

# Markov Regenerative Process - steady-state analysis

MVT exam

Stefano MARTINA

[stefano.martina@stud.unifi.it](mailto:stefano.martina@stud.unifi.it)

Tommaso PAPINI

[tommaso.papini1@stud.unifi.it](mailto:tommaso.papini1@stud.unifi.it)



Università degli Studi di Firenze

October 7, 2015



This work is licensed under a [Creative Commons Attribution-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-sa/4.0/).

# Markov Regenerative Processes (MRP)

## Definition

A Markov Regenerative Process (MRP) is a stochastic process that sooner or later, with probability one, will reach a **regenerative** state (will be regenerated).

## Regenerative state

A state where the process loses its memory.

# Markov Regenerative Processes (MRP)

## Definition

A Markov Regenerative Process (MRP) is a stochastic process that sooner or later, with probability one, will reach a **regenerative** state (will be regenerated).

## Regenerative state

A state where the process loses its memory.

# Markov Regenerative Processes (MRP)

## Definition

A Markov Regenerative Process (MRP) is a stochastic process that sooner or later, with probability one, will reach a **regenerative** state (will be regenerated).

## Regenerative state

A state where the process loses its memory.



CTMC

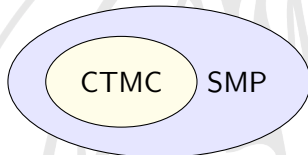
# Markov Regenerative Processes (MRP)

## Definition

A Markov Regenerative Process (MRP) is a stochastic process that sooner or later, with probability one, will reach a **regenerative** state (will be regenerated).

## Regenerative state

A state where the process loses its memory.



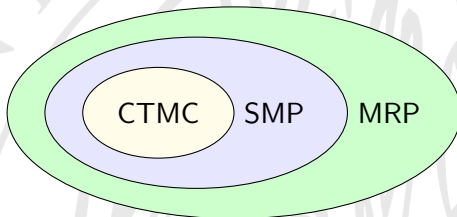
# Markov Regenerative Processes (MRP)

## Definition

A Markov Regenerative Process (MRP) is a stochastic process that sooner or later, with probability one, will reach a **regenerative** state (will be regenerated).

## Regenerative state

A state where the process loses its memory.



# The steady-state problem

## Transient probabilities

The probability distribution that the process will be in a certain state, after given  $t$  time.

## Steady-state

For ergodic systems, it represents the probability distribution that the process will be in a certain state, as time goes to infinity.

- ✓ ORIS current state:
  - Transient analysis for Markov Regenerative Process (MRP)
  - Steady-state analysis for everything else
- ✓ Until now! ☹
- ✓ **Warning:** we assume that the MRP is ergodic!

# The steady-state problem

## Transient probabilities

The probability distribution that the process will be in a certain state, after given  $t$  time.

## Steady-state

For ergodic systems, it represents the probability distribution that the process will be in a certain state, as time goes to infinity.

- ✓ ORIS current state:
  - Transient analysis for Markov Regenerative Process (MRP)
  - Steady-state analysis for everything else
- ✓ Until now! ☺
- ✓ **Warning:** we assume that the MRP is ergodic!



# The steady-state problem

## Transient probabilities

The probability distribution that the process will be in a certain state, after given  $t$  time.

## Steady-state

For ergodic systems, it represents the probability distribution that the process will be in a certain state, as time goes to infinity.

- ✓ ORIS current state:
  - Transient analysis for Markov Regenerative Process (MRP)
  - Steady-state analysis for everything else
- ✓ Until now! ☺
- ✓ **Warning:** we assume that the MRP is ergodic!

# The steady-state problem

## Transient probabilities

The probability distribution that the process will be in a certain state, after given  $t$  time.

## Steady-state

For ergodic systems, it represents the probability distribution that the process will be in a certain state, as time goes to infinity.

- ✓ ORIS current state:
  - Transient analysis for Markov Regenerative Process (MRP)
  - Steady-state analysis for everything else
- ✓ Until now! ☺
- ✓ **Warning:** we assume that the MRP is ergodic!

# The steady-state problem

## Transient probabilities

The probability distribution that the process will be in a certain state, after given  $t$  time.

## Steady-state

For ergodic systems, it represents the probability distribution that the process will be in a certain state, as time goes to infinity.

- ✓ ORIS current state:
  - Transient analysis for Markov Regenerative Process (MRP)
  - Steady-state analysis for everything else
- ✓ Until now! ☺
- ✓ **Warning:** we assume that the MRP is ergodic!

# The steady-state problem

## Transient probabilities

The probability distribution that the process will be in a certain state, after given  $t$  time.

## Steady-state

For ergodic systems, it represents the probability distribution that the process will be in a certain state, as time goes to infinity.

- ✓ ORIS current state:
  - Transient analysis for Markov Regenerative Process (MRP)
  - Steady-state analysis for everything else
- ✓ Until now! ☺
- ✓ **Warning:** we assume that the MRP is ergodic!

# MRP steady-state analysis - The theory

General idea:

1. Calculate the embedded DTMC steady-state on the regenerative states
2. Calculate the sojourn times in each marking, after each regenerative state
3. Combine the two above in order to calculate the MRP steady-state

Embedded DTMC  
steady-state

Sojourn times

MRP steady-state

```
graph TD; A[Embedded DTMC steady-state] --> C[MRP steady-state]; B[Sojourn times] --> C;
```

The diagram illustrates the process of calculating the MRP steady-state. It features three boxes: a light blue box labeled 'Embedded DTMC steady-state', a light yellow box labeled 'Sojourn times', and a light green box labeled 'MRP steady-state'. Arrows from the first two boxes point towards the third box, indicating that the MRP steady-state is derived from these two components.

# MRP steady-state analysis - The theory

General idea:

1. Calculate the embedded DTMC steady-state on the regenerative states
2. Calculate the sojourn times in each marking, after each regenerative state
3. Combine the two above in order to calculate the MRP steady-state

Embedded DTMC  
steady-state

Sojourn times

MRP steady-state

# MRP steady-state analysis - The theory

General idea:

1. Calculate the embedded DTMC steady-state on the regenerative states
2. Calculate the sojourn times in each marking, after each regenerative state
3. Combine the two above in order to calculate the MRP steady-state

Embedded DTMC  
steady-state

Sojourn times

MRP steady-state

```
graph TD; A[Embedded DTMC steady-state] --> C[MRP steady-state]; B[Sojourn times] --> C;
```

The diagram illustrates the process of calculating the MRP steady-state. It features three boxes: a purple box labeled 'Embedded DTMC steady-state', a yellow box labeled 'Sojourn times', and a green box labeled 'MRP steady-state'. Arrows from the purple and yellow boxes point towards the green box, indicating that the MRP steady-state is derived from these two components.

# MRP steady-state analysis - The theory

General idea:

1. Calculate the embedded DTMC steady-state on the regenerative states
2. Calculate the sojourn times in each marking, after each regenerative state
3. Combine the two above in order to calculate the MRP steady-state

Embedded DTMC  
steady-state

Sojourn times

MRP steady-state

```
graph TD; A[Embedded DTMC steady-state] --> C[MRP steady-state]; B[Sojourn times] --> C;
```



# Main classes implemented

- ✓ `class` EmbeddedDTMC
  - written from scratch
  - calculate embedded DTMC steady-state
- ✓ `class` RegenerativeSteadyStateAnalysis
  - based on `class` RegenerativeTransientAnalysis
  - calculate MRP steady-state

# Steady-state of the embedded DTMC on regenerative states

## Steady-state in Discrete Time Markov Process (DTMC)

If the steady-state of a Discrete Time Markov Process (DTMC) exists and is unique (if is ergodic), then it's calculated by solving for  $v$  the linear system:

$$\begin{cases} v = vP \\ |v| = 1 \end{cases}$$

- ✓ We want to calculate the steady-state of the embedded DTMC of the MRP in the regenerative states
- ✓ But we don't have  $P$ ! ☹

# Steady-state of the embedded DTMC on regenerative states

## Steady-state in Discrete Time Markov Process (DTMC)

If the steady-state of a Discrete Time Markov Process (DTMC) exists and is unique (if is ergodic), then it's calculated by solving for  $v$  the linear system:

$$\begin{cases} v = vP \\ |v| = 1 \end{cases}$$

- ✓ We want to calculate the steady-state of the embedded DTMC of the MRP in the regenerative states
- ✓ But we don't have  $P$ ! ☹

# Steady-state of the embedded DTMC on regenerative states

## Steady-state in Discrete Time Markov Process (DTMC)

If the steady-state of a Discrete Time Markov Process (DTMC) exists and is unique (if is ergodic), then it's calculated by solving for  $v$  the linear system:

$$\begin{cases} v = vP \\ |v| = 1 \end{cases}$$

- ✓ We want to calculate the steady-state of the embedded DTMC of the MRP in the regenerative states
- ✓ But we don't have  $P$ ! ☹

## Reaching probability feature

- ✓ We add a new **reaching probability feature** to each state:  
`class ReachingProbabilityFeature`
- ✓ Inside `SteadyStateInitialStateBuilder`: set it to 1
- ✓ Inside `SteadyStatePostProcessor`: multiply the parent's reaching probability by the probability to chose a certain child

If we run a transient analysis on the Petri Net (PN) we get:

- ✓ `regenerationClasses`
- ✓ `Map<DeterministicEnablingState, Map<DeterministicEnablingState, Set<State>>>`
- ✓ sum reaching probability feature of each State to compute elements of **P**

Now we can solve the linear system:

$$\begin{cases} v = vP \\ |v| = 1 \end{cases} = \begin{cases} (P' - I)v' = 0 \\ \sum_i v_i = 1 \end{cases}$$

✓ RealMatrix & RealVector

✓ QR decomposition solver

- DecompositionSolver solver = **new** QRDecomposition(coefficients).getSolver();
- RealVector steadyState = solver.solve(constants);

✓ Convert steadyState into a

Map<DeterministicEnablingState, BigDecimal>

## Sojourn time $a_{ij}$

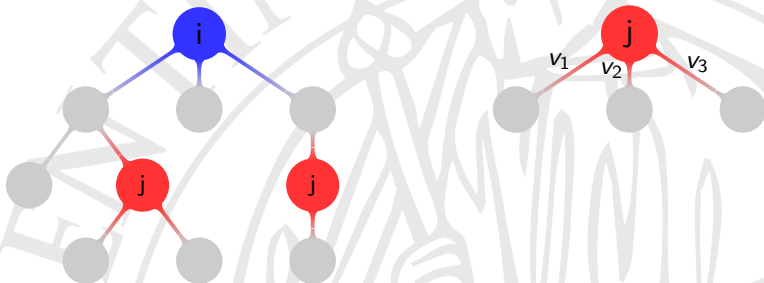
### Definition

The sojourn time  $a_{ij}$  represents the average time spent in the  $j$ -th marking after the (last)  $i$ -th regeneration.

# How to compute $a_{ij}$ ?

$a_{ij}$  is:

- ✓ sum of avg time spent in marking  $j$  occurrences
  - sum of avg time before each variable fires
    - ★ condition each variable to be the minimum (i.e. the one that fires)
    - ★ compute avg time before that variable fires (thanks Marco!)





# When to compute $a_{ij}$ ?

During the transient analysis!

- ✓ transient analysis generates succession trees for each regenerative state
  - regenerative state as root
  - following regenerative states as leaves
  - reachable markings as inner nodes
- ✓ during the tree generation compute and accumulate  $a_{ij}$  for each marking occurrence found

# Markov Regenerative Process (MRP) steady-state

Let's combine the embedded DTMC steady-state and the sojourn times!

$$\pi_j = \frac{\sum_i v_i a_{ij}}{K}$$

- ✓ We multiply the sojourn time in the marking  $j$  after the regeneration  $i$  by the probability of reaching the  $i$ -th regeneration
- ✓ We do this for each regeneration that leads to the marking  $j$  before another regeneration
- ✓  $K$  is a normalization factor calculated as the sum of  $\pi_j$

# Markov Regenerative Process (MRP) steady-state

Let's combine the embedded DTMC steady-state and the sojourn times!

$$\pi_j = \frac{\sum_i v_i a_{ij}}{K}$$

- ✓ We multiply the sojourn time in the marking  $j$  after the regeneration  $i$  by the probability of reaching the  $i$ -th regeneration
- ✓ We do this for each regeneration that leads to the marking  $j$  before another regeneration
- ✓  $K$  is a normalization factor calculated as the sum of  $\pi_j$

# Markov Regenerative Process (MRP) steady-state

Let's combine the embedded DTMC steady-state and the sojourn times!

$$\pi_j = \frac{\sum_i v_i a_{ij}}{K}$$

- ✓ We multiply the sojourn time in the marking  $j$  after the regeneration  $i$  by the probability of reaching the  $i$ -th regeneration
- ✓ We do this for each regeneration that leads to the marking  $j$  before another regeneration
- ✓  $K$  is a normalization factor calculated as the sum of  $\pi_j$

# Markov Regenerative Process (MRP) steady-state

Let's combine the embedded DTMC steady-state and the sojourn times!

$$\pi_j = \frac{\sum_i v_i a_{ij}}{K}$$

- ✓ We multiply the sojourn time in the marking  $j$  after the regeneration  $i$  by the probability of reaching the  $i$ -th regeneration
- ✓ We do this for each regeneration that leads to the marking  $j$  before another regeneration
- ✓  $K$  is a normalization factor calculated as the sum of  $\pi_j$

# Markov Regenerative Process (MRP) steady-state

Let's combine the embedded DTMC steady-state and the sojourn times!

$$\pi_j = \frac{\sum_i v_i a_{ij}}{K}$$

- ✓ We multiply the sojourn time in the marking  $j$  after the regeneration  $i$  by the probability of reaching the  $i$ -th regeneration
- ✓ We do this for each regeneration that leads to the marking  $j$  before another regeneration
- ✓  $K$  is a normalization factor calculated as the sum of  $\pi_j$

# Test

## Unit test

- ✓ Class `SteadyStateTest` with **JUnit** tests
- ✓ Three different models:
  - `TestCaseSMP`
  - `TestCase2ParallelTasks`
  - `TestCaseRejuvenation`
- ✓ For each test:
  1. launch MRP steady state **analysis**
  2. check if the result is comparable to the **expected** value (with a tolerance)

# Test

## Unit test

- ✓ Class `SteadyStateTest` with **JUnit** tests
- ✓ Three different models:
  - `TestCaseSMP`
  - `TestCase2ParallelTasks`
  - `TestCaseRejuvenation`
- ✓ For each test:
  1. launch MRP steady state **analysis**
  2. check if the result is comparable to the **expected** value (with a tolerance)



# Test

## Unit test

- ✓ Class `SteadyStateTest` with **JUnit** tests
- ✓ Three different models:
  - `TestCaseSMP`
  - `TestCase2ParallelTasks`
  - `TestCaseRejuvenation`
- ✓ For each test:
  1. launch MRP steady state **analysis**
  2. check if the result is comparable to the **expected** value (with a tolerance)

# Test

## Unit test

- ✓ Class `SteadyStateTest` with **JUnit** tests
- ✓ Three different models:
  - `TestCaseSMP`
  - `TestCase2ParallelTasks`
  - `TestCaseRejuvenation`
- ✓ For each test:
  1. launch MRP steady state **analysis**
  2. check if the result is comparable to the **expected** value (with a tolerance)

# Test

## Unit test

- ✓ Class `SteadyStateTest` with **JUnit** tests
- ✓ Three different models:
  - `TestCaseSMP`
  - `TestCase2ParallelTasks`
  - `TestCaseRejuvenation`
- ✓ For each test:
  1. launch MRP steady state **analysis**
  2. check if the result is comparable to the **expected** value (with a tolerance)

# Test

## Unit test

- ✓ Class `SteadyStateTest` with **JUnit** tests
- ✓ Three different models:
  - `TestCaseSMP`
  - `TestCase2ParallelTasks`
  - `TestCaseRejuvenation`
- ✓ For each test:
  1. launch MRP steady state **analysis**
  2. check if the result is comparable to the **expected** value (with a tolerance)

# Test

## Unit test

- ✓ Class SteadyStateTest with JUnit tests
- ✓ Three different models:
  - TestCaseSMP
  - TestCase2ParallelTasks
  - TestCaseRejuvenation
- ✓ For each test:
  1. launch MRP steady state analysis
  2. check if the result is comparable to the expected value (with a tolerance)

# Test

## Unit test

- ✓ Class SteadyStateTest with JUnit tests
- ✓ Three different models:
  - TestCaseSMP
  - TestCase2ParallelTasks
  - TestCaseRejuvenation
- ✓ For each test:
  1. launch MRP steady state analysis
  2. check if the result is comparable to the expected value (with a tolerance)

# Test

## Unit test

- ✓ Class SteadyStateTest with JUnit tests
- ✓ Three different models:
  - TestCaseSMP
  - TestCase2ParallelTasks
  - TestCaseRejuvenation
- ✓ For each test:
  1. launch MRP steady state analysis
  2. check if the result is comparable to the expected value (with a tolerance)