bounds on various passage time CDFs. Output from the commands can be either viewed graphically or exported in a data file with an accompanying GNUplot script.

In the following we give details on the component and model descriptions and the supported analyses and commands. We illustrate all the features on the client/server example and show how each of Figures 1 – 6 was produced.

B. Models

Each GPA input file contains a definition of a single grouped model, also known as the *system equation*. The grouped model is a parallel composition of different components, which need to be defined. Definitions of these components require numerical constants for the action rate definitions, such as r_{req} in $(req, r_{req}).Server\_get$.

The syntax of GPA input files first requires definitions of the numerical constants, such as `rreq = 2.0;`. For example, the definitions of constants used in the above examples can be found in Figure 9 in the Appendix. After a list of such definitions, a list of definitions of the needed components follows. These are of the form

```
Client = (req,rreq).Client_waiting;
```

The grammar GPA uses is very close to the language of GPEPA: The prefix operator $(a, r).P$ is written as `(a,r).P`, where in place of `r` there can be any arithmetic expression using real values and the previously defined numerical constants. The choice $P + Q$ is written as `P+Q`, the parallel cooperation $\underset{a,b,c}{\bowtie}$ as `<a,b,c>`. The string `stop` represents the component not capable of any action.

After the component definitions, a single grouped model is given. A labelled group $\mathbf{G}\{\cdots\}$ is written as `G{...}`. Multiple copies of a component inside a group $C[n]$ are written as `C[n]`, where in the place of `n` there can be any arithmetic expression using real values and the previously defined numerical constants. Multiple components in parallel within a group $P|Q|R$ are written as `P|Q|R`.

The full syntax of GPA can be found in Figure 10 in the appendix.

The similarity of the GPA syntax and the formal GPEPA definition can be easily seen from the GPA description of the client/server example defined in GPEPA in Section III:

```
Client         = (req,rreq).Client_waiting;
Client_waiting = (data,rdata).Client_think;
Client_think   = (think,rthink).Client;

Server         = (req,rreq).Server_get
               +(break,rbreak).Server_broken;
Server_get     = (data,rdata).Server
               +(break,rbreak).Server_broken;
Server_broken  = (reset,rreset).Server;

Clients{Client[c]}<req,data>Servers{Server[s]}
```

C. ODE Analysis and Comparison with Simulation

After the GPEPA model definition, a list of analyses on the model follows. Each analysis accepts a list of parameters in brackets `( )` and then a list of *commands* in braces { }.

The analysis `odes` performs the ODE approximation of the moments of component counts and accumulated rewards.

```
odes(stopTime=5.0, stepSize=0.01,
                   density=10){...}
```

The parameter `stopTime` specifies the time until which the generated ODEs are numerically solved, the parameter `stepSize` gives the length of time between two successive data points and the parameter `density` is the number of implicit data points between each two data points that will be used by the built-in Runge-Kutta numerical ODE solver. The commands for the analysis determine which moments have to be included in the generated set of ODEs in order to get a closed system that can be numerically solved.

The analysis `simulation` performs stochastic simulation of the CTMC underlying the previously specified GPEPA model.

```
simulation(stopTime=5.0, stepSize=0.01,
                   replications=1000){...}
```

The `stopTime` parameter specifies when the simulation stops. The `stepSize` parameter gives the distance between two successive data points and the parameter `replications` the number of replications over which averages are taken. GPA implements the Gillespie algorithm for simulating the CTMC. It records all the moments that are needed by the enclosed commands.

The `comparison` analysis is useful for comparing the results from ODE approximation with the "exact" values from the stochastic simulation.

```
comparison(odes(...){...},
           simulation(...){...}){...}
```

It takes parameters `odes` analysis and a `simulation` analysis that both have the same `stopTime` and `stepSize`. The resulting data points used are the difference between the results from the two analyses.

D. Commands and Functionality

GPA can plot the results from different analyses or output raw data by means of optional file redirection.

The `plot` command provides direct plots of arithmetic expressions involving higher and joint moments of component counts and accumulated rewards. Each group component pair is identified by the syntax `Group:Component`. The expectation operator `E[]` is used to plot the moments. In case of moments of component counts, it accepts products of powers of group:component pairs, such as `E[G1:C1^2 G2:C2]`. In case of accumulated rewards, the product of component counts to be accumulated over time is placed inside the `acc()` operator, for example `E[acc(G1:C1 G2:C2^2)]`. GPA allows general combined moments that are expectations of products of component counts and accumulated products of counts, such as `E[G1:C1^3 acc(G2:C2 G3:C3)^2]`. Note that the time parameter is omitted, so for example `E[G:P]` stands for $\mathbb{E}[N_{G,P}(t)]$.

Additionally, GPA allows convenient shorthands. For example, the variance of a combined product can be plotted by `Var[*]`, which stands for `E[*^2]-E[*]^2`. The following are examples of the `plot` command in the client/server model:

```
plot(E[Clients:Client],E[acc(Servers:Server)]);
plot(E[Clients:Client Servers:Server_get^2]);
plot(E[Clients:Client]^2.0+Var[Servers:S]/s);
```

GPA also allows plots of the passage and completion times from Section III-C through the command `bounds`. The first argument is an expression which is a linear combination of products of component counts or accumulated rewards. The second argument is the target value the expression has to reach. The following arguments are orders of moments from which bounds to the passage/completion time probabilities are computed, using the method from [5]. For each order, the command plots against time both the lower and the upper bound on the probability of the expression reaching the target value within the time. Additionally, if the enclosing analysis is a simulation, the exact probabilities are sampled from the replications and included in the plot.

For example, to get the completion time probabilities of the accumulated count $\int_0^t \mathit{Client}(u)\mathrm{d}u$ reaching the value 10.0, using moments of order up to 2 and 4 respectively, we can run the following command within the braces of an analysis:

```
bounds(acc(Clients:Client),10.0,2,4)
```

To estimate the accuracy of the ODE approximation without running the corresponding simulation, GPA provides the command plotSwitchpoints. This command plots the switch point distance for all the min functions that are present in the ODEs generated by the enclosing `odes` analysis. It takes an integer argument that gives the maximum order of the moments within the arguments to the min functions.

E. Examples

We show how each of the plots from the previous section was generated with GPA. The plots are produced by commands within an `odes` analysis with the `stopTime` set to 6.0, the `stepSize` set to 0.005 and `density` 10. The plots where an exact probability is shown (i.e. Figures 4, 5) or a comparison with the exact means is used (i.e. Figure 6), the respective commands are placed within a `simulation` analysis with the above values of `stopTime` and `stepSize` and `replications` equal to 10^5.

Figure 1 showing transient means and standard deviations of the client and server populations is plot by the following command (only the expressions for the **`Clients`**`:Client` component are shown, the others are analogous):

```
plot(E[Clients:Client],
 E[Clients:Client]+Var[Clients:Client]^0.5,
 E[Clients:Client]-Var[Clients:Client]^0.5,
...)
```

The command to plot Figure 2 is analogous to the above, but with **Clients**`:Client` replaced by `acc(`**Servers**`:Server_get)`.

Similarly, to plot Figure 3, **Clients**`:Client` is replaced by **Clients**`:_<think>Done`, which stands for

```
Clients:Client<think>Done+
    Clients:Client_waiting<think>Done+
        Clients:Client_think<think>Done
```

The bounds in Figure 4 and Figure 5 are plot by:

```
bounds(Clients:_<think>Done,30.0,2,4,6)
bounds(acc(Servers:Server_get),30.0,2,4,6)
```

The exact probabilities are output when the `bounds` command is inside the above simulation analysis.

The transient error of the approximation in Figure 6 is plot by the command:

```
plot(E[Clients:Client],
     E[Clients:Client_waiting],
     E[Clients:Client_think]);
```

from within a `comparison` analysis comparing the above ODE analysis and simulation.

The switch point distance plot is plot by the command:

```
plotSwitchpoints(1);
```

from within the above ODE analysis.

F. Design Details

GPA follows a modular design that is suitable for further extensions. Additionally, the individual modules follow a layered dependency structure for easy reuse.

The basic high level components of the architecture of GPA are shown in Figure 7.

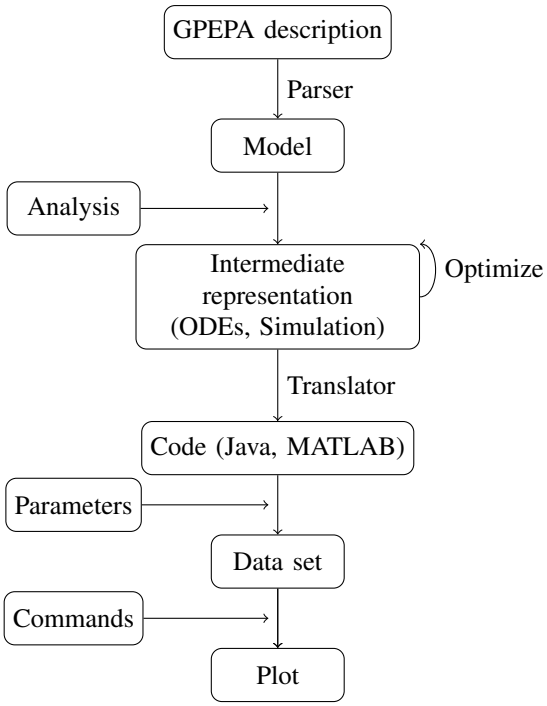


Figure 7: High level components of the GPA architecture, shown in the context of executing GPA on an input file.

GPA first parses the input description of the GPEPA model. Depending on the current analysis, GPA generates an abstract intermediate representation of the analysis applied to the model.

In case of the ODE analysis, this represents the set of differential equations. On this representation, GPA performs various optimisations. The right hand sides of the ODEs generated from the above mentioned techniques can be very complex, especially for ODEs for moments of high orders, possibly consisting of 10^6 and more multiplications and additions. GPA tries to identify repeated expressions and factors them out into variables. The optimised representation is then instantiated into code of a specific target language. By default, GPA generates Java code which gets compiled into a Java class suitable for efficient numerical computation, where all the high level data structures are flattened into arrays of floating point numbers. At this stage the parameters of the numerical constants used in the rates and initial populations get instantiated. This is a useful feature if the same model is analysed with different values of the constants – the expensive generation of ODEs is performed only once. After the code is used by the solver, in case of Java by the built-in implementation of the fourth order Runge-Kutta method, GPA creates the resulting data set that can be further manipulated by the commands.

The architecture of GPA allows convenient implementation of translators of the intermediate representation into various other languages. As an example, GPA provides a translator into MATLAB, which generates the corresponding ODE function and also translates the plot commands accordingly. This allows the modeler to use a wider range of numerical ODE solvers.

Similarly, the simulation analysis creates an intermediate representation of a function updating the rates corresponding to different actions in the model that is further optimised and translated into a specific language and used by an implementation of the Gillespie algorithm.

For testing purposes and for experiments with the underlying theory, GPA implements a simple random model generator. This randomly generates the individual components and the rates and places them into a generated grouped model.

For further implementation details including the list of used libraries, we refer the reader to the freely available source code in [19].

V. Rational Rates

In this section we look at the approximations GPA uses in the presence of rational rates. These arise when multiple components within a group are able to perform the same synchronised action. An obvious example in the context of our client server model is when we include an additional class of clients that are able to perform the synchronised req and $data$ actions, possibly at different rates. That is when we define components

$$\begin{aligned} Client' &\stackrel{def}{=} (req, r_{req'}).Client'\_waiting \\ Client'\_waiting &\stackrel{def}{=} (data, r_{data'}).Client'\_think \\ Client'\_think &\stackrel{def}{=} (think, r_{think'}).Client' \end{aligned}$$

and change the **Clients** group to be

$$\mathbf{Clients}\{Client[c] \parallel Client'[c']\}$$

The rate of performing a synchronised req action by the original class of clients at time t then is

$$\frac{C(t)r_{req}}{C(t)r_{req} + C'(t)r_{req'}} \times \min(S(t)r_{req}, C(t)r_{req} + C'(t)r_{req'}) \quad \text{(V.1)}$$

The expectation of the above rate is present in the ODEs for the means of the component counts. An obvious approximation

$$\frac{\mathbb{E}[C(t)]r_{req}}{\mathbb{E}[C(t)]r_{req} + \mathbb{E}[C'(t)]r_{req'}} \times \min(\mathbb{E}[S(t)]r_{req}, \mathbb{E}[C(t)]r_{req} + \mathbb{E}[C'(t)]r_{req'}) \quad \text{(V.2)}$$

gives results with high degree of accuracy (error in the region of 1% even at low scales). We also empirically observe similar accuracy for other models, including a large number of randomly generated ones.

In case of ODEs for second order moments, the rates such as (V.1) are multiplied by other component counts. For example, the ODE for $\mathbb{E}[C_w(t)^2]$ contains the rate from (V.1) multiplied by $C_w(t)$. The approximation originally suggested in [2]

$$\frac{\mathbb{E}[C(t)]r_{req}}{\mathbb{E}[C(t)]r_{req} + \mathbb{E}[C'(t)]r_{req'}} \times \min(\ \mathbb{E}[S(t)\,C_w(t)]r_{req},\ \mathbb{E}[C(t)\,C_w(t)]r_{req} + \mathbb{E}[C'(t)\,C_w(t)]r_{req'}) \tag{V.3}$$

is subject to a high error that propagates into an error of more than 100% in the variance of $C_w(t)$. Moreover, this error doesn't seem to get relatively smaller with increasing scale of the model.

Therefore GPA uses a better approximation. Intuitively, there is a correlation between the fraction in (V.1) and the multiplicative term $C_w(t)$. This correlation is completely ignored by the approximation in (V.3). To include some information about this correlation, we can multiply both the numerator and the denominator of the fraction by the term $C_w(t)$. This gives an improved approximation

$$\frac{\mathbb{E}[C(t)\,C_w(t)]r_{req}}{\mathbb{E}[C(t)\,C_w(t)]r_{req} + \mathbb{E}[C'(t)\,C_w(t)]r_{req'}} \times \min(\ \mathbb{E}[S(t)\,C_w(t)]r_{req},\ \mathbb{E}[C(t)\,C_w(t)]r_{req} + \mathbb{E}[C'(t)\,C_w(t)]r_{req'}) \tag{V.4}$$

which produces results with a reasonable degree of accuracy, below 10% even at small scales. Moreover, similar accuracy can be observed for other models and our experiments also suggest that this approximation becomes exact as the component populations are scaled up, in the same sense as in the case of split-free models.

Figure 8 demonstrates the difference between the two approximations in case of the variance of C_w components.

It shows the quantity $\mathrm{dVar}[C_w(t)]/\mathrm{d}t$ and both the original and improved approximations. It can be seen that the improved approximation matches the exact value quite accurately. The original approximation starts to differ in the region where the ignored correlation between C_w components and the fractions present in the ODE become significant. This difference becomes even more prominent in the solution to the ODE, where the error accumulates, rendering the approximation unusable.

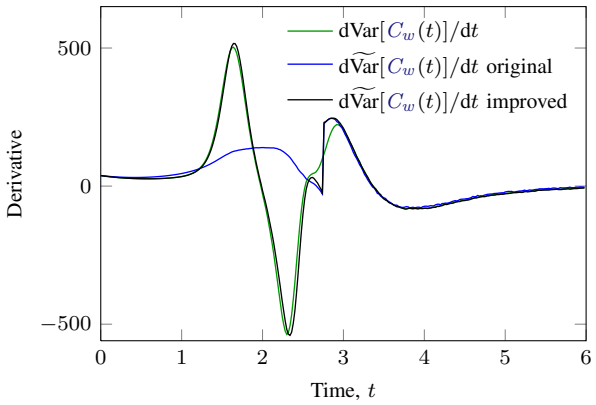


Figure 8: Comparison of the different rational function approximations.

VI. Conclusion and Further Work

We introduced a new tool called GPA that provides fast analysis of very large scale PEPA models. GPA implements, for the first time for a stochastic process algebra, the existing techniques for generating ODE approximations to moments of component counts and ODEs approximating moments of accumulated rewards, a useful feature for modeling real systems. For models with large populations of identical components, these ODE approximations are the only alternative to the computationally expensive stochastic simulation. GPA can use the ODE approximations to moments to generate bounds on the passage time and completion time probabilities.

We showed how GPA can be used to estimate the accuracy of the approximations by using the switch point distance plots.

GPA also provides comparisons with stochastic simulation, allowing further experiments with the underlying theory. This feature was useful in choosing a suitable approximation for more complicated models where the associated ODEs contain fractions. This approximation was presented together with observations that suggest a high degree of accuracy where the previously defined technique failed to give usable results. We plan to investigate this formally and possibly prove that this approximation gives convergence properties similar to the known case of ODEs without fractions.

The briefly presented architecture of GPA is suitable for further extensions and for reuse in tools for different formalisms.

In future, we plan to add support for passive rates, using the techniques from [20]. We also plan to extend the input language to allow more sophisticated performance specifications, for example using stochastic probes [21] or performance trees [22].

For testing purposes, GPA implements a simple random model generator. We plan to implement a more sophisticated generator that could be used as a benchmark suite for comparing efficiency and accuracy of different approximation techniques.

References

[1] J. Hillston, "Fluid flow approximation of PEPA models," in *Second International Conference on the Quantitative Evaluation of Systems (QEST'05)*, pp. 33–42, IEEE, 2005.

[2] R. A. Hayden and J. T. Bradley, "A fluid analysis framework for a Markovian process algebra," *Theoretical Computer Science*, vol. 411, pp. 2260–2297, May 2010.

[3] A. Stefanek, R. A. Hayden, and J. T. Bradley, "Rapid analysis of Rewards and Completion times in massively parallel Markov models." in preparation, 2010.

[4] R. A. Hayden and J. T. Bradley, "Fluid passage-time calculation in large Markov models," *submitted to Performance Evaluation*, 2009.

[5] A. Tari, M. Telek, and P. Buchholz, *A Unified Approach to the Moments Based Distribution Estimation Unbounded Support*, vol. 3670 of *Lecture Notes in Computer Science*, pp. 79–93. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[6] A. Stefanek, R. A. Hayden, and J. T. Bradley, "A new tool for the performance analysis of massively parallel computer systems," in *Eighth Workshop on Quantitative Aspects of Programming Languages (QAPL 2010), March 27-28, 2010, Paphos, Cyprus*, Electronic Proceedings in Theoretical Computer Science, March 2010.

[7] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster, "The Möbius Framework and its Implementation," *IEEE Transactions on Software Engineering*, vol. 28, pp. 956–969, October 2002.

[8] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: Probabilistic Symbolic Model Checker," in *TOOLS'02, Proceedings of the 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation* (A. J. Field et al., ed.), vol. 2324 of *Lecture Notes in Computer Science*, (London), pp. 200–204, Springer-Verlag, 2002.

[9] A. Phillips and L. Cardelli, "A Correct Abstract Machine for the Stochastic Pi-calculus," in *BIOCONCUR'04, Concurrent Models in Molecular Biology*, August 2004.

[10] A. Phillips and L. Cardelli, "Efficient, correct simulation of biological processes in stochastic Pi-calculus," in *CMSB'07, Proceedings of Computational Methods in Systems Biology*, vol. 4695 of *Lecture Notes in Computer Science*, pp. 184–199, Springer-Verlag, September 2007.

[11] A. Stefanek, "Continuous and spatial extension of stochastic Pi-calculus," Master's thesis, Department of Computing, Imperial College London, July 2009.

[12] A. Clark, "The ipclib PEPA Library," in *QEST'07, 4th International Conference on the Quantitative Evaluation of Systems*, pp. 55–56, IEEE, September 2007.

[13] J. T. Bradley, N. J. Dingle, S. T. Gilmore, and W. J. Knottenbelt, "Derivation of Passage-time Densities in PEPA Models using ipc: the Imperial PEPA Compiler," in *MASCOTS'03, Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, pp. 344–351, IEEE, October 2003.

[14] M. Tribastone, "The PEPA Plug-in Project," in *QEST'07, Proceedings of the 4th Int. Conference on the Quantitative Evaluation of Systems*, pp. 53–54, IEEE Computer Society, September 2007.

[15] S. Ramsey, D. Orrell, and H. Bolouri, "Dizzy: Stochastic Simulation of Large-scale Genetic Regulatory Networks," *Journal of Bioinformatics and Computational Biology*, vol. 3, no. 2, pp. 415–436, 2005.

[16] D. T. Gillespie, "Exact stochastic simulation of coupled chemical reactions," *Journal of Physical Chemistry*, vol. 81, no. 25, pp. 2340–2361, 1977.

[17] C. S. Gillespie, "Moment-closure approximations for mass-action models," *IET Systems Biology*, vol. 3, no. 1, pp. 52–58, 2009.

[18] T. Kurtz, "Solutions of ordinary differential equations as limits of pure jump Markov processes," *Applied Probability*, vol. 7, pp. 49–58, April 1970.

[19] "Grouped PEPA Analyser http://www.doc.ic.ac.uk/~as1005/GPA."

[20] R. A. Hayden and J. T. Bradley, "Evaluating fluid semantics for passive stochastic process algebra cooperation," *Performance Evaluation*, 2010.

[21] A. Clark and S. T. Gilmore, "State-Aware Performance Analysis with eXtended Stochastic Probes," *Lecture Notes In Computer Science; Vol. 5261*, 2008.

[22] T. Suto, J. T. Bradley, and W. J. Knottenbelt, "Performance Trees: A New Approach to Quantitative Performance Specification," in *14th IEEE International Symposium on Modeling, Analysis, and Simulation*, pp. 303–313, IEEE, 2006.

Appendix

```
rreq = 2.0;  rbreak = 0.1; rthink = 0.2;
rdata = 1.0; rreset = 2.0;
c = 100.0;   s = 50.0;
```

Figure 9: Used constants in the client/server example.

Models

$$
\begin{aligned}
\mathit{System} &:= \mathit{ParameterDefinition}^{*}\ \ \mathit{ComponentDefinition}^{*}\ \ \mathit{ModelDefinition}\ \ \mathit{Analysis}^{*} \\
\\
\mathit{ParameterDefinition} &:= \mathit{parameterId}\ =\ \mathit{realnumber};\\
\\
\mathit{ComponentDefinition} &:= \mathit{componentID}\ =\ \mathit{Component} \\
\mathit{Component} &:= \mathit{Component}{<}\mathit{ActionList}^{?}{>}\mathit{Component} \\
&\quad \mid \mathit{Summation} \mid \mathit{componentId} \mid (\mathit{Component}) \\
\mathit{Summation} &:= \mathit{Prefix}(+\mathit{Prefix})^{*} \\
\mathit{Prefix} &:= (\mathit{actionId},\ \mathit{parameterId}\ ).((\mathit{Summation}) \mid \texttt{stop} \mid \mathit{componentId}\ |);\\
\\
\mathit{ModelDefinition} &:= (\mathit{ModelDefinition}{<}\mathit{ActionList}^{?}{>}\mathit{ModelDefinition}) \\
&\quad \mid \mathit{groupLabel}\{\mathit{ComponentsParallel}\} \\
\mathit{ComponentsParallel} &:= \mathit{Component}(\ |\mathit{Component})^{*} \\
\mathit{Component} &:= \mathit{componentId}\ ([\mathit{parameterId}])^{?} \\
\mathit{ActionList} &:= \mathit{actionId}\ (,\ \mathit{actionId})^{*}
\end{aligned}
$$

Analyses

$$
\begin{aligned}
\mathit{Analysis} &:= \mathit{ODEs} \mid \mathit{Simulation} \mid \mathit{Comparison} \\
\mathit{ODEs} &:= \texttt{odes}(\texttt{stopTime}\ {=}\mathit{realnumber}, \texttt{stepSize}\ {=}\mathit{realnumber}, \texttt{density}\ {=}\mathit{integer}) \\
&\quad \{\mathit{Command}^{*}\} \\
\mathit{Simulation} &:= \texttt{simulation}(\texttt{stopTime}\ {=}\mathit{realnumber}, \texttt{stepSize}\ {=}\mathit{realnumber}, \texttt{replications}\ {=}\mathit{integer}) \\
&\quad \{\mathit{Command}^{*}\} \\
\mathit{Comparison} &:= \texttt{comparsion}(\mathit{ODEs},\mathit{Simulation})\{\mathit{Command}^{*}\}
\end{aligned}
$$

Commands

$$
\begin{aligned}
\mathit{Command} &:= \mathit{CommandNoFile}(\text{->}\text{"}\mathit{filename}\text{"})^{?}; \\
\mathit{CommandNoFile} &:= \texttt{plot}(\mathit{MomentExpressions}) \mid \texttt{plotSwitchpoints}(\mathit{integer}) \\
\mathit{MomentExpressions} &:= \mathit{MomentExpression}(,\ \mathit{MomentExpression})^{*} \\
\mathit{MomentExpression} &:= \mathit{MomentExpression}(+ \mid - \mid * \mid / \mid \hat{}\,)\mathit{MomentExpression} \\
&\quad \mid \texttt{E}[\mathit{Moment}(+\mathit{Moment})^{*}] \mid \texttt{Var}[\mathit{GCPair}(+\mathit{GCPair})^{*}] \\
&\quad \mid (\texttt{Standardised})^{?}\texttt{Central}[\mathit{GCPair},\mathit{integer}] \\
&\quad \mid \mathit{realnumber} \mid \mathit{parameterId} \mid (\mathit{MomentExpression}) \\
\mathit{Moment} &:= \mathit{CountMoment}^{?}(\mathit{AccumulatedMoment})(\hat{}\,\mathit{integer})^{?})^{*} \\
\mathit{AccumulatedMoment} &:= \texttt{acc}(\mathit{CountMoment}) \\
\mathit{CountMoment} &:= (\mathit{GCPair}(\hat{}\,\mathit{integer})^{?})^{+} \\
\mathit{GCPair} &:= \mathit{groupLabel}{:}\mathit{Component}
\end{aligned}
$$

Figure 10: GPA syntax definition