SPA for quantitative analysis: Lecture 3 — Case Studies and Tools

Jane Hillston

LFCS, School of Informatics
The University of Edinburgh
Scotland

5th March 2013

Outline

Outline

Dynamic behaviour

- The behaviour of a model is dictated by the semantic rules governing the combinators of the language.
- The possible evolutions of a model are captured by applying these rules exhaustively, generating a labelled transition system.
- This can be viewed as a graph in which each node is a state of the model (comprised of the local states of each of the components) and the arcs represent the actions which can cause the move from one state to another.
- The language is also equipped with observational equivalence which can be used to compare models.

Dynamic behaviour

- The behaviour of a model is dictated by the semantic rules governing the combinators of the language.
- The possible evolutions of a model are captured by applying these rules exhaustively, generating a labelled transition system.
- This can be viewed as a graph in which each node is a state of the model (comprised of the local states of each of the components) and the arcs represent the actions which can cause the move from one state to another.
- The language is also equipped with observational equivalence which can be used to compare models.

Dynamic behaviour

- The behaviour of a model is dictated by the semantic rules governing the combinators of the language.
- The possible evolutions of a model are captured by applying these rules exhaustively, generating a labelled transition system.
- This can be viewed as a graph in which each node is a state of the model (comprised of the local states of each of the components) and the arcs represent the actions which can cause the move from one state to another.
- The language is also equipped with observational equivalence which can be used to compare models.

Dynamic behaviour

- The behaviour of a model is dictated by the semantic rules governing the combinators of the language.
- The possible evolutions of a model are captured by applying these rules exhaustively, generating a labelled transition system.
- This can be viewed as a graph in which each node is a state of the model (comprised of the local states of each of the components) and the arcs represent the actions which can cause the move from one state to another.
- The language is also equipped with observational equivalence which can be used to compare models.

PEPA Eclipse Plug-In input

$$P_1 \stackrel{def}{=} (start, r_1).P_2 \qquad P_2 \stackrel{def}{=} (run, r_2).P_3 \qquad P_3 \stackrel{def}{=} (stop, r_3).P_1$$

$$P_1 \parallel P_1$$

PEPA Eclipse Plug-In input

$$P_1 \stackrel{def}{=} (start, r_1).P_2 \qquad P_2 \stackrel{def}{=} (run, r_2).P_3 \qquad P_3 \stackrel{def}{=} (stop, r_3).P_1$$

$$P_1 \parallel P_1$$

State space

1	$P_1 \parallel P_1$
2	$P_1 \parallel P_2$
3	$P_2 \parallel P_1$
4	$P_1 \parallel P_3$
5	$P_2 \parallel P_2$
6	$P_3 \parallel P_1$
7	$P_3 \parallel P_2$
8	$P_3 \parallel P_2$
9	$P_3 \parallel P_3$

PEPA Eclipse Plug-In input

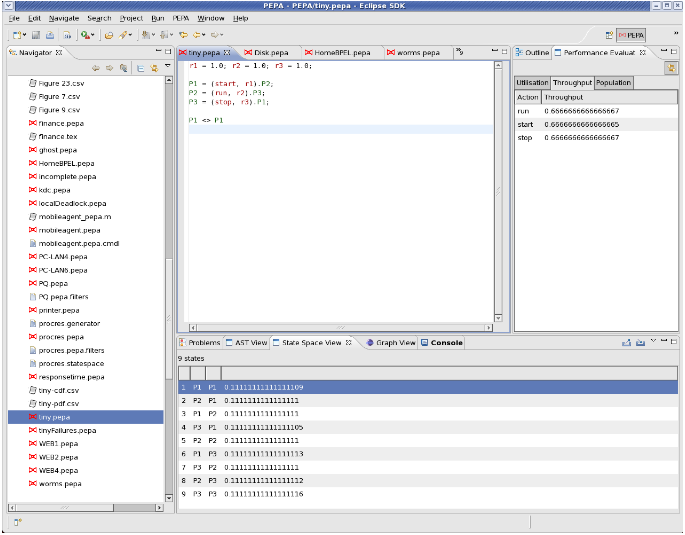
$$P_1 \stackrel{def}{=} (start, r_1).P_2 \qquad P_2 \stackrel{def}{=} (run, r_2).P_3 \qquad P_3 \stackrel{def}{=} (stop, r_3).P_1$$

$$P_1 \parallel P_1$$

CTMC representation computed by the plug-in

$$\begin{pmatrix}
-2r_1 & r_1 & r_1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -r_1 - r_2 & 0 & r_2 & r_1 & 0 & 0 & 0 & 0 \\
0 & 0 & -r_1 - r_2 & 0 & r_1 & r_2 & 0 & 0 & 0 \\
r_3 & 0 & 0 & -r_1 - r_3 & 0 & 0 & 0 & r_1 & 0 \\
0 & 0 & 0 & 0 & -2r_2 & 0 & r_2 & r_2 & 0 \\
r_3 & 0 & 0 & 0 & 0 & -r_1 - r_3 & r_1 & 0 & 0 \\
0 & r_3 & 0 & 0 & 0 & 0 & -r_2 - r_3 & 0 & r_2 \\
0 & 0 & r_3 & 0 & 0 & 0 & 0 & -r_2 - r_3 & r_2 \\
0 & 0 & 0 & r_3 & 0 & r_3 & 0 & 0 & -2r_3
\end{pmatrix}$$

The PEPA Eclipse Plug-in processing the model



Outline

PEPA Case Studies (1)

- Multiprocessor access-contention protocols (Gilmore, Hillston and Ribaudo, Edinburgh and Turin)
- Protocols for fault-tolerant systems (Clark, Gilmore, Hillston and Ribaudo, Edinburgh and Turin)
- Multimedia traffic characteristics (Bowman et al, Kent)
- QoS protocols for mobile devices (Wang *et al.*, EE department, Edinburgh)
- Software Architectures (Pooley, Bradley and Thomas, Heriot-Watt and Durham)
- Switch behaviour in active networks (Hillston, Kloul and Mokhtari, Edinburgh and Versailles)

PEPA Case Studies (1)

- Multiprocessor access-contention protocols (Gilmore, Hillston and Ribaudo, Edinburgh and Turin)
- Protocols for fault-tolerant systems (Clark, Gilmore, Hillston and Ribaudo, Edinburgh and Turin)
- Multimedia traffic characteristics (Bowman et al, Kent)
- QoS protocols for mobile devices (Wang *et al.*, EE department, Edinburgh)
- Software Architectures (Pooley, Bradley and Thomas, Heriot-Watt and Durham)
- Switch behaviour in active networks (Hillston, Kloul and Mokhtari, Edinburgh and Versailles)

PEPA Case Studies (1)

- Multiprocessor access-contention protocols (Gilmore, Hillston and Ribaudo, Edinburgh and Turin)
- Protocols for fault-tolerant systems (Clark, Gilmore, Hillston and Ribaudo, Edinburgh and Turin)
- Multimedia traffic characteristics (Bowman et al, Kent)
- QoS protocols for mobile devices (Wang *et al.*, EE department, Edinburgh)
- Software Architectures (Pooley, Bradley and Thomas, Heriot-Watt and Durham)
- Switch behaviour in active networks (Hillston, Kloul and Mokhtari, Edinburgh and Versailles)

PEPA Case Studies (1)

- Multiprocessor access-contention protocols (Gilmore, Hillston and Ribaudo, Edinburgh and Turin)
- Protocols for fault-tolerant systems (Clark, Gilmore, Hillston and Ribaudo, Edinburgh and Turin)
- Multimedia traffic characteristics (Bowman et al, Kent)
- QoS protocols for mobile devices (Wang *et al.*, EE department, Edinburgh)
- Software Architectures (Pooley, Bradley and Thomas, Heriot-Watt and Durham)
- Switch behaviour in active networks (Hillston, Kloul and Mokhtari, Edinburgh and Versailles)

PEPA Case Studies (1)

- Multiprocessor access-contention protocols (Gilmore, Hillston and Ribaudo, Edinburgh and Turin)
- Protocols for fault-tolerant systems (Clark, Gilmore, Hillston and Ribaudo, Edinburgh and Turin)
- Multimedia traffic characteristics (Bowman et al, Kent)
- QoS protocols for mobile devices (Wang *et al.*, EE department, Edinburgh)
- Software Architectures (Pooley, Bradley and Thomas, Heriot-Watt and Durham)
- Switch behaviour in active networks (Hillston, Kloul and Mokhtari, Edinburgh and Versailles)

PEPA Case Studies (1)

- Multiprocessor access-contention protocols (Gilmore, Hillston and Ribaudo, Edinburgh and Turin)
- Protocols for fault-tolerant systems (Clark, Gilmore, Hillston and Ribaudo, Edinburgh and Turin)
- Multimedia traffic characteristics (Bowman et al, Kent)
- QoS protocols for mobile devices (Wang *et al.*, EE department, Edinburgh)
- Software Architectures (Pooley, Bradley and Thomas, Heriot-Watt and Durham)
- Switch behaviour in active networks (Hillston, Kloul and Mokhtari, Edinburgh and Versailles)

PEPA Case Studies (2)

- Task scheduling in a Grid-based processing system, (Benoit *et al.*, Edinburgh)
- Probability of airbag deployment (Clark, Gilmore and Tribastone, Edinburgh)
- Crowd Interactions in Smart Environments (Harrison, Massink and Latella, Newcastle and ISTI)
- Spread of a computer virus via a computer network (Bradley, Gilmore and Hillston, Imperial and Edinburgh)
- Disease spread within animal populations (Benkrine, McCaig, Norman and Shankland, Stirling)

PEPA Case Studies (2)

- Task scheduling in a Grid-based processing system, (Benoit *et al.*, Edinburgh)
- Probability of airbag deployment (Clark, Gilmore and Tribastone, Edinburgh)
- Crowd Interactions in Smart Environments (Harrison, Massink and Latella, Newcastle and ISTI)
- Spread of a computer virus via a computer network (Bradley, Gilmore and Hillston, Imperial and Edinburgh)
- Disease spread within animal populations (Benkrine, McCaig, Norman and Shankland, Stirling)

PEPA Case Studies (2)

- Task scheduling in a Grid-based processing system, (Benoit *et al.*, Edinburgh)
- Probability of airbag deployment (Clark, Gilmore and Tribastone, Edinburgh)
- Crowd Interactions in Smart Environments (Harrison, Massink and Latella, Newcastle and ISTI)
- Spread of a computer virus via a computer network (Bradley, Gilmore and Hillston, Imperial and Edinburgh)
- Disease spread within animal populations (Benkrine, McCaig, Norman and Shankland, Stirling)

PEPA Case Studies (2)

- Task scheduling in a Grid-based processing system, (Benoit *et al.*, Edinburgh)
- Probability of airbag deployment (Clark, Gilmore and Tribastone, Edinburgh)
- Crowd Interactions in Smart Environments (Harrison, Massink and Latella, Newcastle and ISTI)
- Spread of a computer virus via a computer network (Bradley, Gilmore and Hillston, Imperial and Edinburgh)
- Disease spread within animal populations (Benkrine, McCaig, Norman and Shankland, Stirling)

PEPA Case Studies (2)

- Task scheduling in a Grid-based processing system, (Benoit *et al.*, Edinburgh)
- Probability of airbag deployment (Clark, Gilmore and Tribastone, Edinburgh)
- Crowd Interactions in Smart Environments (Harrison, Massink and Latella, Newcastle and ISTI)
- Spread of a computer virus via a computer network (Bradley, Gilmore and Hillston, Imperial and Edinburgh)
- Disease spread within animal populations (Benkrine, McCaig, Norman and Shankland, Stirling)

Outline

Roland the Gunslinger

- This sequence of small examples are based around a character called Roland Deschain.
- Roland is a gunslinger and his life consists of wandering around firing his gun.
- We will consider Roland in a number of different scenarios.
- These are not intended to be serious but they serve to
  - illustrate the main features of the language,
  - give you some experience of how models are constructed, and
  - demonstrate a variety of solution techniques.

Roland the Gunslinger

- This sequence of small examples are based around a character called Roland Deschain.
- Roland is a gunslinger and his life consists of wandering around firing his gun.
- We will consider Roland in a number of different scenarios.
- These are not intended to be serious but they serve to
  - illustrate the main features of the language,
  - give you some experience of how models are constructed, and
  - demonstrate a variety of solution techniques.

Roland the Gunslinger

- This sequence of small examples are based around a character called Roland Deschain.
- Roland is a gunslinger and his life consists of wandering around firing his gun.
- We will consider Roland in a number of different scenarios.
- These are not intended to be serious but they serve to
  - illustrate the main features of the language,
  - give you some experience of how models are constructed, and
  - demonstrate a variety of solution techniques.

Roland the Gunslinger

- This sequence of small examples are based around a character called Roland Deschain.
- Roland is a gunslinger and his life consists of wandering around firing his gun.
- We will consider Roland in a number of different scenarios.
- These are not intended to be serious but they serve to
  - illustrate the main features of the language,
  - give you some experience of how models are constructed, and
  - demonstrate a variety of solution techniques.

Roland alone

In the first scenario we consider Roland alone, with the single activity of firing his gun which is a six-shooter. When his gun is empty Roland will reload the gun and then continue shooting.

$$
\begin{aligned}
Roland_6 &\stackrel{def}{=} (fire, r_{fire}).Roland_5 \\
Roland_5 &\stackrel{def}{=} (fire, r_{fire}).Roland_4 \\
Roland_4 &\stackrel{def}{=} (fire, r_{fire}).Roland_3 \\
Roland_3 &\stackrel{def}{=} (fire, r_{fire}).Roland_2 \\
Roland_2 &\stackrel{def}{=} (fire, r_{fire}).Roland_1 \\
Roland_1 &\stackrel{def}{=} (fire, r_{fire}).Roland_{empty} \\
Roland_{empty} &\stackrel{def}{=} (reload, r_{reload}).Roland_6
\end{aligned}
$$

Roland with two guns

All self-respecting gun-slingers have one gun in each hand. If we suppose that Roland has two guns then he should be allowed to fire either gun independently. A simplistic model of this has two instances of *Roland* in parallel:

$$Roland_6 \parallel Roland_6$$

But this does not capture the fact that Roland needs both hands in order to reload either gun. The simplest solution is to assume that Roland only reloads both guns when both are empty.

$$Roland_6 \underset{\{reload\}}{\bowtie} Roland_6$$

From now on we restrict Roland to his shotgun, which has two shots and requires both hands for firing.

Roland with two guns

All self-respecting gun-slingers have one gun in each hand. If we suppose that Roland has two guns then he should be allowed to fire either gun independently. A simplistic model of this has two instances of *Roland* in parallel:

$$Roland_6 \parallel Roland_6$$

But this does not capture the fact that Roland needs both hands in order to reload either gun. The simplest solution is to assume that Roland only reloads both guns when both are empty.

$$Roland_6 \underset{\{reload\}}{\bowtie} Roland_6$$

From now on we restrict Roland to his shotgun, which has two shots and requires both hands for firing.

Roland with two guns

All self-respecting gun-slingers have one gun in each hand. If we suppose that Roland has two guns then he should be allowed to fire either gun independently. A simplistic model of this has two instances of *Roland* in parallel:

$$Roland_6 \parallel Roland_6$$

But this does not capture the fact that Roland needs both hands in order to reload either gun. The simplest solution is to assume that Roland only reloads both guns when both are empty.

$$Roland_6 \underset{\{reload\}}{\bowtie} Roland_6$$

From now on we restrict Roland to his shotgun, which has two shots and requires both hands for firing.

Roland meets an Enemy

- Upon his travels Roland encounters some enemies and when he does so he must fight them.
- Roland is the wildest gunslinger in the west so we assume that no enemy has the skill to seriously harm Roland.
- Each time Roland fires he might miss or hit his target.
- But with nothing to stop him he will keep firing until he successfully hits (and kills) the enemy.
- We assume that some sense of cowboy honour prevents any enemy attacking Roland if he is already involved in a gun fight.

The model

$$
\begin{aligned}
Roland_{idle} &\stackrel{def}{=} (attack, r_{attack}).Roland_2 \\
Roland_2 &\stackrel{def}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} \\
&\quad +(miss, r_{miss}).Roland_1 \\
Roland_1 &\stackrel{def}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} \\
&\quad +(miss, r_{miss}).Roland_{empty} \\
Roland_{empty} &\stackrel{def}{=} (reload, r_{reload}).Roland_2
\end{aligned}
$$

Parameter settings for the $Roland_2$ model

parameter	*value*	*explanation*
r_{fire}	1.0	Roland can fire the gun once per-second
$p_{hit\text{-}success}$	0.8	Roland has an 80% success rate
r_{hit}	0.8	$r_{fire} \times p_{hit\text{-}success}$
r_{miss}	0.2	$r_{fire} \times (1 - p_{hit\text{-}success})$
r_{reload}	0.3	It takes Roland about 3 seconds to reload
r_{attack}	0.01	Roland is attacked once every 100 seconds

Steady state analysis

We can calculate the probability that at arbitrary time Roland is involved in a battle.

This can be based on the steady state probability that *Roland* is in any of the states in which a battle is on-going, i.e. $Roland_2$, $Roland_1$ and $Roland_{empty}$.

```
 State Measure 'roland peaceful'
mean               9.5490716180e-01
State Measure 'roland in battle'
mean               0.0450928382e-01
```

$>$ 95% chance that Roland is not currently involved in a gun battle.

Steady state analysis

We can calculate the probability that at arbitrary time Roland is involved in a battle.

This can be based on the steady state probability that *Roland* is in any of the states in which a battle is on-going, i.e. $Roland_2$, $Roland_1$ and $Roland_{empty}$.

```
 State Measure 'roland peaceful'
mean                9.5490716180e-01
State Measure 'roland in battle'
mean                0.0450928382e-01
```

$>$ 95% chance that Roland is not currently involved in a gun battle.

Steady state analysis

We can calculate the probability that at arbitrary time Roland is involved in a battle.

This can be based on the steady state probability that *Roland* is in any of the states in which a battle is on-going, i.e. $Roland_2$, $Roland_1$ and $Roland_{empty}$.

```
 State Measure 'roland peaceful'
mean                9.5490716180e-01
State Measure 'roland in battle'
mean                0.0450928382e-01
```

$>$ 95% chance that Roland is not currently involved in a gun battle.

Steady state analysis

We can calculate the probability that at arbitrary time Roland is involved in a battle.

This can be based on the steady state probability that *Roland* is in any of the states in which a battle is on-going, i.e. $Roland_2$, $Roland_1$ and $Roland_{empty}$.

```
 State Measure 'roland peaceful'
mean               9.5490716180e-01
State Measure 'roland in battle'
mean               0.0450928382e-01
```

$>$ 95% chance that Roland is not currently involved in a gun battle.

Passage-Time Analysis

Passage-time analysis allows us to calculate measures such as the probability that Roland has killed his enemy at a given time after he is attacked.

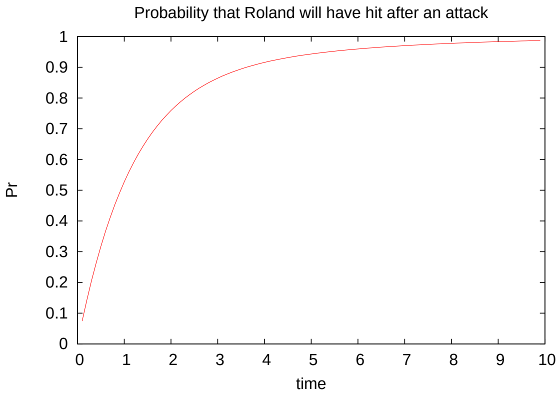
This would involve calculating the probability that the model performs a *hit* action within the given time after performing an *attack* action.

Passage-Time Analysis

Passage-time analysis allows us to calculate measures such as the probability that Roland has killed his enemy at a given time after he is attacked.

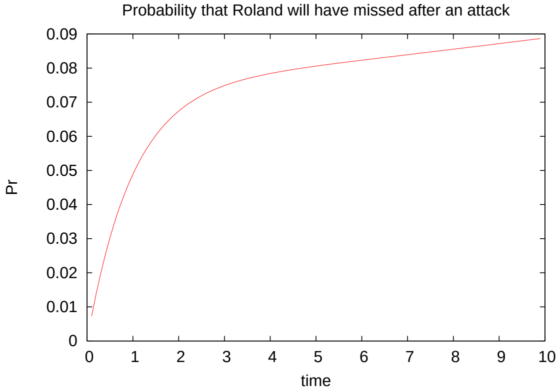
This would involve calculating the probability that the model performs a *hit* action within the given time after performing an *attack* action.

Passage-Time Analysis results



The probability that Roland will successfully perform a *hit* action a given time after an *attack*. Gun battles typically last about 5 seconds and occurs about every 100 seconds. The probability that Roland has performed a *hit* action five seconds after an *attack* action is $\approx 90\%$

Passage-Time Analysis results



The probability that Roland has performed a *miss* action a given time after an *attack* action. These probabilities are much lower because Roland's probability of success are high and if he successfully kills an enemy we must wait for the next attack in order to have a chance of seeing a *miss*.

Cooperation

- In the previous model Roland's enemies were represented only implicitly.
- We now consider a model in which the enemies appear explicitly and allow them to fight back.
- However for now we still assume that they are rather ineffectual and so they never seriously injure Roland.
- This model can be used to calculate properties such as the likelihood that an enemy will manage to fire one shot before they are killed by Roland.

Cooperation

- In the previous model Roland's enemies were represented only implicitly.
- We now consider a model in which the enemies appear explicitly and allow them to fight back.
- However for now we still assume that they are rather ineffectual and so they never seriously injure Roland.
- This model can be used to calculate properties such as the likelihood that an enemy will manage to fire one shot before they are killed by Roland.

Cooperation

- In the previous model Roland's enemies were represented only implicitly.
- We now consider a model in which the enemies appear explicitly and allow them to fight back.
- However for now we still assume that they are rather ineffectual and so they never seriously injure Roland.
- This model can be used to calculate properties such as the likelihood that an enemy will manage to fire one shot before they are killed by Roland.

Cooperation

- In the previous model Roland's enemies were represented only implicitly.
- We now consider a model in which the enemies appear explicitly and allow them to fight back.
- However for now we still assume that they are rather ineffectual and so they never seriously injure Roland.
- This model can be used to calculate properties such as the likelihood that an enemy will manage to fire one shot before they are killed by Roland.

Revised Model

$$
\begin{aligned}
Roland_{idle} &\stackrel{def}{=} (attack, \top).Roland_2 \\
Roland_2 &\stackrel{def}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} \\
&\qquad + (miss, r_{miss}).Roland_1 \\
Roland_1 &\stackrel{def}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} \\
&\qquad + (miss, r_{miss}).Roland_{empty} \\
Roland_{empty} &\stackrel{def}{=} (reload, r_{reload}).Roland_2 \\
\\
Enemies_{idle} &\stackrel{def}{=} (attack, r_{attack}).Enemies_{attack} \\
Enemies_{attack} &\stackrel{def}{=} (fire, r_{e\text{-}miss}).Enemies_{attack} \\
&\qquad + (hit, \top).Enemies_{idle} \\
\\
Roland_{idle} &\underset{\{hit, attack\}}{\bowtie} Enemies_{idle}
\end{aligned}
$$

Additional parameters

parameter	*value*	*explanation*
r_{attack}	0.01	Roland is attacked once every 100 seconds
$r_{e\text{-}miss}$	0.3	Enemies can fire only once every 3 seconds

Levels of abstraction

- Notice that in this model the behaviour of the enemy has been simplified.
- There is no running out of bullets or reloading.
- This model can be thought of as an approximation to a more complicated component similar to the one which models Roland.
- Here the rate at which the enemy fires encompasses all of the actions, including the reloading of an empty gun.
- We may choose to model a component in such an abstract way when the focus of our modelling is really elsewhere in the model.

Levels of abstraction

- Notice that in this model the behaviour of the enemy has been simplified.
- There is no running out of bullets or reloading.
- This model can be thought of as an approximation to a more complicated component similar to the one which models Roland.
- Here the rate at which the enemy fires encompasses all of the actions, including the reloading of an empty gun.
- We may choose to model a component in such an abstract way when the focus of our modelling is really elsewhere in the model.

Levels of abstraction

- Notice that in this model the behaviour of the enemy has been simplified.
- There is no running out of bullets or reloading.
- This model can be thought of as an approximation to a more complicated component similar to the one which models Roland.
- Here the rate at which the enemy fires encompasses all of the actions, including the reloading of an empty gun.
- We may choose to model a component in such an abstract way when the focus of our modelling is really elsewhere in the model.

Levels of abstraction

- Notice that in this model the behaviour of the enemy has been simplified.
- There is no running out of bullets or reloading.
- This model can be thought of as an approximation to a more complicated component similar to the one which models Roland.
- Here the rate at which the enemy fires encompasses all of the actions, including the reloading of an empty gun.
- We may choose to model a component in such an abstract way when the focus of our modelling is really elsewhere in the model.

Levels of abstraction

- Notice that in this model the behaviour of the enemy has been simplified.
- There is no running out of bullets or reloading.
- This model can be thought of as an approximation to a more complicated component similar to the one which models Roland.
- Here the rate at which the enemy fires encompasses all of the actions, including the reloading of an empty gun.
- We may choose to model a component in such an abstract way when the focus of our modelling is really elsewhere in the model.

Model Validation

It is also sometimes useful to carry out a validation of the model by calculating a metric which we believe we already know the value of.

For example in this model we could make such a sanity check by calculating the probability that the model is in a state in which Roland is idle but the enemies are not, or vice versa.

This should never occur and hence the probability should be zero.

Model Validation

It is also sometimes useful to carry out a validation of the model by calculating a metric which we believe we already know the value of.

For example in this model we could make such a sanity check by calculating the probability that the model is in a state in which Roland is idle but the enemies are not, or vice versa.

This should never occur and hence the probability should be zero.

Model Validation

It is also sometimes useful to carry out a validation of the model by calculating a metric which we believe we already know the value of.

For example in this model we could make such a sanity check by calculating the probability that the model is in a state in which Roland is idle but the enemies are not, or vice versa.

This should never occur and hence the probability should be zero.

Sensitivity Analysis

- Sensitivity analysis studies how much influence particular parameter values, such as activity rates, have on performance metrics calculated for the system as a whole.
- A single activity in a PEPA model may have a significant impact on the dynamics of the model, or, conversely, may exert very little influence.
- Sensitivity analysis is performed by solving the model many times while varying the rates slightly.
- For this model we chose to vary three of the rates involved and measured the passage time between an *attack* and a *hit* activity, for each combintation of rates.

Sensitivity Analysis

- Sensitivity analysis studies how much influence particular parameter values, such as activity rates, have on performance metrics calculated for the system as a whole.
- A single activity in a PEPA model may have a significant impact on the dynamics of the model, or, conversely, may exert very little influence.
- Sensitivity analysis is performed by solving the model many times while varying the rates slightly.
- For this model we chose to vary three of the rates involved and measured the passage time between an *attack* and a *hit* activity, for each combintation of rates.

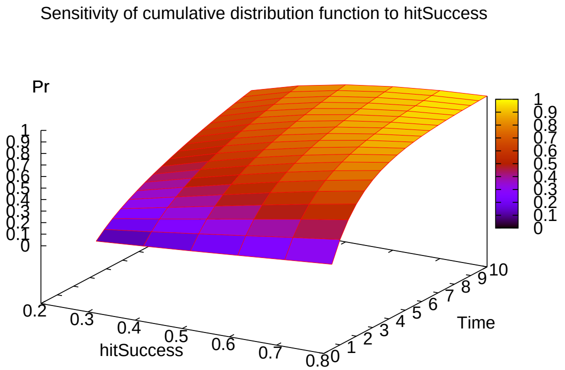
Sensitivity Analysis

- Sensitivity analysis studies how much influence particular parameter values, such as activity rates, have on performance metrics calculated for the system as a whole.
- A single activity in a PEPA model may have a significant impact on the dynamics of the model, or, conversely, may exert very little influence.
- Sensitivity analysis is performed by solving the model many times while varying the rates slightly.
- For this model we chose to vary three of the rates involved and measured the passage time between an *attack* and a *hit* activity, for each combintation of rates.

Sensitivity Analysis

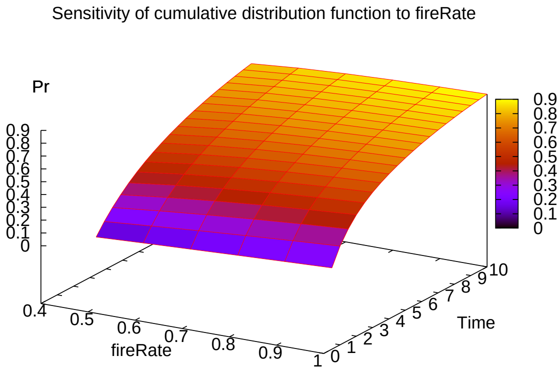
- Sensitivity analysis studies how much influence particular parameter values, such as activity rates, have on performance metrics calculated for the system as a whole.
- A single activity in a PEPA model may have a significant impact on the dynamics of the model, or, conversely, may exert very little influence.
- Sensitivity analysis is performed by solving the model many times while varying the rates slightly.
- For this model we chose to vary three of the rates involved and measured the passage time between an *attack* and a *hit* activity, for each combintation of rates.

Sensitivity Analysis: Results



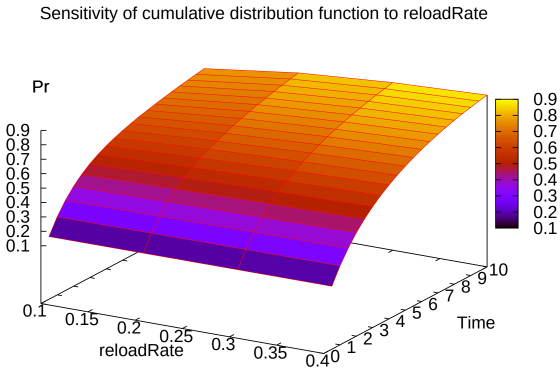
The effect of varying the $p_{hit\text{-}success}$ parameter.

Sensitivity Analysis: Results



The effect of varying the r_{fire} parameter.

Sensitivity Analysis: Results



The effect of varying the r_{reload} parameter.

Accurate Enemies

- We now allow the enemies of Roland to actually hit him. This means that Roland may die. It is important to note that this has the consequence that the model will always deadlock. The underlying Markov process is no longer ergodic.
- We assume that the enemies can only hit Roland once every 50 seconds. This rate approximates the rate of a more detailed model in which we would assign a process to the enemies which is much like that of the process which describes Roland.
- The only new parameter is $r_{e\text{-}hit}$ which is assigned a value 0.02 to reflect this assumption.

Accurate Enemies

- We now allow the enemies of Roland to actually hit him. This means that Roland may die. It is important to note that this has the consequence that the model will always deadlock. The underlying Markov process is no longer ergodic.
- We assume that the enemies can only hit Roland once every 50 seconds. This rate approximates the rate of a more detailed model in which we would assign a process to the enemies which is much like that of the process which describes Roland.
- The only new parameter is $r_{e\text{-}hit}$ which is assigned a value 0.02 to reflect this assumption.

Accurate Enemies

- We now allow the enemies of Roland to actually hit him. This means that Roland may die. It is important to note that this has the consequence that the model will always deadlock. The underlying Markov process is no longer ergodic.
- We assume that the enemies can only hit Roland once every 50 seconds. This rate approximates the rate of a more detailed model in which we would assign a process to the enemies which is much like that of the process which describes Roland.
- The only new parameter is $r_{e\text{-}hit}$ which is assigned a value 0.02 to reflect this assumption.

New Roland

$$
\begin{aligned}
Roland_{idle} &\stackrel{def}{=} (attack, \top).Roland_2 \\
Roland_2 &\stackrel{def}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} \\
&+ (miss, r_{miss}).Roland_1 \\
&+ (e\text{-}hit, \top).Roland_{dead} \\
Roland_1 &\stackrel{def}{=} (hit, r_{hit}).(reload, r_{reload}).Roland_{idle} \\
&+ (miss, r_{miss}).Roland_{empty} \\
&+ (e\text{-}hit, \top).Roland_{dead} \\
Roland_{empty} &\stackrel{def}{=} (reload, r_{reload}).(reload, r_{reload}).Roland_2 \\
&+ (e\text{-}hit, \top).Roland_{dead} \\
Roland_{dead} &\stackrel{def}{=} Stop
\end{aligned}
$$

New Enemy

$$
\begin{aligned}
Enemies_{idle} &\stackrel{def}{=} (attack, r_{attack}).Enemies_{attack} \\
Enemies_{attack} &\stackrel{def}{=} (e\text{-}hit, r_{e\text{-}hit}).Enemies_{idle} \\
&+ (hit, \top).Enemies_{idle}
\end{aligned}
$$

$$
Roland_{idle} \underset{\{hit, attack, e\text{-}hit\}}{\bowtie} Enemies_{idle}
$$

Model Analysis

Steady-State Analysis Since there is an infinite supply of enemies eventually Roland will always die and the model will deadlock.

Transient Analysis Transient analysis on this model can be used to calculate the probability that Roland is dead after a given amount of time. As time increases this should tend towards probability 1.

Passage-Time Analysis Passage-time analysis could be used to calculate the probability of a given event happening at a given time after another given event, e.g. from an attack on Roland until he dies or wins the gun fight.

Model Analysis

Steady-State Analysis Since there is an infinite supply of enemies eventually Roland will always die and the model will deadlock.

Transient Analysis Transient analysis on this model can be used to calculate the probability that Roland is dead after a given amount of time. As time increases this should tend towards probability 1.

Passage-Time Analysis Passage-time analysis could be used to calculate the probability of a given event happening at a given time after another given event, e.g. from an attack on Roland until he dies or wins the gun fight.

Model Analysis

Steady-State Analysis Since there is an infinite supply of enemies eventually Roland will always die and the model will deadlock.

Transient Analysis Transient analysis on this model can be used to calculate the probability that Roland is dead after a given amount of time. As time increases this should tend towards probability 1.

Passage-Time Analysis Passage-time analysis could be used to calculate the probability of a given event happening at a given time after another given event, e.g. from an attack on Roland until he dies or wins the gun fight.

Roland makes a friend

In the next revision of the model we introduce an accomplice who is befriended by Roland and who, when Roland is attacked, fights alongside him.

In this scenario cooperation is used to synchronise between components of the model such that they observe events which they neither directly cause nor are directly affected by.

Whenever either Roland or the accomplice kills the enemy the other must witness this action, so as to stop firing at a dead opponent (it would be a waste of ammunition!).

Roland makes a friend

In the next revision of the model we introduce an accomplice who is befriended by Roland and who, when Roland is attacked, fights alongside him.

In this scenario cooperation is used to synchronise between components of the model such that they observe events which they neither directly cause nor are directly affected by.

Whenever either Roland or the accomplice kills the enemy the other must witness this action, so as to stop firing at a dead opponent (it would be a waste of ammunition!).

Roland makes a friend

In the next revision of the model we introduce an accomplice who is befriended by Roland and who, when Roland is attacked, fights alongside him.

In this scenario cooperation is used to synchronise between components of the model such that they observe events which they neither directly cause nor are directly affected by.

Whenever either Roland or the accomplice kills the enemy the other must witness this action, so as to stop firing at a dead opponent (it would be a waste of ammunition!).

A new component for Roland

$$
\begin{aligned}
Roland_{idle} &\stackrel{def}{=} (attack, \top).Roland_2 \\
&+ (befriend, r_{befriend}).Roland_{idle} \\
Roland_2 &\stackrel{def}{=} (hit, r_{hit}).Roland_{hit} + (miss, r_{miss}).Roland_1 \\
&+ (a\text{-}hit, \top).Roland_{idle} \\
Roland_1 &\stackrel{def}{=} (hit, r_{hit}).Roland_{hit} \\
&+ (miss, r_{miss}).Roland_{empty} \\
&+ (a\text{-}hit, \top).(reload, r_{reload}).Roland_{idle} \\
Roland_{hit} &\stackrel{def}{=} (enemy\text{-}die, \top).(reload, r_{reload}).Roland_{idle} \\
Roland_{empty} &\stackrel{def}{=} (reload, r_{reload}).Roland_2 \\
&+ (a\text{-}hit, \top).(reload, r_{reload}).Roland_{idle}
\end{aligned}
$$

Synchronising Roland and the Accomplice

- When there is an accomplice, he and Roland fight together against the enemy.
- This involves some cooperation between them.
- However we do not want to leave Roland vulnerable when there is no accomplice present because some of his actions become blocked.
- To prevent this we introduce a dummy component representing the absence of an accomplice.
- In this state the accomplice component will passively participate in any attack which Roland makes.

Synchronising Roland and the Accomplice

- When there is an accomplice, he and Roland fight together against the enemy.
- This involves some cooperation between them.
- However we do not want to leave Roland vulnerable when there is no accomplice present because some of his actions become blocked.
- To prevent this we introduce a dummy component representing the absence of an accomplice.
- In this state the accomplice component will passively participate in any attack which Roland makes.

Synchronising Roland and the Accomplice

- When there is an accomplice, he and Roland fight together against the enemy.
- This involves some cooperation between them.
- However we do not want to leave Roland vulnerable when there is no accomplice present because some of his actions become blocked.
- To prevent this we introduce a dummy component representing the absence of an accomplice.
- In this state the accomplice component will passively participate in any attack which Roland makes.

Synchronising Roland and the Accomplice

- When there is an accomplice, he and Roland fight together against the enemy.
- This involves some cooperation between them.
- However we do not want to leave Roland vulnerable when there is no accomplice present because some of his actions become blocked.
- To prevent this we introduce a dummy component representing the absence of an accomplice.
- In this state the accomplice component will passively participate in any attack which Roland makes.

Synchronising Roland and the Accomplice

- When there is an accomplice, he and Roland fight together against the enemy.
- This involves some cooperation between them.
- However we do not want to leave Roland vulnerable when there is no accomplice present because some of his actions become blocked.
- To prevent this we introduce a dummy component representing the absence of an accomplice.
- In this state the accomplice component will passively participate in any attack which Roland makes.

Synchronising Roland and the Accomplice

- When there is an accomplice, he and Roland fight together against the enemy.
- This involves some cooperation between them.
- However we do not want to leave Roland vulnerable when there is no accomplice present because some of his actions become blocked.
- To prevent this we introduce a dummy component representing the absence of an accomplice.
- In this state the accomplice component will passively participate in any attack which Roland makes.

$$
\begin{aligned}
Acmpl_{abs} \stackrel{def}{=} \; & (befriend, r_{befriend}).Acmpl_{idle} \\
& + (hit, \top).Acmpl_{abs} \\
& + (attack, \top).Acmpl_{abs}
\end{aligned}
$$

Component for the Accomplice

$$
\begin{aligned}
Acmpl_{idle} &\stackrel{def}{=} (attack, \top).Acmpl_2 \\
Acmpl_2 &\stackrel{def}{=} (a\text{-}hit, r_{a\text{-}hit}).Acmpl_{hit} + (hit, \top).Acmpl_{idle} \\
&+ (miss, r_{miss}).Acmpl_1 \\
&+ (enemy\text{-}hit, \top).Acmpl_{abs} \\
Acmpl_1 &\stackrel{def}{=} (a\text{-}hit, r_{a\text{-}hit}).Acmpl_{hit} \\
&+ (hit, \top).(reload, r_{a\text{-}reload}).Acmpl_{idle} \\
&+ (miss, r_{miss}).Acmpl_{empty} \\
&+ (enemy\text{-}hit, \top).Acmpl_{abs} \\
Acmpl_{hit} &\stackrel{def}{=} (enemy\text{-}die, \top).(reload, r_{a\text{-}reload}).Acmpl_{idle} \\
Acmpl_{empty} &\stackrel{def}{=} (reload, r_{a\text{-}reload}).Acmpl_2 \\
&+ (enemy\text{-}hit, \top).Acmpl_{abs} \\
&+ (hit, \top).(reload, r_{a\text{-}reload}).Acmpl_{idle}
\end{aligned}
$$

Parameter Settings for the Accomplice

parameter	*value*	*explanation*
$r_{befriend}$	0.001	Roland befriends a stranger once every 1000 seconds
$r_{a\text{-}fire}$	1.0	the accomplice can also fire once per second
$p_{a\text{-}hit\text{-}success}$	0.6	the accomplice has a 60 percent accuracy
$r_{a\text{-}hit}$	0.6	$r_{fire} \times p_{hit\text{-}success}$
$r_{a\text{-}miss}$	0.4	$r_{fire} \times (1.0 - p_{hit\text{-}success})$
$r_{a\text{-}reload}$	0.25	it takes the accomplice 4 seconds to reload

Component for the Enemy

The component representing the enemy is similar to before.

$$
\begin{aligned}
Enemies_{idle} &\stackrel{def}{=} (attack, r_{attack}).Enemies_{attack} \\
Enemies_{attack} &\stackrel{def}{=} (enemy\text{-}hit, r_{e\text{-}hit}).Enemies_{attack} \\
&\quad + (enemy\text{-}die, r_{die}).Enemies_{idle}
\end{aligned}
$$

The system equation is as follows:

$$
(Roland_2 \underset{\{attack, hit, a\text{-}hit, befriend\}}{\bowtie} Acmpl_{abs}) \underset{\{attack, enemy\text{-}die, enemy\text{-}hit\}}{\bowtie} Enemies_{idle}
$$

Model Analysis

Steady-State Analysis

- As before we can determine the probability that Roland is involved in a gun battle at an arbitrary time.
- We could also determine the likelihood that Roland has an accomplice at an arbitrary time.
- Since Roland cannot perform a befriending action while currently involved in a battle, the probabilty that Roland is in such a battle clearly affects the probability that he is alone in his quest.
- So, for example, if Roland's success rate is reduced then gun battles will take longer to resolve, hence Roland will be involved in a gun battle more often, and therefore he will befriend fewer accomplices.

Model Analysis

Steady-State Analysis

- As before we can determine the probability that Roland is involved in a gun battle at an arbitrary time.
- We could also determine the likelihood that Roland has an accomplice at an arbitrary time.
- Since Roland cannot perform a befriending action while currently involved in a battle, the probabilty that Roland is in such a battle clearly affects the probability that he is alone in his quest.
- So, for example, if Roland's success rate is reduced then gun battles will take longer to resolve, hence Roland will be involved in a gun battle more often, and therefore he will befriend fewer accomplices.

Model Analysis

Steady-State Analysis

- As before we can determine the probability that Roland is involved in a gun battle at an arbitrary time.
- We could also determine the likelihood that Roland has an accomplice at an arbitrary time.
- Since Roland cannot perform a befriending action while currently involved in a battle, the probabilty that Roland is in such a battle clearly affects the probability that he is alone in his quest.
- So, for example, if Roland's success rate is reduced then gun battles will take longer to resolve, hence Roland will be involved in a gun battle more often, and therefore he will befriend fewer accomplices.

Model Analysis

Steady-State Analysis

- As before we can determine the probability that Roland is involved in a gun battle at an arbitrary time.
- We could also determine the likelihood that Roland has an accomplice at an arbitrary time.
- Since Roland cannot perform a befriending action while currently involved in a battle, the probabilty that Roland is in such a battle clearly affects the probability that he is alone in his quest.
- So, for example, if Roland's success rate is reduced then gun battles will take longer to resolve, hence Roland will be involved in a gun battle more often, and therefore he will befriend fewer accomplices.

Model Analysis

Transient Analysis

An example transient analysis would be to determine the expected time after Roland has set off before he meets his first accomplice.

Model Analysis

Passage-Time Analysis

- An example analysis would be to calculate the passage-time from an *attack* action until the death of the enemy or of the accomplice.
- Since all gun battles now end in the enemy being killed stopping the analysis there would give us the expected duration of any one gun battle.
- There is also the possibility to start the analysis from the *befriend* action and stop it with the death of the accomplice.

Model Analysis

Passage-Time Analysis

- An example analysis would be to calculate the passage-time from an *attack* action until the death of the enemy or of the accomplice.
- Since all gun battles now end in the enemy being killed stopping the analysis there would give us the expected duration of any one gun battle.
- There is also the possibility to start the analysis from the *befriend* action and stop it with the death of the accomplice.

Model Analysis

Passage-Time Analysis

- An example analysis would be to calculate the passage-time from an *attack* action until the death of the enemy or of the accomplice.
- Since all gun battles now end in the enemy being killed stopping the analysis there would give us the expected duration of any one gun battle.
- There is also the possibility to start the analysis from the *befriend* action and stop it with the death of the accomplice.

Hiding

- Currently there is nothing in the model to stop an enemy from disrupting the interaction between Roland and his accomplice, e.g. by performing a *befriend* action.
- One way to avoid this is to 'hide' those actions only Roland and the accomplice should cooperate on.
- To do this for our model we can simply change the system equation:

$$((Roland_2 \underset{L_1}{\bowtie} Acmpl)/L_1) \underset{L_2}{\bowtie} Enemies_{idle}$$

where $L_1 = \{hit, a\text{-}hit, befriend\}$ and $L_2 = \{attack, enemy\text{-}die, enemy\text{-}hit\}$.

Hiding

- Currently there is nothing in the model to stop an enemy from disrupting the interaction between Roland and his accomplice, e.g. by performing a *befriend* action.
- One way to avoid this is to 'hide' those actions only Roland and the accomplice should cooperate on.
- To do this for our model we can simply change the system equation:

$$((Roland_2 \underset{L_1}{\bowtie} Acmpl)/L_1) \underset{L_2}{\bowtie} Enemies_{idle}$$

where $L_1 = \{hit, a\text{-}hit, befriend\}$ and $L_2 = \{attack, enemy\text{-}die, enemy\text{-}hit\}$.

Hiding

- Currently there is nothing in the model to stop an enemy from disrupting the interaction between Roland and his accomplice, e.g. by performing a *befriend* action.
- One way to avoid this is to 'hide' those actions only Roland and the accomplice should cooperate on.
- To do this for our model we can simply change the system equation:

$$((Roland_2 \underset{L_1}{\bowtie} Acmpl)/L_1) \underset{L_2}{\bowtie} Enemies_{idle}$$

where $L_1 = \{hit, a\text{-}hit, befriend\}$ and $L_2 = \{attack, enemy\text{-}die, enemy\text{-}hit\}$.

Outline

The PEPA Eclipse Plug-in

Calculating by hand the transitions of a PEPA model and subsequently expressing these in a form which was suitable for solution was a tedious task prone to errors. The PEPA Eclipse Plug-in relieves the modeller of this work.

The PEPA Eclipse Plug-in: functionality

The plug-in will report errors in the model function:

- deadlock,
- absorbing states,
- static synchronisation mismatch (cooperations which do not involve active participants).

The plug-in also generates the transition graph of the model, computes the number of states, formulates the Markov process matrix Q and communicates the matrix to a solver.

The plug-in provides a simple pattern language for selecting states from the stationary distribution.

The PEPA Eclipse Plug-in: functionality

The plug-in will report errors in the model function:

- deadlock,
- absorbing states,
- static synchronisation mismatch (cooperations which do not involve active participants).

The plug-in also generates the transition graph of the model, computes the number of states, formulates the Markov process matrix $\boldsymbol{Q}$ and communicates the matrix to a solver.

The plug-in provides a simple pattern language for selecting states from the stationary distribution.

The PEPA Eclipse Plug-in: functionality

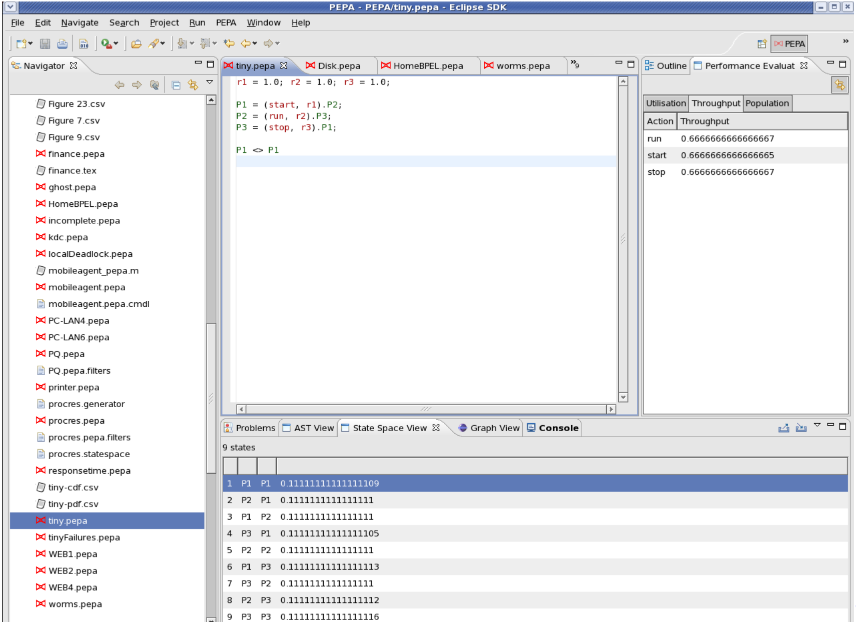
The plug-in will report errors in the model function:

- deadlock,
- absorbing states,
- static synchronisation mismatch (cooperations which do not involve active participants).

The plug-in also generates the transition graph of the model, computes the number of states, formulates the Markov process matrix $\boldsymbol{Q}$ and communicates the matrix to a solver.

The plug-in provides a simple pattern language for selecting states from the stationary distribution.

The PEPA Eclipse Plug-in processing the model



The PEPA website

`http://www.dcs.ed.ac.uk/pepa`

From the website the PEPA Eclipse Plug-in and some other tools are available for download.

There is also information about people involved in the PEPA project, projects undertaken and a collection of published papers.

Outline

Web Service Composition: Introduction

We consider an example of a business application which is composed from a number of offered web services.

A user accesses the application via an SMS message requesting directions to the nearest facility (post-office, restaurant, bank etc.) and receives a response as an MMS message containing a map.

Since the application involves a users' current location there is an access control issue since it must be ensured that the web service consumer has the requisite authority to execute the web service it requests.

Moreover the service provider imposes a restriction that only one request may be handled for each SMS message received.

Web Service Composition: Introduction

We consider an example of a business application which is composed from a number of offered web services.

A user accesses the application via an SMS message requesting directions to the nearest facility (post-office, restaurant, bank etc.) and receives a response as an MMS message containing a map.

Since the application involves a users' current location there is an access control issue since it must be ensured that the web service consumer has the requisite authority to execute the web service it requests.

Moreover the service provider imposes a restriction that only one request may be handled for each SMS message received.

Web Service Composition: Introduction

We consider an example of a business application which is composed from a number of offered web services.

A user accesses the application via an SMS message requesting directions to the nearest facility (post-office, restaurant, bank etc.) and receives a response as an MMS message containing a map.

Since the application involves a users' current location there is an access control issue since it must be ensured that the web service consumer has the requisite authority to execute the web service it requests.

Moreover the service provider imposes a restriction that only one request may be handled for each SMS message received.

Web Service Composition: Introduction

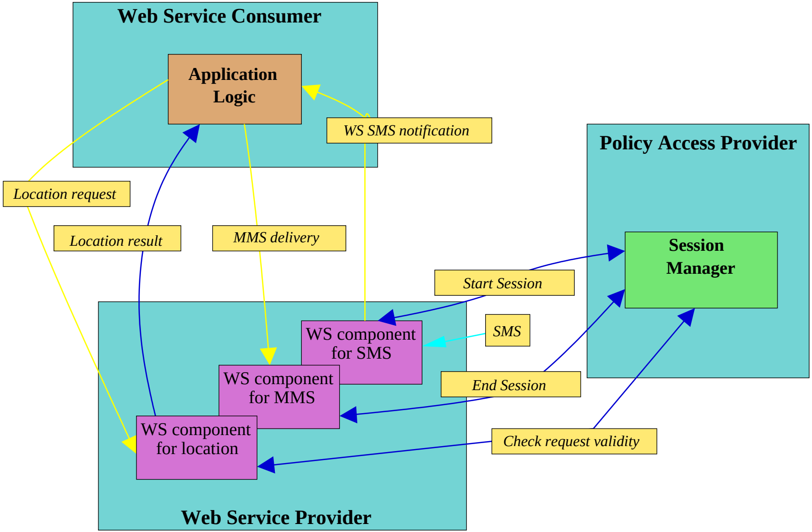
We consider an example of a business application which is composed from a number of offered web services.

A user accesses the application via an SMS message requesting directions to the nearest facility (post-office, restaurant, bank etc.) and receives a response as an MMS message containing a map.

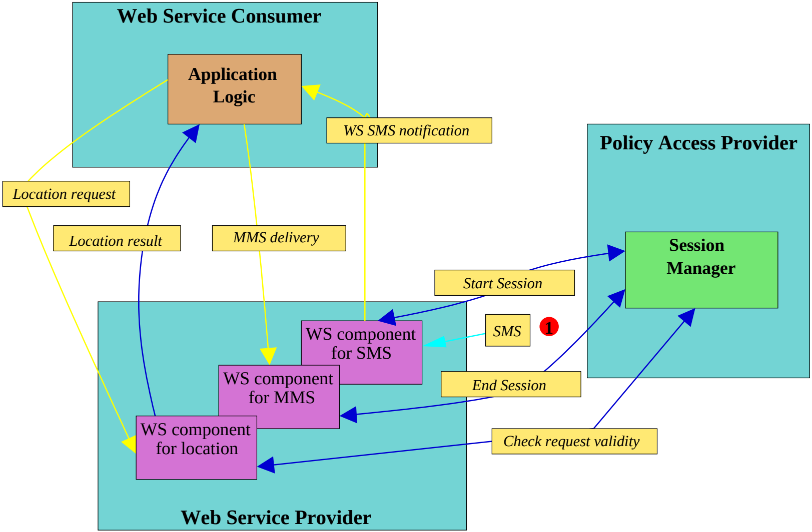
Since the application involves a users' current location there is an access control issue since it must be ensured that the web service consumer has the requisite authority to execute the web service it requests.

Moreover the service provider imposes a restriction that only one request may be handled for each SMS message received.

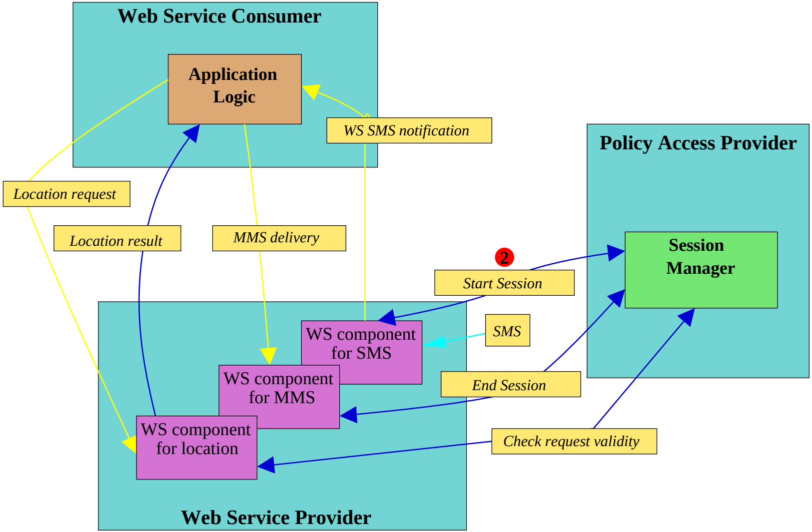
Schematic view



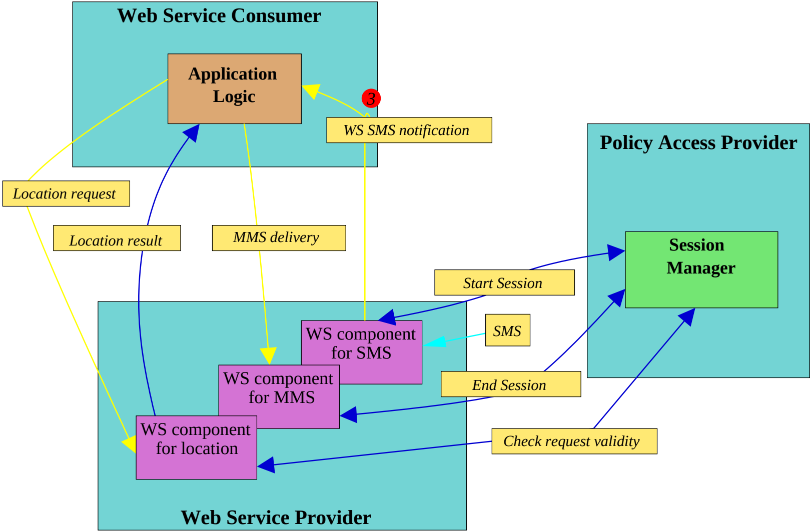
Schematic view



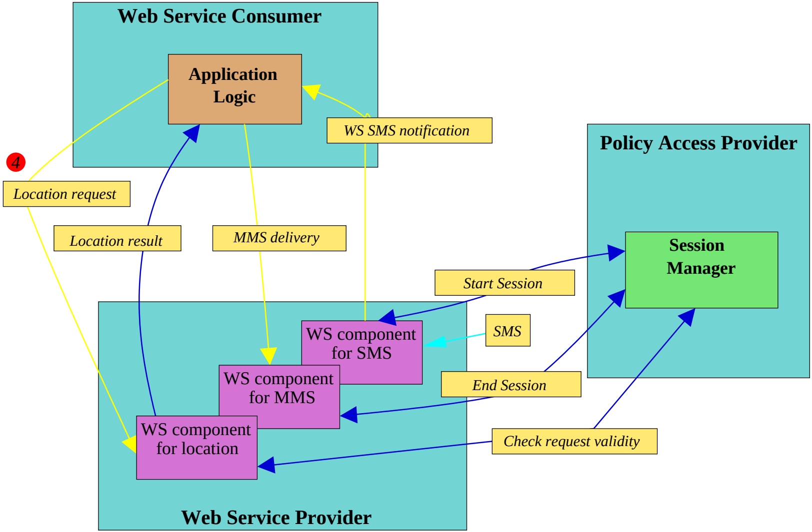
Schematic view



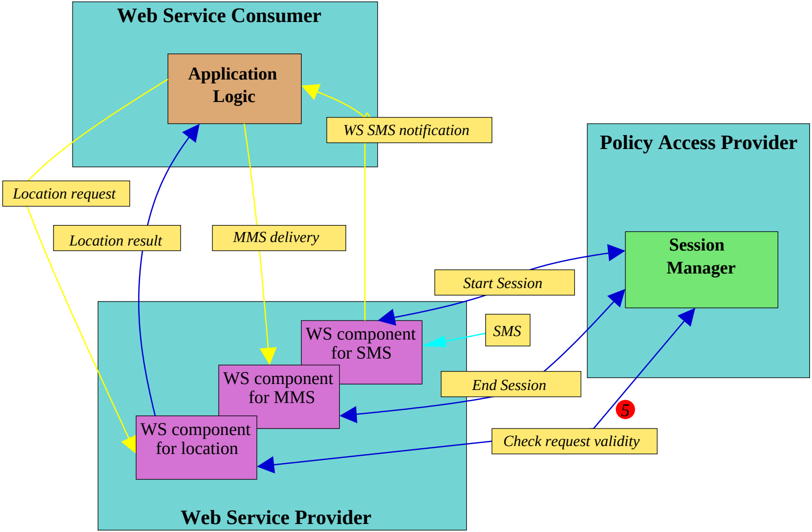
Schematic view



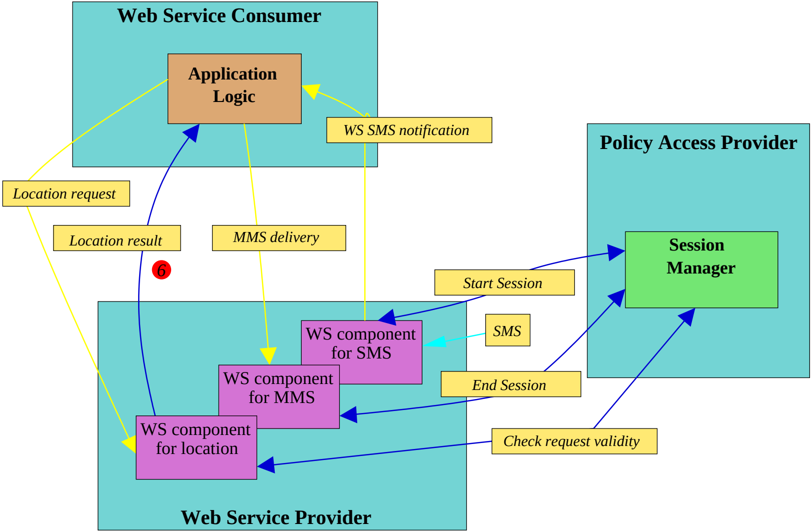
Schematic view



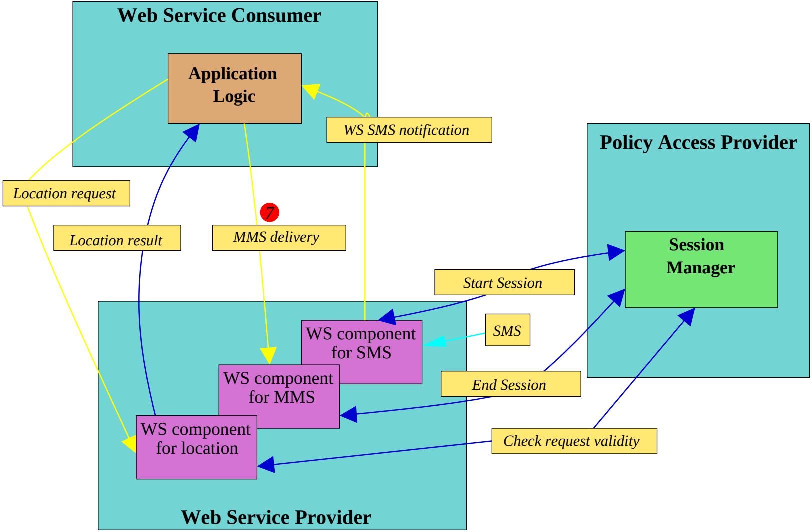
Schematic view



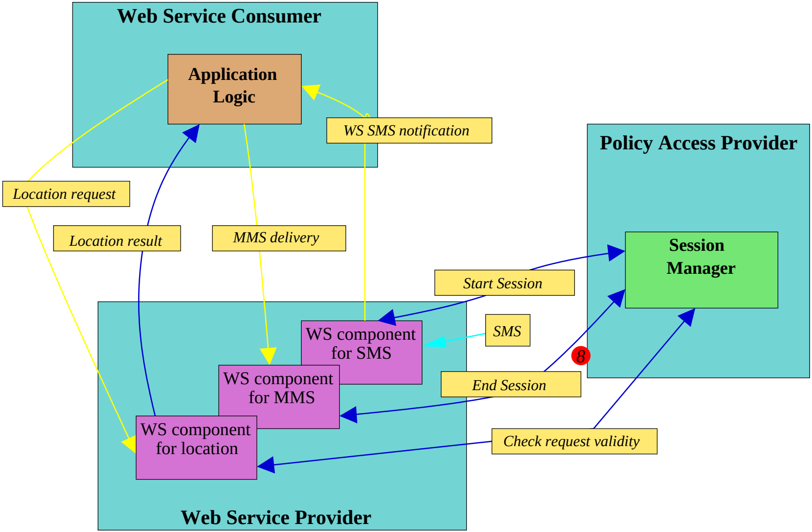
Schematic view



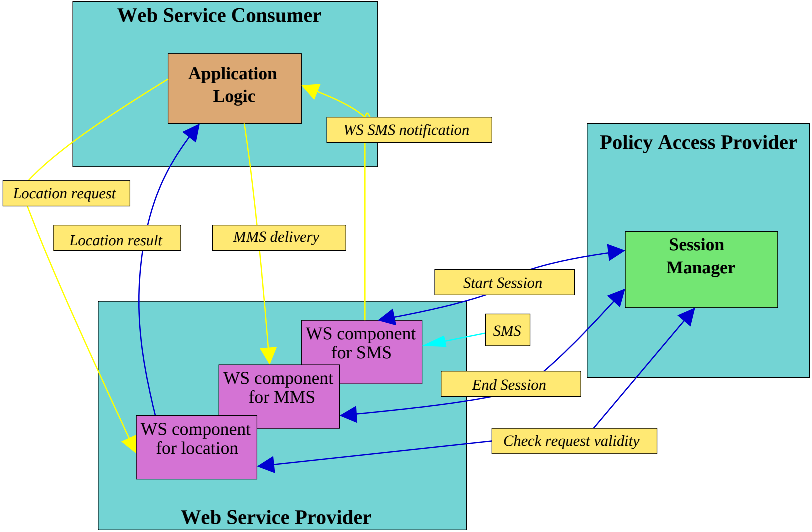
Schematic view



Schematic view



Schematic view



The PEPA model

The PEPA model of the system consists of four components:

- The user;
- The web service provider;
- The web service consumer, and
- The policy access provider.

The Web Service Provider consists of three distinct elements but the web service consumer is associated with a session which accesses each element in sequence.

Concurrency is introduced into the model by allowing multiple sessions rather than by representing the constituent web services separately.

The PEPA model

The PEPA model of the system consists of four components:

- The user;
- The web service provider;
- The web service consumer, and
- The policy access provider.

The Web Service Provider consists of three distinct elements but the web service consumer is associated with a session which accesses each element in sequence.

Concurrency is introduced into the model by allowing multiple sessions rather than by representing the constituent web services separately.

The PEPA model

The PEPA model of the system consists of four components:

- The user;
- The web service provider;
- The web service consumer, and
- The policy access provider.

The Web Service Provider consists of three distinct elements but the web service consumer is associated with a session which accesses each element in sequence.

Concurrency is introduced into the model by allowing multiple sessions rather than by representing the constituent web services separately.

The PEPA model

The PEPA model of the system consists of four components:

- The user;
- The web service provider;
- The web service consumer, and
- The policy access provider.

The Web Service Provider consists of three distinct elements but the web service consumer is associated with a session which accesses each element in sequence.

Concurrency is introduced into the model by allowing multiple sessions rather than by representing the constituent web services separately.

Component *Customer*

The customer's behaviour is simply modelled with two local states.

$$
\begin{aligned}
\mathit{Customer} &\stackrel{\mathit{def}}{=} (\mathit{getSMS}, r_1).\mathit{Customer}_1 \\
\mathit{Customer}_1 &\stackrel{\mathit{def}}{=} (\mathit{getMap}, \top).\mathit{Customer} \\
&\quad + (\mathit{get404}, \top).\mathit{Customer}
\end{aligned}
$$

We associate the user-perceived system performance with the throughput of the *getMap* action which can be calculated directly from the steady state probability distribution of the underlying Markov chain.

Component *Customer*

The customer's behaviour is simply modelled with two local states.

$$
\begin{aligned}
Customer &\stackrel{def}{=} (getSMS, r_1).Customer_1 \\
Customer_1 &\stackrel{def}{=} (getMap, \top).Customer \\
&+ (get404, \top).Customer
\end{aligned}
$$

We associate the user-perceived system performance with the throughput of the *getMap* action which can be calculated directly from the steady state probability distribution of the underlying Markov chain.

Component *Customer*

The customer's behaviour is simply modelled with two local states.

$$
\begin{aligned}
\mathit{Customer} &\stackrel{\mathit{def}}{=} (\mathit{getSMS}, r_1).\mathit{Customer}_1 \\
\mathit{Customer}_1 &\stackrel{\mathit{def}}{=} (\mathit{getMap}, \top).\mathit{Customer} \\
&\quad + (\mathit{get404}, \top).\mathit{Customer}
\end{aligned}
$$

We associate the user-perceived system performance with the throughput of the *getMap* action which can be calculated directly from the steady state probability distribution of the underlying Markov chain.

Component *WSConsumer*

Once a session has been started, it initiates a request for the user's current location and waits for a response.

For valid requests, location is returned and used to compute the appropriate map, which is then sent via an MMS message, using the web service.

$$
\begin{aligned}
WSConsumer &\stackrel{def}{=} (notify, \top).WSConsumer_2 \\
WSConsumer_2 &\stackrel{def}{=} (locReq, r_4).WSConsumer_3 \\
WSConsumer_3 &\stackrel{def}{=} (locRes, \top).WSConsumer_4 \\
&+ (locErr, \top).WSConsumer \\
WSConsumer_4 &\stackrel{def}{=} (compute, r_7).WSConsumer_5 \\
WSConsumer_5 &\stackrel{def}{=} (sendMMS, r_9).WSConsumer
\end{aligned}
$$

Component *WSConsumer*

Once a session has been started, it initiates a request for the user's current location and waits for a response.

For valid requests, location is returned and used to compute the appropriate map, which is then sent via an MMS message, using the web service.

$$
\begin{aligned}
WSConsumer &\stackrel{\mathit{def}}{=} (notify, \top).WSConsumer_2 \\
WSConsumer_2 &\stackrel{\mathit{def}}{=} (locReq, r_4).WSConsumer_3 \\
WSConsumer_3 &\stackrel{\mathit{def}}{=} (locRes, \top).WSConsumer_4 \\
&\quad + (locErr, \top).WSConsumer \\
WSConsumer_4 &\stackrel{\mathit{def}}{=} (compute, r_7).WSConsumer_5 \\
WSConsumer_5 &\stackrel{\mathit{def}}{=} (sendMMS, r_9).WSConsumer
\end{aligned}
$$

Component *WSConsumer*

Once a session has been started, it initiates a request for the user's current location and waits for a response.

For valid requests, location is returned and used to compute the appropriate map, which is then sent via an MMS message, using the web service.

$$
\begin{aligned}
WSConsumer &\stackrel{\mathit{def}}{=} (notify, \top).WSConsumer_2 \\
WSConsumer_2 &\stackrel{\mathit{def}}{=} (locReq, r_4).WSConsumer_3 \\
WSConsumer_3 &\stackrel{\mathit{def}}{=} (locRes, \top).WSConsumer_4 \\
&\quad + (locErr, \top).WSConsumer \\
WSConsumer_4 &\stackrel{\mathit{def}}{=} (compute, r_7).WSConsumer_5 \\
WSConsumer_5 &\stackrel{\mathit{def}}{=} (sendMMS, r_9).WSConsumer
\end{aligned}
$$

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

We assume that there is a distinct instance of the component *WSProvider* for each distinct session.

The *checkValid* action is represented twice, to capture the two possible distinct outcomes of the action.

- If the check is successful the location must be returned to the Web Service Consumer in the form of a map (*getMap*).
- If the check revealed an invalid request (*locErr*) then an error must be returned to the Web Service Consumer (*get404*) and the session terminated (*stopSession*).

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

We assume that there is a distinct instance of the component *WSProvider* for each distinct session.

The *checkValid* action is represented twice, to capture the two possible distinct outcomes of the action.

- If the check is successful the location must be returned to the Web Service Consumer in the form of a map (*getMap*).
- If the check revealed an invalid request (*locErr*) then an error must be returned to the Web Service Consumer (*get404*) and the session terminated (*stopSession*).

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

We assume that there is a distinct instance of the component *WSProvider* for each distinct session.

The *checkValid* action is represented twice, to capture the two possible distinct outcomes of the action.

- If the check is successful the location must be returned to the Web Service Consumer in the form of a map (*getMap*).
- If the check revealed an invalid request (*locErr*) then an error must be returned to the Web Service Consumer (*get404*) and the session terminated (*stopSession*).

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

We assume that there is a distinct instance of the component *WSProvider* for each distinct session.

The *checkValid* action is represented twice, to capture the two possible distinct outcomes of the action.

- If the check is successful the location must be returned to the Web Service Consumer in the form of a map (*getMap*).
- If the check revealed an invalid request (*locErr*) then an error must be returned to the Web Service Consumer (*get404*) and the session terminated (*stopSession*).

Component *WSProvider*

The use of sessions restricts a user's access to the services of the Web Service Provider to be sequential.

We assume that there is a distinct instance of the component *WSProvider* for each distinct session.

The *checkValid* action is represented twice, to capture the two possible distinct outcomes of the action.

- If the check is successful the location must be returned to the Web Service Consumer in the form of a map (*getMap*).
- If the check revealed an invalid request (*locErr*) then an error must be returned to the Web Service Consumer (*get404*) and the session terminated (*stopSession*).

Component *WSProvider*

$$
\begin{aligned}
WSProvider &\stackrel{def}{=} (getSMS, \top).WSProvider_2 \\
WSProvider_2 &\stackrel{def}{=} (startSession, r_2).WSProvider_3 \\
WSProvider_3 &\stackrel{def}{=} (notify, r_3).WSProvider_4 \\
WSProvider_4 &\stackrel{def}{=} (locReq, \top).WSProvider_5 \\
WSProvider_5 &\stackrel{def}{=} (checkValid, 99 \cdot \top).WSProvider_6 \\
&\quad + (checkValid, \top).WSProvider_{10}
\end{aligned}
$$

Component *WSProvider* cont.

$$
\begin{aligned}
WSProvider_6 &\stackrel{def}{=} (locRes, r_6).WSProvider_7 \\
WSProvider_7 &\stackrel{def}{=} (sendMMS, \top).WSProvider_8 \\
WSProvider_8 &\stackrel{def}{=} (getMap, r_8).WSProvider_9 \\
WSProvider_9 &\stackrel{def}{=} (stopSession, r_2).WSProvider \\
WSProvider_{10} &\stackrel{def}{=} (locErr, r_6).WSProvider_{11} \\
WSProvider_{11} &\stackrel{def}{=} (get404, r_8).WSProvider_9
\end{aligned}
$$

Component *PAProvider*

We consider a stateless implementation of the policy access provider.

$$
\begin{aligned}
\mathit{PAProvider} &\stackrel{\mathit{def}}{=} (\mathit{startSession}, \top).\mathit{PAProvider} \\
&+ (\mathit{checkValid}, r_5).\mathit{PAProvider} \\
&+ (\mathit{stopSession}, \top).\mathit{PAProvider}
\end{aligned}
$$

Model Component *WSComp*

The complete system is composed of some number of instances of the components interacting on their shared activities:

$$\begin{aligned} WSComp \stackrel{def}{=} & ((Customer[N_C] \underset{L_1}{\bowtie} WSProvider[N_{WSP}]) \\ & \underset{L_2}{\bowtie} WSConsumer[N_{WSC}]) \\ & \underset{L_3}{\bowtie} PAProvider[N_{PAP}] \end{aligned}$$

where the cooperation sets are

$$\begin{aligned} L_1 &= \{getSMS, getMap, get404\} \\ L_2 &= \{notify, locReq, locRes, locErr, sendMMS\} \\ L_3 &= \{startSession, checkValid, stopSession\} \end{aligned}$$

Parameter Values

param.	*value*	*explanation*
r_1	0.0010	rate customers request maps
r_2	0.5	rate session can be started
r_3	0.1	notification exchange between consumer and provider
r_4	0.1	rate requests for location can be satisfied
r_5	0.05	rate the provider can check the validity of the request
r_6	0.1	rate location information can be returned to consumer
r_7	0.05	rate maps can be generated
r_8	0.02	rate MMS messages can be sent from provider to customer
r_9	$10.0 * r_8$	rate MMS messages can be sent via the Web Service

Steady State Analysis for System Tuning

- Suppose that we want to design the system in such a way that it can handle 30 independent customers.
- Some parameters such as the network delays may be constrained by the available technology.
- However, there are a number of degrees of freedom which let us vary, for example, the number of threads of control of the components of the system.
- The aim of the analysis is to deliver a satisfactory service in a cost-effective way.
- The simplest example of a cost function may be a linearly dependency on the number of copies of a component or the rate at which an activity is performed.

Steady State Analysis for System Tuning

- Suppose that we want to design the system in such a way that it can handle 30 independent customers.
- Some parameters such as the network delays may be constrained by the available technology.
- However, there are a number of degrees of freedom which let us vary, for example, the number of threads of control of the components of the system.
- The aim of the analysis is to deliver a satisfactory service in a cost-effective way.
- The simplest example of a cost function may be a linearly dependency on the number of copies of a component or the rate at which an activity is performed.

Steady State Analysis for System Tuning

- Suppose that we want to design the system in such a way that it can handle 30 independent customers.
- Some parameters such as the network delays may be constrained by the available technology.
- However, there are a number of degrees of freedom which let us vary, for example, the number of threads of control of the components of the system.
- The aim of the analysis is to deliver a satisfactory service in a cost-effective way.
- The simplest example of a cost function may be a linearly dependency on the number of copies of a component or the rate at which an activity is performed.

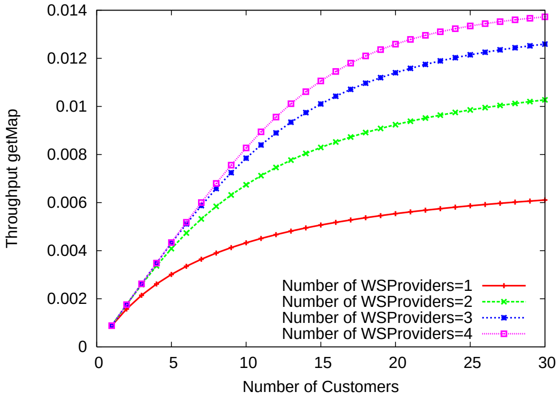
Steady State Analysis for System Tuning

- Suppose that we want to design the system in such a way that it can handle 30 independent customers.
- Some parameters such as the network delays may be constrained by the available technology.
- However, there are a number of degrees of freedom which let us vary, for example, the number of threads of control of the components of the system.
- The aim of the analysis is to deliver a satisfactory service in a cost-effective way.
- The simplest example of a cost function may be a linearly dependency on the number of copies of a component or the rate at which an activity is performed.

Steady State Analysis for System Tuning

- Suppose that we want to design the system in such a way that it can handle 30 independent customers.
- Some parameters such as the network delays may be constrained by the available technology.
- However, there are a number of degrees of freedom which let us vary, for example, the number of threads of control of the components of the system.
- The aim of the analysis is to deliver a satisfactory service in a cost-effective way.
- The simplest example of a cost function may be a linearly dependency on the number of copies of a component or the rate at which an activity is performed.

Throughput of the *getMap* action

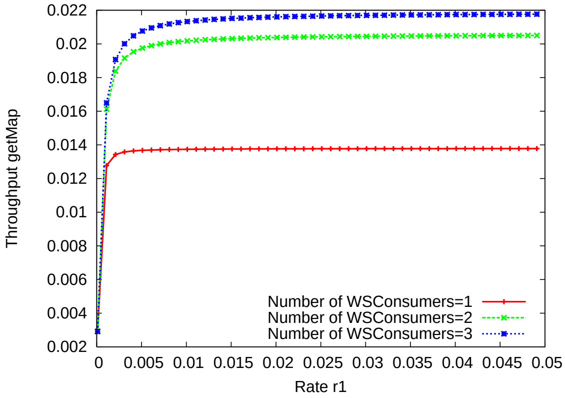


As the number of customers varies between 1 and 30 for various numbers of copies of the *WSProvider* component.

Throughput of the *getMap* action

- Under heavy load increasing the number of providers initially leads to a sharp increase in the throughput. However the gain deteriorates so that the system with four copies is just 8.7% faster than the system with three.
- In the following we settle on three copies of *WSProvider*.

Throughput of *getMap* action



As the request arrival rate (r_1) varies for differing numbers of *WSConsumer*.

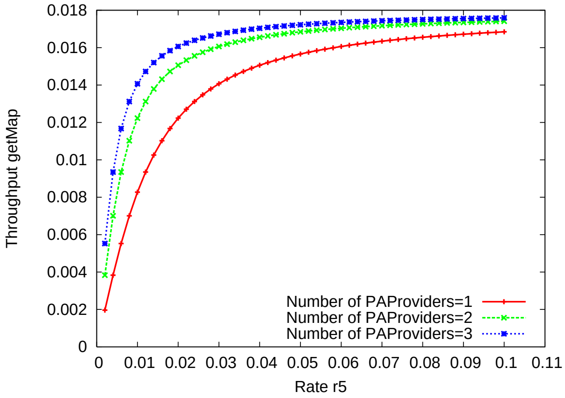
Throughput of *getMap* action

- Every line starts to plateau at approximately $r_1 = 0.010$ following an initial sharp increase. This suggests that the user is the bottle next in the system when the arrival rate is lower. Conversely, at high rates the system becomes congested.
- Whilst having two copies of *WSConsumer*, corresponding to two operating threads of control, improves performance significantly, the subsequent increase with three copies is less pronounced.
- So we set the number of copies of *WSConsumer* to 2.

Optimising the number of copies of *PAProvider*

- Here we are particularly interested in the overall impact of the rate at which the validity check is performed.
- Slower rates may mean more computationally expensive validation.
- Faster rates may involve less accuracy and lower security of the system.

Throughput of *getMap* action



As the validity check rate (r_5) varies for differing numbers of *PAProvider*.

Throughput of *getMap* action

- A sharp increase followed by a constant levelling off suggests that optimal rate values lie on the left of the plateau, as faster rates do not improve the system considerably.
- As for the optimal number of copies of *PAProvider*, deploying two copies rather than one dramatically increases the quality of service of the overall system.
- With a similar approach as previously discussed, the modeller may want to consider the trade-off between the cost of adding a third copy and the throughput increase.

An alternative design for *PAProvider*

- The original design of *PAProvider* is stateless.
- Any of its services can be called at any point, the correctness of the system being guaranteed by implementation-specific constraints such as session identifiers being uniquely assigned to the clients and passed as parameters of the method calls.
- Alternatively we may consider a stateful implementation, modelled as a sequential component with three local states.
- This implementation has the consequence that there can never be more than N_{PAP} *WSProvider* which have started a session with a *PAProvider*

An alternative design for *PAProvider*

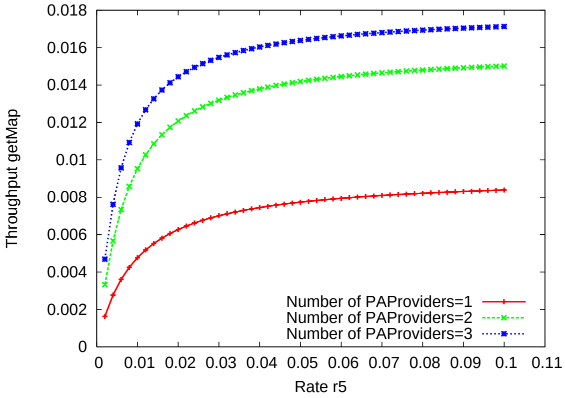
- The original design of *PAProvider* is stateless.
- Any of its services can be called at any point, the correctness of the system being guaranteed by implementation-specific constraints such as session identifiers being uniquely assigned to the clients and passed as parameters of the method calls.
- Alternatively we may consider a stateful implementation, modelled as a sequential component with three local states.
- This implementation has the consequence that there can never be more than N_{PAP} *WSProvider* which have started a session with a *PAProvider*

Component *PAProvider* — Stateful Version

It maintains a thread for each session and carries out the validity check on behalf of the Web Service Provider.

$$
\begin{aligned}
\mathit{PAProvider} &\stackrel{\mathit{def}}{=} (\mathit{startSession}, \top).\mathit{PAProvider}_2 \\
\mathit{PAProvider}_2 &\stackrel{\mathit{def}}{=} (\mathit{checkValid}, r_5).\mathit{PAProvider}_3 \\
\mathit{PAProvider}_3 &\stackrel{\mathit{def}}{=} (\mathit{stopSession}, \top).\mathit{PAProvider}
\end{aligned}
$$

Throughput of *getMap* action



As the validity check rate (r_5) varies for differing numbers of *PAProvider* (stateful version).

Throughput of *getMap* action

- In this case the incremental gain in adding more copies has become more marked.
- However, the modeller may want to prefer the original version, as three copies of the stateful provider deliver about as much as the throughput of only one copy of the stateless implementation.

Acknowledgements

The models of Roland the Gunslinger are due to Allan Clark and the model of the web service composition is joint work with Mirco Tribastone.

Outline

Querying models

So far we have focussed on the construction of the model and demonstrated the use of some particular examples to derive quantitative measures.

PEPA is complemented by a couple of formal approaches to query models.

- stochastic model checking based on CSL formulae; and
- (eXtended) stochastic probes within the PEPA model.

Querying models

So far we have focussed on the construction of the model and demonstrated the use of some particular examples to derive quantitative measures.

PEPA is complemented by a couple of formal approaches to query models.

- stochastic model checking based on CSL formulae; and
- (eXtended) stochastic probes within the PEPA model.

Model checking

Model checking requires two inputs:

- a description of the system, usually given in some high-level modelling formalism such as a process algebra description, or a Petri net;
- a specification of one or more desired properties of the system, normally using termporal logics such as CTL (Computational Tree Logic) or LTL (Linear-time Temporal Logic).

Model checking

Model checking requires two inputs:

- a description of the system, usually given in some high-level modelling formalism such as a process algebra description, or a Petri net;
- a specification of one or more desired properties of the system, normally using termporal logics such as CTL (Computational Tree Logic) or LTL (Linear-time Temporal Logic).

Model checking

From the high-level description the model checker constructs a labelled transition system which captures all possible behaviours of the system.

The model checking algorithms then automatically verify whether or not each property is satisfied in the system.

Model checking

From the high-level description the model checker constructs a labelled transition system which captures all possible behaviours of the system.

The model checking algorithms then automatically verify whether or not each property is satisfied in the system.

Stochastic model checking

In stochastic model checking it is assumed that the labelled transition system is a Continuous Time Markov Chain (CTMC).

This makes stochastic process algebras suitable high-level language for stochastic model checking.

The logic is also enhanced to query not just logical behaviour (whether some property is satisfied or not) but also quantified behaviour (e.g. the probability that a property is satisfied at a particular time).

Stochastic model checking

In stochastic model checking it is assumed that the labelled transition system is a Continuous Time Markov Chain (CTMC).

This makes stochastic process algebras suitable high-level language for stochastic model checking.

The logic is also enhanced to query not just logical behaviour (whether some property is satisfied or not) but also quantified behaviour (e.g. the probability that a property is satisfied at a particular time).

Stochastic model checking

In stochastic model checking it is assumed that the labelled transition system is a Continuous Time Markov Chain (CTMC).

This makes stochastic process algebras suitable high-level language for stochastic model checking.

The logic is also enhanced to query not just logical behaviour (whether some property is satisfied or not) but also quantified behaviour (e.g. the probability that a property is satisfied at a particular time).

Model checking

There are two broad approaches to model checking:

- Explicit state model checking (exhaustive exploration for all possible states/executions): exact results obtained via numerical computation.
- Statistical model-checking (discrete event simulation and sampling over multiple runs): approximate results.

PRISM model and model checking

- Probabilistic model checking in PRISM is based on a CTMC and the logic CSL.
- Formally the mapping from PEPA is based on the structured operational semantics, generating the underlying CTMC in the usual way.
- In practice PEPA is an input language for PRISM with a direct mapping between PEPA components and the interacting, reactive modules of the PRISM input language.
- Note, however, that this places a restriction to have synchronisations in which only one participant is active as PRISM cannot handle the apparent rate based calculations of cooperation in PEPA.

PRISM model and model checking

- Probabilistic model checking in PRISM is based on a CTMC and the logic CSL.
- Formally the mapping from PEPA is based on the structured operational semantics, generating the underlying CTMC in the usual way.
- In practice PEPA is an input language for PRISM with a direct mapping between PEPA components and the interacting, reactive modules of the PRISM input language.
- Note, however, that this places a restriction to have synchronisations in which only one participant is active as PRISM cannot handle the apparent rate based calculations of cooperation in PEPA.

PRISM model and model checking

- Probabilistic model checking in PRISM is based on a CTMC and the logic CSL.
- Formally the mapping from PEPA is based on the structured operational semantics, generating the underlying CTMC in the usual way.
- In practice PEPA is an input language for PRISM with a direct mapping between PEPA components and the interacting, reactive modules of the PRISM input language.
- Note, however, that this places a restriction to have synchronisations in which only one participant is active as PRISM cannot handle the apparent rate based calculations of cooperation in PEPA.

PRISM model and model checking

- Probabilistic model checking in PRISM is based on a CTMC and the logic CSL.
- Formally the mapping from PEPA is based on the structured operational semantics, generating the underlying CTMC in the usual way.
- In practice PEPA is an input language for PRISM with a direct mapping between PEPA components and the interacting, reactive modules of the PRISM input language.
- Note, however, that this places a restriction to have synchronisations in which only one participant is active as PRISM cannot handle the apparent rate based calculations of cooperation in PEPA.

The CSL logic

The syntax of CSL is as follows:

$$\begin{aligned}\phi \;::=\;& \texttt{true} \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \\ & \mathbf{P}_{\sim p}[\phi \; \mathbf{U}^{I} \; \phi] \mid \mathbf{S}_{\sim p}[\phi] \mid \\ & \mathbf{R}_{\sim r}[I^{=t}] \mid \mathbf{R}_{\sim r}[C^{\leq t}] \mid \mathbf{R}_{\sim r}[\mathbf{F}\; \phi] \mid \mathbf{R}_{\sim r}[\mathbf{S}]\end{aligned}$$

where a is an atomic proposition, $\sim \in \{<, \leq, \geq, >\}$, $p \in [0, 1]$, I is an interval of $\mathbb{R}^{\geq 0}$ and $r, t \in \mathbb{R}^{\geq 0}$.

P and **S** are probabilistic operators which include a probabilistic bound $\sim p$.

R is a reward operator with a reward bound $\sim r$.

Probabilistic operators

A formula $\mathbf{P}_{\sim p}[\phi\ \mathbf{U}^I\ \phi]$ is true in a state s if the probability of the formula $(\phi\ \mathbf{U}^I\ \phi)$ being satisfied from state s meets the bound $\sim p$.

A formula of type $\phi_1\ \mathbf{U}^I\ \phi_2$ is an until formula.

It is true of a path σ through the state space if, for some time instant $t \in I$, at time t in the path σ the CSL subformula ϕ_2 is true and the subformula ϕ_1 is true at all preceding time instants.

A formula $\mathbf{S}_{\sim p}[\phi]$ is true in state s if the probability that the formula ϕ being satisfied in a steady state reached from state s meets the bound $\sim p$.

Probabilistic operators

A formula $\mathbf{P}_{\sim p}[\phi \; \mathbf{U}^I \; \phi]$ is true in a state s if the probability of the formula $(\phi \; \mathbf{U}^I \; \phi)$ being satisfied from state s meets the bound $\sim p$.

A formula of type $\phi_1 \; \mathbf{U}^I \; \phi_2$ is an until formula.

It is true of a path σ through the state space if, for some time instant $t \in I$, at time t in the path σ the CSL subformula ϕ_2 is true and the subformula ϕ_1 is true at all preceding time instants.

A formula $\mathbf{S}_{\sim p}[\phi]$ is true in state s if the probability that the formula ϕ being satisfied in a steady state reached from state s meets the bound $\sim p$.

Probabilistic operators

A formula $\mathbf{P}_{\sim p}[\phi \; \mathbf{U}^{I} \; \phi]$ is true in a state s if the probability of the formula $(\phi \; \mathbf{U}^{I} \; \phi)$ being satisfied from state s meets the bound $\sim p$.

A formula of type $\phi_1 \; \mathbf{U}^{I} \; \phi_2$ is an until formula.

It is true of a path σ through the state space if, for some time instant $t \in I$, at time t in the path σ the CSL subformula ϕ_2 is true and the subformula ϕ_1 is true at all preceding time instants.

A formula $\mathbf{S}_{\sim p}[\phi]$ is true in state s if the probability that the formula ϕ being satisfied in a steady state reached from state s meets the bound $\sim p$.

The CSL Reward operator

The CSL reward operator **R** is used to express properties concerning the expected value of rewards.

$\mathbf{R}_{\sim r}[I^{=t}]$ asserts that the expected value of the state reward at time instant t meets the bound $\sim r$.

$\mathbf{R}_{\sim r}[C^{\leq t}]$ refers to the expected reward accumulated up until t.

$\mathbf{R}_{\sim r}[\mathbf{F}\ \phi]$ asserts that the expected reward accumulated before a state satisfying ϕ is reached meets the bound $\sim r$.

$\mathbf{R}_{\sim r}[\mathbf{S}]$ asserts that the long-run/steady state expected reward meets the bound $\sim r$.

The CSL Reward operator

The CSL reward operator **R** is used to express properties concerning the expected value of rewards.

$\mathbf{R}_{\sim r}[I^{=t}]$ asserts that the expected value of the state reward at time instant t meets the bound $\sim r$.

$\mathbf{R}_{\sim r}[C^{\leq t}]$ refers to the expected reward accumulated up until t.

$\mathbf{R}_{\sim r}[\mathbf{F}\ \phi]$ asserts that the expected reward accumulated before a state satisfying ϕ is reached meets the bound $\sim r$.

$\mathbf{R}_{\sim r}[\mathbf{S}]$ asserts that the long-run/steady state expected reward meets the bound $\sim r$.

The CSL Reward operator

The CSL reward operator **R** is used to express properties concerning the expected value of rewards.

$\mathbf{R}_{\sim r}[I^{=t}]$ asserts that the expected value of the state reward at time instant t meets the bound $\sim r$.

$\mathbf{R}_{\sim r}[C^{\leq t}]$ refers to the expected reward accumulated up until t.

$\mathbf{R}_{\sim r}[\mathbf{F}\ \phi]$ asserts that the expected reward accumulated before a state satisfying ϕ is reached meets the bound $\sim r$.

$\mathbf{R}_{\sim r}[\mathbf{S}]$ asserts that the long-run/steady state expected reward meets the bound $\sim r$.

The CSL Reward operator

The CSL reward operator **R** is used to express properties concerning the expected value of rewards.

$\mathbf{R}_{\sim r}[I^{=t}]$ asserts that the expected value of the state reward at time instant t meets the bound $\sim r$.

$\mathbf{R}_{\sim r}[C^{\leq t}]$ refers to the expected reward accumulated up until t.

$\mathbf{R}_{\sim r}[\mathbf{F}\ \phi]$ asserts that the expected reward accumulated before a state satisfying ϕ is reached meets the bound $\sim r$.

$\mathbf{R}_{\sim r}[\mathbf{S}]$ asserts that the long-run/steady state expected reward meets the bound $\sim r$.

The CSL Reward operator

The CSL reward operator **R** is used to express properties concerning the expected value of rewards.

$\mathbf{R}_{\sim r}[I^{=t}]$ asserts that the expected value of the state reward at time instant t meets the bound $\sim r$.

$\mathbf{R}_{\sim r}[C^{\leq t}]$ refers to the expected reward accumulated up until t.

$\mathbf{R}_{\sim r}[\mathbf{F}\ \phi]$ asserts that the expected reward accumulated before a state satisfying ϕ is reached meets the bound $\sim r$.

$\mathbf{R}_{\sim r}[\mathbf{S}]$ asserts that the long-run/steady state expected reward meets the bound $\sim r$.

Computation in PRISM

The underlying computation in PRISM for explicit state model checking involves a combination of:

graph-theoretical algorithms, for conventional temporal logic model checking and qualitative probabilistic model checking;

numerical computation, for quantitative probabilistic model checking, i.e. calculation of probabilities and reward values.

Statistical model checking

The basic idea of statistical model checking is to simulate the system for finitely many runs, and use statistics to infer whether the samples provide evidence for the satisfaction or violation of the property of interest.

Advantages of statistical model checking

- Much larger models can be handled since the state space does not need to be constructed and stored all at once.
- Since the approach is based on observations and samples it can be applied to any system which is executable — the underlying stochastic process does not need to be a CTMC.
- Since many independent samples are required it is susceptible to coarse-grained parallelization.

Advantages of statistical model checking

- Much larger models can be handled since the state space does not need to be constructed and stored all at once.
- Since the approach is based on observations and samples it can be applied to any system which is executable — the underlying stochastic process does not need to be a CTMC.
- Since many independent samples are required it is susceptible to coarse-grained parallelization.

Advantages of statistical model checking

- Much larger models can be handled since the state space does not need to be constructed and stored all at once.
- Since the approach is based on observations and samples it can be applied to any system which is executable — the underlying stochastic process does not need to be a CTMC.
- Since many independent samples are required it is susceptible to coarse-grained parallelization.

Advantages of statistical model checking

- Much larger models can be handled since the state space does not need to be constructed and stored all at once.
- Since the approach is based on observations and samples it can be applied to any system which is executable — the underlying stochastic process does not need to be a CTMC.
- Since many independent samples are required it is susceptible to coarse-grained parallelization.

These advantages are off-set by the disadvantage that is it an approximation compared with the exact, explicit state approach.

General framework

Consider a CTMC $\mathcal{X}$ and a property ϕ.

An execution or run of $\mathcal{X}$ is a, possibly infinite, sequence of states in $\mathcal{X}$.

We wish to decide $\mathbf{P}_{\sim p}[\phi]$, i.e. whether $\mathcal{X}$ satisfies ϕ with probability satisfying the bound $\sim p$.

The result of each execution is taken to be the result of a Bernoulli trial, 0 or 1, according to whether ϕ is satisfied or not.

Let q be the probability that ϕ is satisfied, then we seek to establish if $q \sim p$.

General framework

Consider a CTMC $\mathcal{X}$ and a property ϕ.

An execution or run of $\mathcal{X}$ is a, possibly infinite, sequence of states in $\mathcal{X}$.

We wish to decide $\mathbf{P}_{\sim p}[\phi]$, i.e. whether $\mathcal{X}$ satisfies ϕ with probability satisfying the bound $\sim p$.

The result of each execution is taken to be the result of a Bernoulli trial, 0 or 1, according to whether ϕ is satisfied or not.

Let q be the probability that ϕ is satisfied, then we seek to establish if $q \sim p$.

General framework

Consider a CTMC $\mathcal{X}$ and a property ϕ.

An execution or run of $\mathcal{X}$ is a, possibly infinite, sequence of states in $\mathcal{X}$.

We wish to decide $\mathbf{P}_{\sim p}[\phi]$, i.e. whether $\mathcal{X}$ satisfies ϕ with probability satisfying the bound $\sim p$.

The result of each execution is taken to be the result of a Bernoulli trial, 0 or 1, according to whether ϕ is satisfied or not.

Let q be the probability that ϕ is satisfied, then we seek to establish if $q \sim p$.

General framework

Consider a CTMC $\mathcal{X}$ and a property ϕ.

An execution or run of $\mathcal{X}$ is a, possibly infinite, sequence of states in $\mathcal{X}$.

We wish to decide $\mathbf{P}_{\sim p}[\phi]$, i.e. whether $\mathcal{X}$ satisfies ϕ with probability satisfying the bound $\sim p$.

The result of each execution is taken to be the result of a Bernoulli trial, 0 or 1, according to whether ϕ is satisfied or not.

Let q be the probability that ϕ is satisfied, then we seek to establish if $q \sim p$.

General framework

Consider a CTMC $\mathcal{X}$ and a property ϕ.

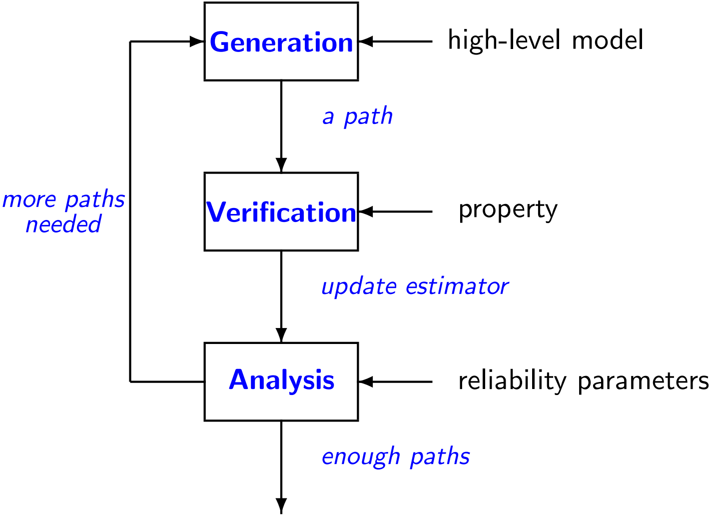
An execution or run of $\mathcal{X}$ is a, possibly infinite, sequence of states in $\mathcal{X}$.

We wish to decide $\mathbf{P}_{\sim p}[\phi]$, i.e. whether $\mathcal{X}$ satisfies ϕ with probability satisfying the bound $\sim p$.

The result of each execution is taken to be the result of a Bernoulli trial, 0 or 1, according to whether ϕ is satisfied or not.

Let q be the probability that ϕ is satisfied, then we seek to establish if $q \sim p$.

Schematic for statistical model checking



References

- A. Clark, S. Gilmore, J. Hillston, and M. Tribastone, chapter *Stochastic Process Algebras*, in Formal Methods for Performance Evaluation: 7th Intl. School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, LNCS 4486, pages 132-179. Springer-Verlag, Bertinoro, Italy, May-June 2007.
- `http://www.dcs.ed.ac.uk/pepa/`
- `http://www.prismmodelchecker.org/`