Computational Models For Complex Systems (6 cfu) 23-24

Notes

University of Pisa

M.Sc. in Computer Science

Contents

Chapter 1

Introduction

1.1 What is a Model of a System?

A model is a simplified and approximate representation of a system, that allows reasoning on the systems properties. **So we construct models in order to understand some aspect of that system**. We include in the model only the aspect of the system that we consider essential, omitting details that would only complicate the analysis.

1.1.1 Understanding the model

After designing our model, then in order to acquire new knowledge, we have to **analyze the model** (like a child play with a toy to understand what it does), then we have to apply some **reasoning** regarding the result given during the analysis.

1.1.2 Errors

All the three steps listed above (Modelling, Analysis, and Reasoning), **can be sources of errors**. The model can be too abstract; the analysis can be too inaccurate; or the interpretation itself could be wrong. **Recall that the result that you get are showing you only something about the real system, you need to generalise what the result means.**

1.1.3 Examples of Models

Planimetries/project and scale models

In architecture planimetries and scale models are used to assest structural properties at design time, in order to evaluate the result in advance.

Life Science

In biology rats (in-vivo model) or a cell colture (in-vitro model) are used as model for the human being.

In Silico Models

In biology there are also computer-based techniques are usually faster and cheaper than in vivo e in vitro models. They simulate the interaction between proteins.

1.1.4 Mathematical Models

Mathematics provides tools for building abstract models of almost everything like **geometry** for Architecture or **differential equations** for Weather. Mathematical models have two advantages:

- They are formal model specification languages, meaning it is not an ambiguous model.
- there are a lot of analytical and numerical methods to analyze model of this type.

1.1.5 Computational Models

Computational Models is similar to Mathematical Models. A Computational Model is a mathematical representation of a dynamical system (systems which evolves over time) taking a computer-executable form.

in Computational Model we are restricting the class of systems of interest to dynamical systems (systems which evolve over time).

Dynamical Systems

Dynamical systems are systems that **evolve** by changing their state over time. Typically the state of a dynamical system is represented by a finite set of variables called **state variables**. In addition we will have some some law/rules/equations that describe how the state variables will vary over time. Our focus will be in prediction or analyze how the state of a dynamic system changes.

There are different types of dynamical systems depending on how the values of the variables change in either a discrete or a continuous way (or both).

Note: time can be interpreted as a discrete or continuous entity.

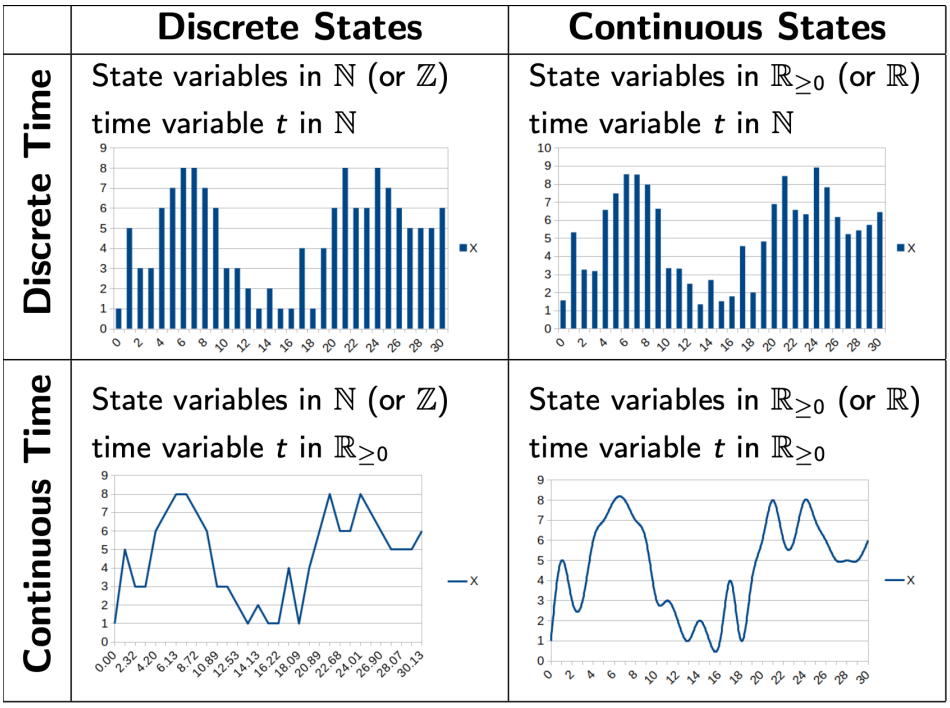


Figure 1.1: All possible types of dynamical systems.

1.1.6 Case of application

What kind of analysis could we do with models of dynamical systems?

- **Reachability of states:** predict the future state of the dynamical system.
- **Behavioral patterns:** the sequence of state that I pass through over time.
- **Effects of perturbations/ control strategies:** once I have a model of a system and I can simulate it, I can try to study how it will behave if I modify it.

1.1.7 How to build a computational models?

There are multiple ways

The Data-driven way

The Data-driven way (e.g. machine learning) consist in starting from the data of the system you want to model and then you apply some machine learning/optimization/whatever method to infer the model automatically. The generated model takes a form suitable for the inference method used. If enough data in available, it often works very well (good predictions), however inferred models are often very difficult to

be interpreted (meaning that we can do good prediction, but we can't explain why they are correct).

The Knowledge-driven way

(also called mechanistic models) consist in trying to reproduce through a mathematical model the internal mechanism of the system in order to reproduce the behaviour and to understand the internals of the systems. It requires limited data, but a good knowledge about the system functioning. Model construction usually requires some effort, and often predictions suffer from approximations but the method generated works also when few data are available; the model is interpretable: it contributes to understanding why a system behaves as observed; modelling allows validation hypotheses on the system functioning.

1.2 What is a complex system?

A complex system is a system consisting of many components (typically with a simple individual behaviours) interacting with each other, from these interaction emerges the global behaviour of the system.

Complex Networks

Complex Networks is a graph with complex structural properties. The dynamics of these networks and its evolution is a field of study (complex networks theory).

1.2.1 Modeling notations for complex systems

Many modeling languages are available for complex systems:

- **mathematics:** Recurrence relations and differential equations.
- **concurrency theory:** we can apply methods seen in the study of concurrent system like Petri nets. Rewrite rules (Multiset rewriting) that describe the different events that happens in a complex systems as rules.
- **artificial life:** approaches proposed to try to reproduce behaviour seen in life. Cellular automata that describe the population as a grid; agents based model in which you explicit as an agent (e.g. an algorithm or set of functions/procedures) and then you can put the agent in a virtual environment to see how they behave together.

1.3 Analyze the model

Modeling languages allow the modeler to express relationships between the state variables of a system and the rules/laws that determine the change of their values over time. The dynamics (or behaviour) of the systems (the actual sequences of states reached by the system over time) can be computed according to the semantics of the modeling language.

1.4 Analyze the behaviour

After the model has been specified, then there are multiple way to analyze the model.

1.4.1 Simulation

You can try to apply some simulation algorithm in order to try to execute that model to have a possible evolution of the system. If the system is deterministic you have only one possible evolution, instead if the system is stochastic-probabilistic you will repeat the simulation several time and get different behaviour of the system. **So simulations can give you only some possible behaviour, not all**. Instead of running simulations to construct some description of the overall behaviour of the systems, it can be done by using **transition systems**.

Transition system

A transition system is a graph (possible infinite) that describe the possible behaviour of a system. A possible transition system is the *Makov chain*

1.4.2 Model Checking

It is an approach that determine whether the whole transition system satisfies a given dynamical property.

1.5 Modeling vs programming

The approach that we will follow to analyze dynamical systems is similar to the approach used to analyze programs

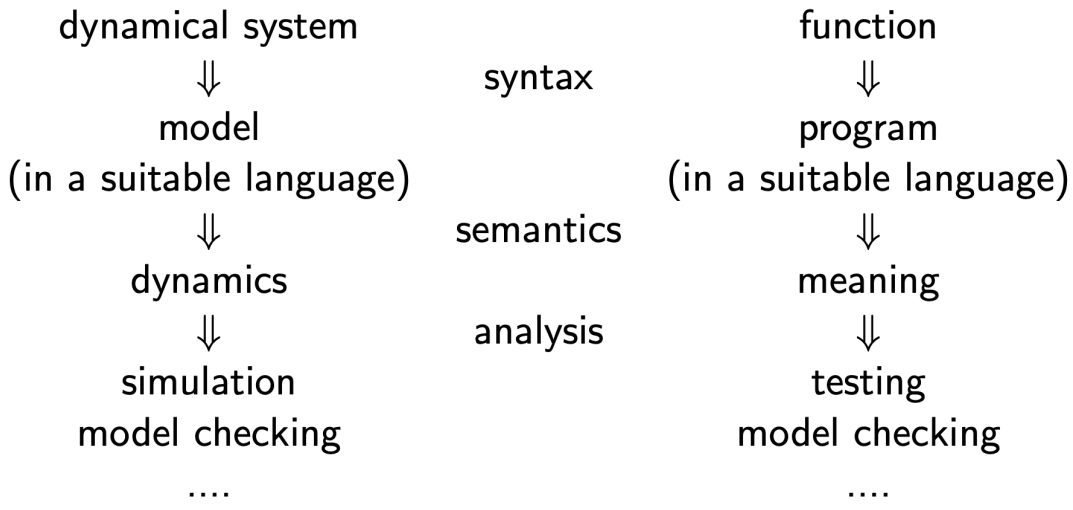


Figure 1.2: Modeling vs programming.

Chapter 2

Discrete Dynamical Systems

They are systems where the evolution is performed using discrete steps, in which the variables describing the state of the system are updated.

2.1 Recurrence relations (Difference equations)

Relations that tell you how from the current state of the discrete systems you obtain the next state of the systems, that is how from the original values the go to the new ones.

In this schema we will consider a generic system that is considered as a **population**(birth/death of individuals). Even the simplest form of interaction between individuals can lead to the emergence of **complex behaviors** (chaotic behavior) in the population.

2.2 Linear Birth Model

The linear birth model is the simplest model describing a recurrence relation.

Given a population, each individual is **indistinguishable** from each other. We denote with `N(t)` the **density of some population** at a time `t`, that is the variable that describe the number of individuals that are part of the population at time `t`.

Given this description we want to predict what will happen to the density of the same population at a future time $t^{'} = t + \Delta t$ assuming that:

- all individuals are indistinguishable from each other.
- there is enough food and space for every individual.
- each individual has λ children every σ time units.

- there is no death occurring in the interval $[t, t + \Delta t)$.
- children do not start reproducing in the interval $[t, t + \Delta t)$.

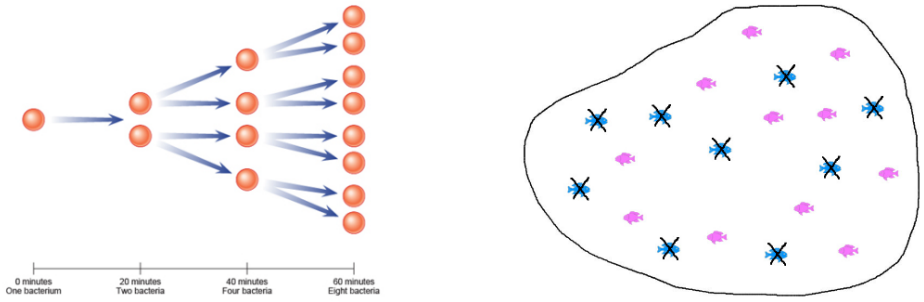


Figure 2.1: Example of populations satisfying the assumption. In the bacteria example in order to assume the no children duplication we set $\Delta t \leq 20 minutes$

2.2.1 Describing the birth process as a relation

The idea is that the size of the population at time $N(t + \Delta t)$ will be $\geq$ than $N(t)$ since we assume **that there is no death occurring in the interval** $[t, t + \Delta t]$. Additionally we know from our assumption that each *adult* individual has λ children every σ time units.

Then, the number of individuals at time $t + \Delta t$ corresponds to the number of individuals at time t, plus the number of newborns at time Δt:

$$N(t + \Delta t) = N(t) + \lambda \frac{\Delta t}{\sigma} N(t)$$

if we group for $N(t)$ then the equation can be rewritten as follows:

$$N(t + \Delta t) = N(t) * (1 + \lambda \frac{\Delta t}{\sigma})$$

Where $\frac{\Delta t}{\sigma}$ **describes the birth moments for every *adult* individual in the interval** $[t, t + \Delta t]$**.**

2.3 Constructing our algorithm

Defined our equation **we can derive a discrete model from it**.

We choose a **time step** (discretization step) that describe an update of the population, in our case **the time necessary for a newborns to be considered an adult so that it can reproduce**, and we set it as Δt.

Instead of representing the equation fully we rewrite it using the **notation of sequence theory** by instead of using the actual value of the time Δt, we just count the steps. Thus we obtain:

$$N_{t+1} = r_d N_t$$

where r stands for *rate*; d stands for *discrete*; r_d represents the **constant birth rate** s.t. $r_d = \lambda \frac{\Delta t}{\sigma}$.

2.3.1 Defining the general term of our equation

Given the recurrence relation we can *sometimes* calculate the **general term**, that is the solution of the recurrence relation. In the case of our linear birth model it is a **non-recursive definition of** N_t.

To do so we first calculate the first terms of N_t: $N_1, N_2, N_3, ...$

$N_1 = r_d N_0$

$N_2 = r_d N_1 = r_d^2 N_0$

$N_3 = r_d N_2 = r_d^3 N_0$

We notice how *it seems* that $N_t = r_d^t N_0$, **but we have to prove it**. To prove the formula, since we are in the realms of the natural numbers, we can use **mathematical induction**:

Base Case (t = 0): $N_0 = r_d^0 N_0$ which is true.

Induction Case (t = k + 1): we assume that the formula is correct for $t = 0$ to $t = k$ and check if it is true for $t = k+1$. We know from assumption that N_{k+1} will be a summation between the previous value for $t = k$ and the newborns in the current step. $N_{k+1} = r_d Nk$ and we know from **induction hypothesis** that $N_k = r_d^k N_0$, thus we can rewrite N_{k+1} as $N_{k+1} = r_d(r_d^k N_0) = r_d^{k+1} N_0$ proving the thesis.

Now having the general term $N_t = r_d^t N_0$ we can compute the solution of our model.

Phase portrait

A way to represent recurrence relations graphically is through **phase portrait**. It consists in putting in the *Cartesian plane*:

- **x-axis**: N_t
- **y-axis**: N_{t+1}

Then in the plane we plot the recurrent relation. First you draw the bisector and you draw the recurrence relation as a function, then by starting from point (N_0, N_0) on the bisector, the other point can be obtained by *"bouncing"* on the curve of the recurrence relation. **You can see how fast the quantity increases (or decreases) over time**.

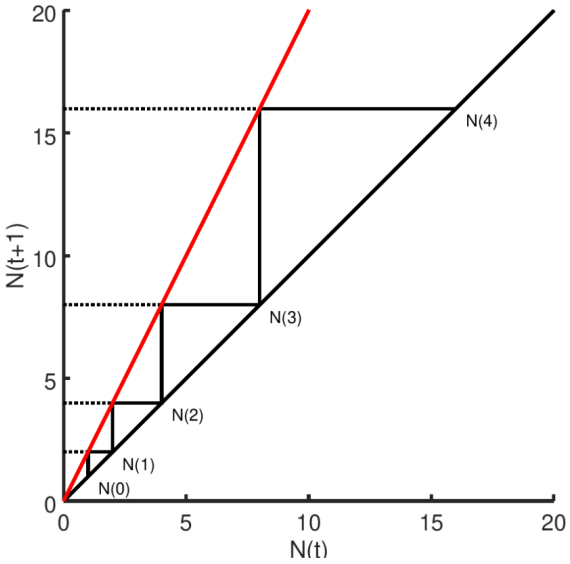


Figure 2.2:

Above an example of phase portrait in our linear birth model.
In red we draw the recurrence equation $N_{t+1} = 2N_t$. in **black** the bisector $N_{t+1} = N_t$.

2.4 Introducing death in our model

We complicate our recurrence relation by considering also deaths, so we are adding a negative term that decrease the number of individuals of our population over time.

Assume that, a constant fraction s_d **of adults that die in every time step** δt. Then our recurrence relation now is:

$$N_{t+1} = r_d N_t - s_d N_t$$

By grouping for N_t we can rewrite the equation as:

$$N_{t+1} = (r_d - s_d) N_t$$

Note: since the number of individuals which die cannot be greater than the whole population, then $0 \leq s_d \leq 1$

Since r_d and s_d are two constant, one positive and one negative, we can group them together in one single constant $\alpha_d = (r_d - s_d)$. We call α_d the **net growth rate**, that is the rate where you discard the individuals who dies. We can rewrite our equations as:

$$N_{t+1} = \alpha_d N_t$$

The difference in respect of the previous case is that α_d will be in general greater than 0, but not necessarily greater than 1.

General trend of our model

There could be many possible cases depending on the value of the constant α_d:

- $\alpha_d > 1$: the overall behaviour of the population is the same as before, **since the population at every steps will increase**.
- $\alpha_d = 1$: the population remains constant, **since the number of newborns always is equal to the number of dead ones**.
- $\alpha_d < 1$: if for instance we assume we have less newborns than the death of old individuals, **then at every step the population reduces**.

2.5 Introducing migration in our model

Independently from the size of the population, **we assume a fixed number of individuals arrive in our region**. We only consider people that arrives, so we model the migration using a constant and positive parameter declared as β.

Then our equation becomes:

$$N_{t+1} = \alpha_d N_t + \beta$$

with $\beta \geq 0$ representing the number of individuals entering the population every Δt time units.

By **mathematical induction** we can prove the general term is:

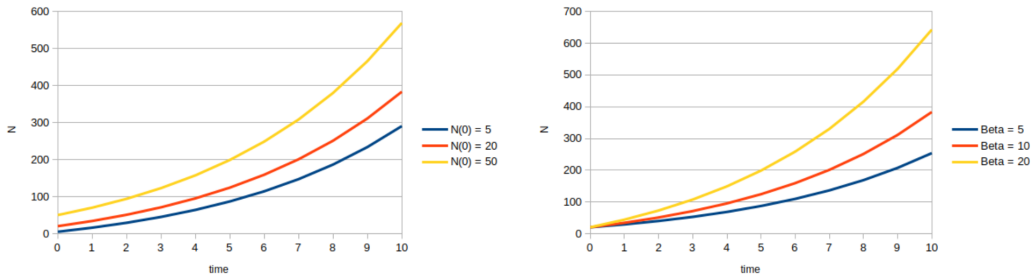
$$N_t = \alpha_d^t N_0 + \sum_{i=0}^{t-1} a_d^i \beta$$

where $\sum_{i=0}^{t-1} a_d^i \beta$ accumulates the β individuals that arrived in the previous step (from 0 to $t-1$) and the children produced by them at every step.

2.5.1 Simulating the migration

Having our general term, we can see what happens by varying α_d, N_0 and β :

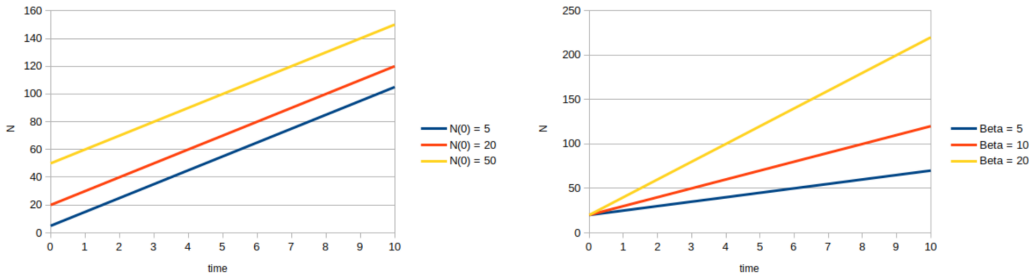
Case $\alpha_d > 1$



$N_0 = 5, 20, 50 \quad \beta = 10$ $\qquad$ $N_0 = 20 \quad \beta = 5, 10, 20$

Figure 2.3: The dynamics is dominated by the birth process (exponential growth).

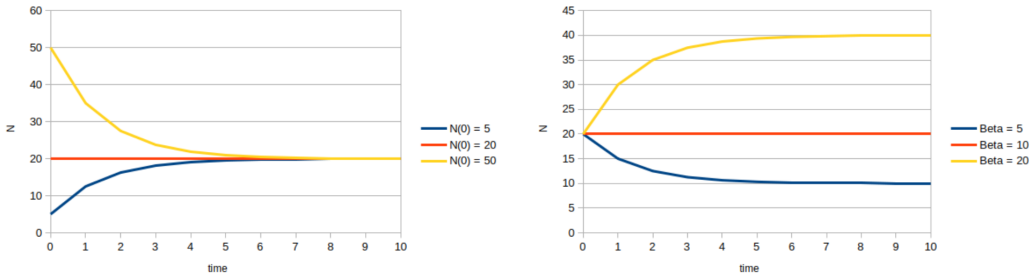
Case $\alpha_d = 1$



$N_0 = 5, 20, 50 \quad \beta = 10$ $\qquad$ $N_0 = 20 \quad \beta = 5, 10, 20$

Figure 2.4: The dynamics is dominated by the migration process (linear growth).

Case $\alpha_d < 1$



$N_0 = 5, 20, 50 \quad \beta = 10$ $N_0 = 20 \quad \beta = 5, 10, 20$

Figure 2.5: The population reaches a dynamic equilibrium: a stable state in which opposite phenomena compensate each other (migration compensates deaths). Note that it is independent from N_0

2.5.2 Computing the equilibrium point

The equilibrium point (also called saturation point) is defined as the moment $t = k+1$ where the computed size of the population is the same as the previous step, that is:

$$N_{t-1} = N_t$$

Knowing that $N_t = \alpha_d N_t + \beta$ we solve the equation in regards to N_t and obtain:

$$N_t = \frac{\beta}{1-\alpha_d}$$

2.5.3 Non-linear models

So far we have described a population where each individual behaves autonomously, which do not fit the definition of complex system (many components with very simple individual behaviour **that interact with each other and the environment**). Introducing interaction between the individuals **requires a non-linear model** (previously we used a linear one). In our (Non-linear) Birth Model now the environment has limited resources such as food and place, thus requiring that the individuals compete **(a form in interaction)** for survival.

2.6 Rewriting our equation

For simplicity assume we are in a closed environment (no migration), then we define K as the **carrying capacity of the environment**, meaning that in the environment there is enough food and space for K individuals.

The population is still governed by the birth rate, but now death is no more a constant, instead is negatively influenced by K:

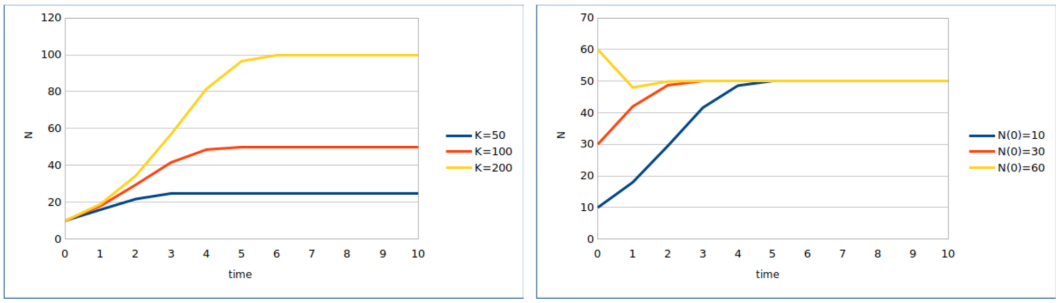
$$N_{t+1} = r_d N_t (1 - \frac{N_t}{K})$$

Note: this non-linear equation is called **logistic equation** (it is an alternative formulation)

So now the birth rate is modulated by the ratio of occupancy on the environment $\frac{N_t}{K}$:

- N_t close to 0 : we have a simple birth process with rate r_d (exponential growth).
- N_t increases: the growth tends to stop.

Eventually the population reaches a **dynamic equilibrium** representing the situation in which environment resources are fully exploited (**saturation**)



$N_0 = 10 \quad K = 50, 100, 200 \qquad N_0 = 10, 30, 60 \quad K = 100$

Figure 2.6: The first graph shows a case where the equilibrium point depends on K. The second graph shows me that the equilibrium point is independent from the initial number of individuals.

2.7 Removing homogeneity

For now we have considered an homogeneous population, but in realistic case we have different individuals with different features and we want to consider each group of individuals. In the system point of view this requires not just a single recurrence equation, but a **system of recurrence equations**.

consider a population of fishes that live in a pond. Each individual can either be a male fish, modelled as M_t or a female fish, modelled as F_t. We consider that a small part of males dies because of fights among them (death rate s_d).

then we construct a system of recurrence equations:

$$\begin{cases} F_{t+1} = r_d F_t (1 - \frac{F_t + M_t}{K}) \\ M_{t+1} = r_d F_t (1 - \frac{F_t + M_t}{K}) - s_d M_t \end{cases}$$

where:

- $r_d F_t$ represent the number of child born, note that they are generated by females.
- $F_t + M_t$ describes the whole population size (to be related with the carrying capacity K).

2.8 Limitation of discrete dynamical models

Discretization of the system dynamics may introduce inaccuracies: recurrence equations assume that nothing happens during the Δt time that occurs between N_t and N_{t+1}. Adjusting Δt to be smaller usually correspond to more accurate approximations, more precisely we should let Δt tend to 0.

Chapter 3

Continuous Dynamical Systems

3.1 Recap - Why we need to introduce ODEs

As we said in the previous chapter, using a discrete representation for the steps in our model can lead us to lose all the informations that happens between the step N_t and the step $N_t + 1$.

The simplest solution is to make the distance in time Δt between the to step very small (≈ 0), but it is not enough.

3.2 Reconsidering the population model

Recall in our population model that with $N(t)$ we denote the **density of some population** at time t. Our goal is to construct a mathematical model able to predict the density of the same population at a time $t^{'} = t + \Delta t$.

3.2.1 Introducing the Ordinary Differential Equation

Given our equation which is based the model:

$$N(t + \Delta t) = N(t) + \lambda \frac{\Delta t}{\sigma} N(t)$$

And the corresponding **recurrence equation**:

$$N_{t+1} = r_d N_t$$

where $r_d = \frac{\Delta t}{\sigma}$ consider the case where $\Delta t \to 0$. This case cannot be done using discretization, because it can lead to inaccuracies.

To handle the case $\Delta t \to 0$ we make some transformation to our equation:

$$N(t+\Delta t) = N(t) + \lambda \frac{\Delta t}{\sigma} N(t) \rightarrow \frac{N(t+\Delta t)-N(t)}{\Delta t}$$

then by simplifying we get:

$$\frac{N(t+\Delta t)-N(t)}{\Delta t} = \frac{\lambda}{\sigma} N(t)$$

We can recognize that the left hand side can be traced back as the **difference quotient** $\frac{f(x+h)-f(x)}{h}$, **which when taken to the limit as h approaches 0 gives the derivative of the function f**.

Let's consider our equation for $\Delta t \rightarrow 0$:

$$\lim_{\Delta t \to 0} \frac{N(t+\Delta t)-N(t)}{\Delta t} = \lim_{\Delta t \to 0} r_c N(t)$$

with $r_c = \frac{\lambda}{\sigma}$

Note that on the right hand side Δt doesn't appear, thus we can remove the limit; the term on the left hand side is the derivative of $N(t)$, indicated as simply $\dot{N}(t)$. Thus we simply the equation as follows:

$$\dot{N}(t) = r_c N(t)$$

We have defined the dynamics of the system as the derivative equal to a constant multiplied by the value of the function at time t.

This equation is known as Ordinary Differential Equation (ODE).

An ODE has the following properties:

- it relates the function N with its derivative $\dot{N}$
- $t \in \mathbb{R}$, **so time is continuous**

We would like to use the Ordinary Differential Equation to run some simulation or to compute a general solution (like we did for the recurrence relation).

3.2.2 Using the ODE

In some very simple case, like our linear birth model, we can find the general solution by finding a closed-form definition of $N(t)$ satisfying the equation. **A closed-form definition is one that depends only on t and some constant**.

for the linear growth model a solution can be found analytically.

Recall our equation:

$$\dot{N}(t) = r_c N(t)$$

Now let's move $N(t)$ from the right to the left hand-side of the equation:

$$\frac{\dot{N}(t)}{N(t)} = r_c$$

Recall that $\frac{\dot{N}(t)}{N(t)} = \ln N(t)$ and r_c is the derivative of $r_c t + c$ for any constant c, obtaining:

$$\ln N(t) = r_c t + c$$

By resolving our equation in regards to $N(t)$ we finally get:

$$N(t) = Ce^{r_c t + c}$$

where $C = e^c$ (typically C is set to be equal to $N(0)$

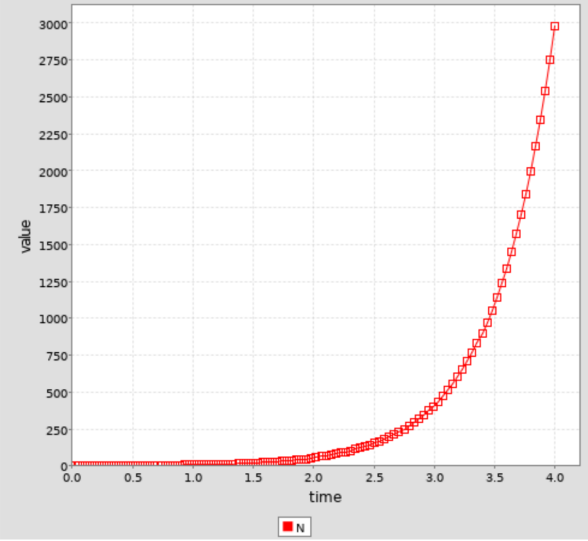


Figure 3.1: This graphs shows us that by setting in our population model $r_c = 2$ and $C = N(0) = 1$ then the population shows an exponential growth over time.

Differences between discrete and continuous model

This behaviour is qualitatively the same as for the discrete model, both showing exponential growth. **What changes is the meaning of the equation**:

- the recurrence relation tells you how to update the variable.
- in our new equation it defines the derivative, **meaning how fast it's changing**.

Note how both equations show rate ≥ 1 only if the birth rate $\frac{\lambda}{\sigma}$ is ≥ 1.

3.3 Example - Radioactive decay

We move from the example of the population model we have seen so far to another one.

Radioactive decay is a process where we have a negative evolution of the population. The idea is that **each molecule decays at a constant rate**, so the whole mass decreases with a rate which is proportional to the mass itself.

We can describe this model by using the following Ordinary Differential Equation:

$$\dot{N}(t) = -d_c N(t)$$

then if we solve it in regards to $N(t)$ we get:

$$N(t) = N(0)e^{-d_c t}$$

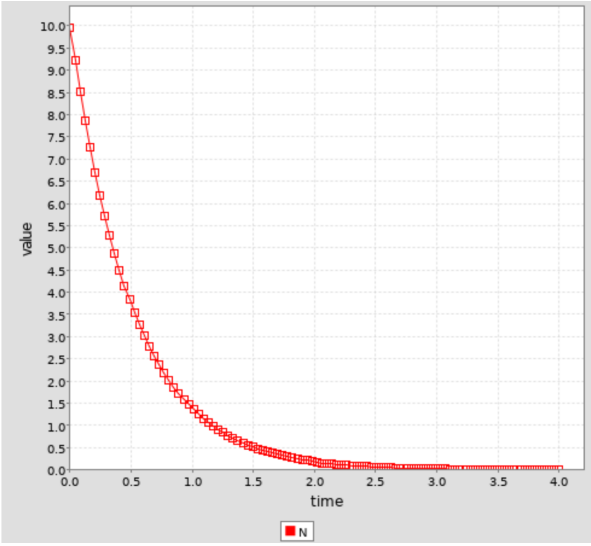


Figure 3.2: Example of applying the ODE of the radioactive decay. Note that by setting $d_c = 2$ and $C = N(0) = 10$ we get $N(t)$ to tend to zero. This is caused by the negative exponent in the ODE.

3.4 Continuous version of the logistic equation

Given the non-linear logistic equation, we can define as follows its continuous version:

$$\dot{N}(t) = r_c N(t)(1 - \tfrac{N(t)}{K})$$

where:

- r_c is the **continuous growth rate**.

- K is the **carrying capacity** of the environment.

Resolving the ODE in regards to $N(t)$ we get:

$$N(t) = \frac{K}{1+(\frac{K}{N(0)}-1)e^{-r_c t}}$$

We notice that when $N(t) \to K$ **the population converges to the carrying capacity of the environment**.

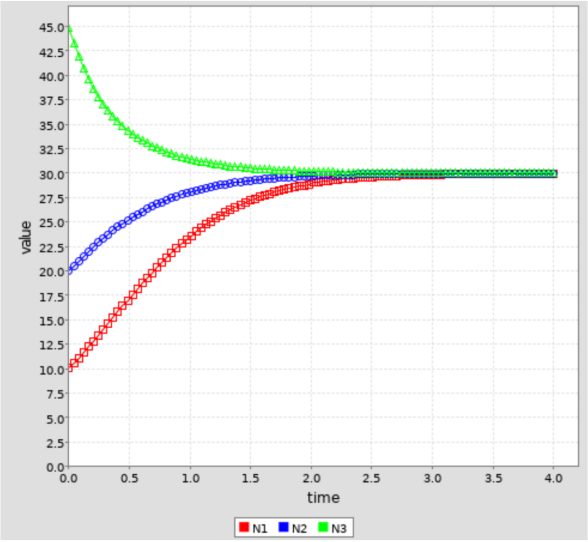


Figure 3.3: Example of the continuous logistic equation. Note how by putting $r_c = 2$, $N(0) = 10$, $K = 30$, the population will eventually converge to the carrying capacity of the environment.

The trend is the same as for the discrete case.

3.5 Systems of ODE

Now let's consider a population of males, indicated as $M(t)$, and females, indicated as $F(t)$. Assume that males fight with each other, so a small part of them dies because of it with a death rate of s_c.

Thus we have to expand our model considering a system of ODEs:

$$\begin{cases} \dot{F}_t = r_c F_t (1 - \dfrac{F_t + M_t}{K}) \\ \dot{M}_t = r_c F_t (1 - \dfrac{F_t + M_t}{K}) - s_c M_t \end{cases}$$

where:

- $r_c F(t)$ is used for both genders, **since both are generated by females**.

- $F(t) + M(t)$ describes the whole population size.

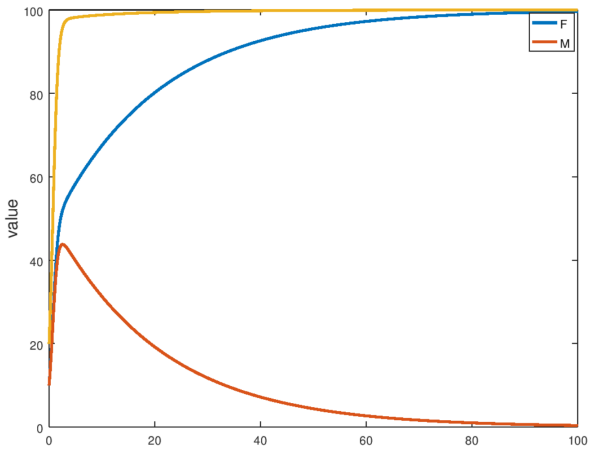


Figure 3.4: Example of the system of ODEs.

What happens in this scenario, after some times you have only females in the population. This shows a completely different behaviour from the discrete case, **because the meaning of their equations is different**:

- The recurrence relations indicates us the size of the population.
- The system of ODEs describes how fast is changing.

3.6 Numerical Solution of ODEs

Recall that we said that computing the solution of an ODE is not always possible. For this reason we prefer working using an approximation of the solution, indicated as **numerical solver** (or numerical simulator). This approximation doesn't compute the general function, **instead it solves the initial value problem, also called Cauchy problem**.

Initial Value Problem

Given an ODE in the form $\dot{N}(t) = f(N(t))$ and an initial value N_0 s.t. $N(0) = N_0$, compute a function $F(t)$ that is a solution of the ODE and s.t. $F(0) = N_0$.

In our case we are interested only in studying the values of $F(t)$ where $t \geq 0$, **hence we perform a numerical simulation starting at** $t = 0$.

3.7 The Euler method

The Euler method is the simplest numerical simulation method available. The main concept behind this approach is to approximate the given continuous system with a recurrence relation specifically designed to approximate the continuous differential equation. **The idea is that, since we know the derivative, we use it as it was the function**.

It is based on the idea of discretizing the dynamics of differential equations by time steps of constant length $\tau = \Delta t$.

Given an Ordinary Differential Equation in the form:

$$\dot{N}(t) = f(N(t))$$

this corresponds to approximating its solution with the following recurrence relation (assuming $N_0 = N(0)$)

$$N_{k+1} = N_k + \tau f(N_k)$$

where $N_k \approx N(\tau k)$

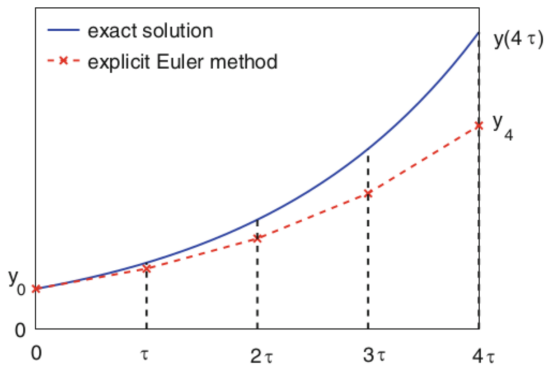


Figure 3.5: A graph showing the correct solution compared to the approximation generated by the Euler method.

3.7.1 Errors in the Euler method

By approximating in each step the Euler method makes local errors that will contribute to a global error at the end of the whole simulation.

The local discretization error is computed as $|N(\tau) - N_1|$ and it is in the order of $O(\tau^2)$ *(the motivation is because we truncate the Taylor Series at the first step).*

The Global discretization error is obtained by accumulate all the local discretization errors after k steps, namely at time $t = k\tau$. The global error is computes as $|N(k\tau) - N_k)$ and is in the order $O(k\tau^2) = O(\tau)$ since $k\tau = t$ is constant.

3.8 Other numerical simulation methods

A linear error of $O(\tau)$) is quite annoying, requiring us to set $\tau \approx 0$ in order to have something acceptable, making the computation very slow (it requires a lot of steps).

To overcome this other methods have been proposed, having a global discretization error of a higher order (e.g. $O(\tau^p)$ for some p) which is better as long as $\tau \to 0$ (hence $\tau < 1$). To work these method use more than one point to approximate the value of the function, making a single step require more time, but maintain the error inside some defined boundaries.

A few examples of such methods are:

- **Runge-Kutta methods:** $p = 2$ in the original formulation, but can be higher
- **Multistep methods (e.g. Adams methods):** extrapolate the value of the next step from the values of the previous k steps obtaining $p \approx k$.

State-of-art methods can also:

- **s**elf-determine the step size τ based on thresholds on local and global discretization errors.
- **d**ynamically adjust the step size τ during their executing (e.g. Adaptive Runge-Kutta)

3.9 Instability and stiff systems

There is another problem that could arise: if you have a derivative which cause the value of a variable in a very fast way, if τ is not small enough you not only pay some error but you could have that your computed solution is unstable.

What we mean for unstable is that we lose informations and your points start to oscillate creating chaos.

This kind of problematic systems are called **stiff systems**. There is no clear definition of stiffness: intuitively contains a very fast term that cannot be captured by our method, worse if we have in our systems also slow terms to take into account.

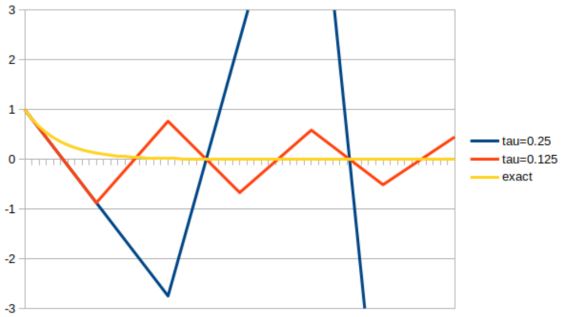


Figure 3.6: An example of a stiff system. In orange is shown the real solution, while in red and blue the one obtained by setting τ to different values

In this cases you need to consider alternative numerical simulation methods called **implicit numerical simulation methods**. All the methods we have seen so far have and implicit version, which requires more computation time for step.

3.10 Implicit Euler method

The idea of the implicit version of the Euler method is that, like the previous version we approximate the function by using its derivative, **but the derivative is not computed on the value N at step k, but at the step $k+1$.**

By using this approach we are no longer defining N_{k+1} in terms of N_k, but as an equation where N_{k+1} is in both sides:

$$N_{k+1} = N_k + \tau f(N_{k+1})$$

where $N_k \approx N(k\tau)$.

There are methods from numerical analysis that are able to solve these type of equations. This method requires more effort for computing a single step, but **often** the local discretization error is smaller permitting us to use greater values of τ.

3.11 Other implicit methods

Implementations of implicit methods that use more than one variables require the modeler to provide the **Jacobian matrix** (partial derivatives) of the function f. Most are able to compute the Jacobian matrix autonomously by doing some approximation.

There exist methods that are able to automatically switch from explicit to implicit methods by determining if the system is stiff.

Chapter 4

Relevant Examples of ODEs

4.1 Changing the notation

In the ODEs that follows we will omit any explicit reference to the time variable t:

- $X(t)$ will be just called X
- $\dot{X(t)}$ will be just called X
- X_0 will be just called $X(0)$

4.2 The Lotka-Volterra model of prey-predator interaction

These are two independent model that resulted to be equivalent:

- *Lotka* designed in 1925 as a description of an hypothetical biochemical oscillator.
- *Volterra* designed in 1926 as a description of two interacting populations.

4.2.1 Introduction

Volterra's model was made to explain a strange phenomenon observed in the Adriatic sea after World War 1. During the war there was a increasing demand of fish, so ecologist and fisherman predicted that the overall population of fish would decrease. To everyone surprise after the Wold War ended, they observed an increase in population for some species.

Volterra's idea was that prey and predator have different **(but related)** dynamics. **The main intuition was that preys proliferate in the absence of predators**.

4.2.2 Making the model

First we abstract the prey population and predator population in the following variables that uses absolute numbers:

- **V:** describes the size (also called density) of the population of preys.
- **B:** describes the size (also called density) of the population of predators.

We the make some basic observation regarding the inner dynamics of preys and predators **when they are isolated from each other**:

- When there are no predators, preys can grow without any limitation.
- When there are no preys, predators die of starvation.

We design a preliminary system of ODEs for describing these observations:

- $\dot{V} = rV$ where r is the **growth rate of preys**.
- $\dot{P} = -sP$ where s is the **death rate of predators**.

Digression

When you construct a model you can follow two approaches:

- Try to model all the possible details in the most accurate way you can. This will lead you to a very complicated model, but if you are able to take all the aspect of the system into account by measuring correctly the parameters introduced, then you should be able to replicate the reality. **So you can make quantitively prediction about that model.** However this can make the analysis difficult because if you want to explain a certain dynamics, it may be hidden behind all the parameters you have.
- Try having a minimal model when you put only the details you think are necessary to understand the trend you want to predict. Then although quantitive prediction are wrong, you will have a more clear observation of the trend.

Volterra's model follow the second approach.

4.2.3 Interaction between the two species

We have developed a system of ODEs in the case when the two species are separated from each other, but what about when both species are present in the environment? **We observe that the predators hunts the preys**. How can I model hunting in a minimal way?

Assuming that a prey and a predator meet, then the predator eats the pray and increases the change of growth of is population. Assuming that the meeting between individuals of both species is random, then to model it we add one term in each of the ODEs in our system:

- In the predator ODE it has a positive sign because by eating the pray it increases the chance of survival of its population
- in the predator ODE it has a negative sign because by being eaten it decreases the chance of survival of its population

The first intuition was that predation is proportional to the quantity of pray and predators. The rate of predations should be proportional to the quantity of meetings between individuals of the two species **(namely it is proportional to the product** $V * P$**)**. Note that the **meeting between individuals is random**.

When a prey and a predator meet, it could happen that the predator eats the prey, **but not always**.

- So we declare a as the portion of meetings resulting in huntings (the predator eats the prey).

As the predators eat, their chance of survival increases and they start to reproduce.

- So we declare b as the number of offsprings produced for each huntings.

4.2.4 Putting all together

By inserting in our system of ODEs the considerations we have made we obtain:

$$\begin{cases} \dot{V} = rV - aVP \\ \dot{P} = -sP + abVP \end{cases}$$

- **First ODE:** it tells us that prey decrease by the **hunting rate** aVP.
- **Second ODE:** it tells us that predators increase by a **hunting and reproduction rate** $abVP$.

We have modelled predation as a direct form of interaction, **it is not mediated by the environment**

4.3 Steady State

A steady state is a combination of values for the variables that remains unchanged over time. **In a steady state, all differential equations are equal to zero**.

4.4 Example - The SIR epidemic models

Epidemic phenomena deals with the spread of infectuous diseases. To study them there are used SIR models.

4.4.1 Introduction

SIR stands for the three types of individuals in our population:

- **Susceptible:** individuals that can be infected.
- **Infected:** infected individuals that can spread the disease to susceptible individuals.
- **Recovered:** infected individuals who passed the infection phase and cannot spread the disease anymore.

There are several variants of the SIR model, some being developed during the COVID pandemic.

4.4.2 Making the model

We design a Ordinary Differential Equation for each type of individual in our population, each ODE describes the ratios of each class of individual:

- **S:** ODE for the Susceptible type.
- **I:** ODE for the Infected type.
- **R:** ODE for the Recovered type.

We make the following assumption:

- The size of the population is constant in time and normalized to 1: $S+I+R=1$.
- Infected people can only transmit the infection, meaning they **cannot** reproduce, die, migrate and so on.
- A infected person can transmit the disease only to susceptible people through personal contact between the two (horizontal transmission).
- The contact between individuals is random, meaning the number of infections is proportional to both I and S.
- After being Infected a person will recover become resistant to that disease.

Following these assumption we can represent the model as the following system of equations:

$$\begin{cases} \dot{S} = -\beta SI \\ \dot{I} = \beta SI - \gamma I \\ \dot{R} = \gamma I \end{cases}$$

where:

- β: is the **infection coefficient** describing probability of infection after the contact between a susceptible individual and an infected one.
- γ: is the **recovery coefficient** describing the rate of recovery of each infected individual.

Dynamic between S and R

S and are have a very intuitive dynamic: S having only a negative sign will inevitably decrease, while R having only a positive sign will increase.

In the case of I, its behaviour strictly depends on the value of β and γ, more precisely:

- if $\beta < \gamma$: then I will decrease (since S ≤ 1)
- if $\beta > \gamma$: then the behaviour of I depends on S. If $S > \frac{\gamma}{\beta}$ then I will increase its value.

4.4.3 Extending the model - Vaccination

The SIR model can be used to study the effects of vaccinations: considering if its convenient or not to introduce mandatory vaccination for some diseases.

To better study vaccinations you need to consider the model for several years. For instance if you vaccinates newborns, you will see the result of your choice only after they will become the main part of your population. By considering a long time span **births and deaths cannot be ignored**, so we need to extend our SIR model.

idea: we add a positive terms to represents births and add a negative terms to represent death. To make this work we will use the following assumptions:

- **the population size is still constant over time** (not too wrong: the size of the population of a country does not change significantly over 10-20 years).
- No vertical transmission of the disease (meaning from parent to children.
- Newborns are considered Susceptible from the get go.

To make the population constant we introduce the coefficient μ for both birth and death. Consider that the population size is normalized to 1. We declare with N the population size s.t. $N = S + I + R$ and $\dot{N} = \mu - \mu N$.

Made these assumption we can extend the model obtaining the following system of ODEs:

$$\begin{cases} \dot{S} = \mu - \beta SI \\ \dot{I} = \beta SI - \gamma I \\ \dot{R} = \gamma I \end{cases}$$

Remember: we are assuming that all type of people could reproduce, but the newborns are always Susceptible, since we are not considering vertical transmission. Thus in the ODE of S we put $]\mu$ alone to represent the number of people born during the current time unit.

4.4.4 Introducing death

As we have stated multiple times, the population size is assumed to be constant and normalized to 1. So having newborns in our model could lead us to an increase of the population, which is wrong. The solution is to introduce death in order to maintain the size.

The death is introduced by using the same parameter as the birth μ, only with a negative sign. The obtained model will be:

$$\begin{cases} \dot{S} = \mu - \beta SI - \mu S \\ \dot{I} = \beta SI - \gamma I - \mu I \\ \dot{R} = \gamma I - \mu R \end{cases}$$

The dynamics of the model is governed by the ration between the positive and negative coefficients in the equation of I:

- $\beta < (\mu + \gamma)$: I can only decrease (since $S \leq 1$.
- $\beta > (\mu + \gamma)$: the behaviour of I depends on S. It increases if $S > \frac{(\mu+\gamma)}{\beta}$.

Note how compared to the previous case where S could only decrease, **now this is no longer true because the births could maintain S above** $\frac{\mu+\gamma}{\beta}$.

4.4.5 Introducing vaccination

Now let's consider the case where we can fight the disease by using vaccination.

We consider to vaccinates the newborns, the model can tell us if this will helps us and the fraction of newborns p are needed to be vaccinated to do so.

We consider that if a newborn is vaccinated, then it has type Recovered.

The model we get is the following:

$$\begin{cases} \dot{S} = (1-p)\mu - \beta SI - \mu S \\ \dot{I} = \beta SI - \gamma I - \mu I \\ \dot{R} = p\gamma I - \mu R \end{cases}$$

4.4.6 Determining the vaccination threshold

We declare with p_c the thresgold value of vaccinations needed, that it the ratio of newborns that should be vaccinated in order to eradicate the disease.

There are two ways to compute p_c:

- The value can be determined by simply performing numerical simulation varying the value of p
- The value of p_c can be computed analytically.

By considering the second opition we obtain the following formula to compute the ratio:

$$p_c = 1 - \frac{\mu+\gamma}{\beta}$$

4.5 Limitation of continuous dynamical models

The main limitation of this approach is that they are deterministic: once you define the ODE and you fix an initial value for the variables you obtain **THE** only possible dynamics of the system.

In many cases there are aspects you did not model and there are events that can happen with some probabilistic distribution that you have not modelled here. To overcome this limitation we need to use a different modelling technique that introduce probability distribution in our model.

Chapter 5

The Chemical Reaction Metaphor

Chemical reactions are illustrative examples of complex systems, they exhibit complex dynamics out of very simple interactions. Also chemical reaction are much easier to write than ODEs, so we will use them as a modelling language. **Chemical Reaction are used as a metaphor to describing interactions.**

5.1 What is a chemical reaction?

A chemical reaction is an interaction between molecules in a chemical solution that cause a transformation of these molecules. We categorize as:

- **reactants:** the group of molecules that participate in the reaction.
- **products:** the group of molecules (it could be a different quantities that the reactants) that are obtained after the transformation.

We abstract molecules as symbols, **it does not matter what they are in the reality**. Molecules are assumed to be in a **chemical solution**, that is a fluid medium where they float. While in these solution they can meet and start a chemical reaction.

In order to represent the quantity of molecules is usually expressed in terms of **concentrations** (number of molecules per unit).

Given the molecule A, [A] denotes the concentration of that molecule, usually expressed in $\frac{mol}{L}$.

Usually the notation for chemical reactions is some sort of rule between the reactants and the products:

$$\ell_1 S_1 + ... + \ell_p S_p \xrightarrow{k} \ell'_1 P_1 + ... + \ell'_\gamma P_\gamma$$

where:

- S_i are **reactants**.
- P_i are **products**.
- ℓ_1, $\ell_i^{'} \in \mathbb{N}$. are **stoichiometric coefficients**. They express the number of reactants/products of each type that are consumed/produced in the reaction
- $k \in \mathbb{N}_{\geq 0}$ is the **kinetic constant**. It express the rate of occurrence of reaction in a chemical solution.

Note: when we omit the l_i, it means that is value is 1.

5.1.1 Types of Chemical Reactions

There are several types of chemical reactions:

- **Synthesis:** $\xrightarrow{k} P$
- **Degradation:** $S \xrightarrow{k}$
- **Transformation:** $S \xrightarrow{k} P$
- **Binding:** $S_1 + S_2 \xrightarrow{k} P$
- **Unbinding:** $S \xrightarrow{k} P$
- **Catalysis:** $E + S \xrightarrow{k} E + P$

and many more

Complex types as aggregation of simpler type

Complex chemical reaction can be redefined as a sequence of simpler chemical reactions. Take for example the Catalysis ($E + S \xrightarrow{k} E + P$), it can be reinterpreted as a sequence of:

- Binding: $E + S \xrightarrow{k} M$
- Transformation: $M \xrightarrow{k} N$
- Unbinding: $N \xrightarrow{k} E + P$

5.1.2 Reversibility of Chemical Reactions

Often chemical reactions can occur in both directions. To indicate formulas that are reversible we will use the symbol $\leftrightarrows^{k}_{k-1}$:

$$\ell_1 S_1 + ... + \ell_p S_p \leftrightarrows^{k}_{k-1} \ell'_1 P_1 + ... + \ell'_\gamma P_\gamma$$

where $k-1 in R_{\leq 0}$ is the kinetic constant of the inverse reaction, transforming products P_i into reactants S_i.

In reality all chemical reaction are reversible, only one of the two direction is so unlikely that is not considered possible.

5.1.3 The mass action kinetics of chemical reactions

The dynamics of chemical reactions is based on the assumption that **molecules float in a well-stirred fluid medium** (e.g. water). In this scenario they are free to move and randomly meet each other. When two molecules meet, they can react.

The dynamics of chemical solution is modeled by the **law of mass action**

Law of mass action

The **rate** of a chemical reaction expresses the number of occurences of such a reaction in a given chemical solution in a time unit.

The law of **mass action** is an empirical law for the computation of the rate of chemical reaction:.

The rate of a chemical reaction is proportional to the product of the concentrations of its reactants.

Given the following chemical reaction:

$$\ell_1 S_1 + ... + \ell_p S_p \xrightarrow{k} \ell'_1 P_1 + ... + \ell'_\gamma P_\gamma$$

Then the rate of a reaction is defined as follows:

$$k[S_1]^{\ell_1}...[S_p]^{\ell_p}$$

The rate of its inverse reaction (from right-to-left) is:

$$k_{-1}[P_1]^{\ell'_1}...[P_\gamma]^{\ell'_\gamma}$$

5.1.4 Understanding the kinetic constant

What is the measure unit of a kinetic constant k? It depends on the number of reactants:

- The measure unit of **concentrations** is $\frac{mol}{L}$
- The measure unit of the **reaction rate** is $\frac{mol}{(L \times sec)}$ (it is the concentration of each product produced in one unit of time.

Note: some kinetic constants have to be changed if you change the measure unit of concentrations (e.g. μmol) or the time unit (e.g. hours).

5.1.5 Dynamic equilibrium

Given an initial concentration we can have an idea about which is the faster reaction that will take place among the one we have. Something we can compute is an equilibrium point for our reversible reactions.

Given a generic reversible reaction:

$$\ell_1 S_1 + ... + \ell_p S_p \leftrightarrows^{k}_{k_{-1}} \ell'_1 P_1 + ... + \ell'_\gamma P_\gamma$$

A dynamic equilibrium is **reached when the two rates are equal**:

$$k[S_1]^{\ell_1}...[S_p]^{\ell_p} = k_{-1}[P_1]^{\ell'_1}...[P_\gamma]^{\ell'_\gamma}$$

Then it is easy to compute the equilibrium using the following formula:

$$\frac{k}{k_{-1}} = \frac{[P_1]^{\ell'_1}...[P_\gamma]^{\ell'_\gamma}}{[S_1]^{\ell_1}...[S_p]^{\ell_p}}$$

5.2 From chemical reactions to Ordinary Differential Equations

We can use reaction rates to be able not only to compute point-wise rates and equilibrium point, but **to be able to analyse the dynamics of a system any time**.

Since rates are frequencies, **so some sort of derivative**, we can use them to define ODEs.

We will consider every molecule of the reaction as a variable of o our system of differential equation. Each molecule will have its own differential equation and then in each of these equations we will add positive and negative terms according to the reaction in which these molecules participates:

- for every reaction where the molecule we are considering is a reactant, that is **it will be consumed to generate a product**, then its ODE will contain a negative term: $-\ell r$
- for every reaction where the molecule we are considering is a product, that will be produced, then its ODE will contain a positive term $+\ell r$

where:

- ℓ: is a **stoichiometric coefficient** of a product S in R.
- r: is the rate of R.

5.3 Reverse engineering ODEs

As we have proved, starting from a system of chemical reaction we con exploit the law of mass actions to translate it into a system of polynomial ODEs.

Let us now consider the opposite: starting from a system of ODEs and finding a way to translate it into a system of chemical reactions. As we will see, **this works often, but not always**.

Notation

We begin from ODEs by omitting the $[\dot{]}$ brackets representing concentrations. The idea is to **start from a generic system of ODEs**, not one that refers to chemical reactions.

Let us consider the following system of ODE as an example:

$$\begin{cases} \frac{dX}{dt} = 6X - 0.2XY \\ \frac{dY}{dt} = 0.4XY - 2Y \end{cases}$$

We can translate it back to a system of chemical reactions by constructing one reaction for each team of each equation by matching each term with the pattern:

$$\ell_i k[S_1]^{\ell_1}...[S_p]^{\ell_p}$$

For example 6X in the equation of X tells us:

- there is a reaction producing X (since the sign of the term is positive)
- the reaction has X as its only reactant

- $\ell_i k = 6$ where ℓ_i is the number of new X to be produced.
- There is a reaction producing Y from X (since the sign of the term is negative).
- to be precise we are only sure that X will not be produced, so we select Y since its not X.
- the reaction has X and Y as its reactant.
- $\ell_i k = 0.2$ where $k = 2$ sincere there are no X to be produced.

Lets translate the whole system:

$$\begin{cases} \frac{dX}{dt} = 6X - 0.2XY \\ \frac{dY}{dt} = 0.4XY - 2Y \end{cases}$$

Will be the following reactions:

- $X \xrightarrow{6} 2X$
- $X + Y \xrightarrow{0.2} Y$
- $X + Y \xrightarrow{0.4} X + 2Y$
- $Y \xrightarrow{2}$

5.3.1 Counterexample

Consider the following example:

$$\begin{cases} \frac{dX}{dt} = 6X - 0.2XY - Y \\ \frac{dY}{dt} = 0.4XY - 2Y \end{cases}$$

Now no reaction can reduce the concentration of X without having X among its reactants.

From this counterexample we derive the following conjecture:

The translation should work for all systems of polynomial ODEs in which each negative term contains all the variable of its equation

5.4 Reverse engineering the Lotka-Volterra model

Let us see how to transform the Lotka-Volterra model into a system of chemical reactions.

Given the system of ODEs:

$$\begin{cases} \dot{V} = rV - aVP \\ \dot{P} = -sP + abVP \end{cases}$$

- **V:** denotes preys
- **P:** denotes predators.
- **a**: denotes the portion of eetings resulting in hunting.
- **b:** denotes the number of offsprings produced for each hunting.

By applying the steps we have stated, we obtain:

- $V \xrightarrow{r} 2V$
- $P \xrightarrow{s}$
- $V + P \xrightarrow{a} (1 + b)P$

5.5 Reverse engineering the SIR model

Let us see how to transform the SIR model into a system of chemical reactions.

Given the system of ODEs:

$$\begin{cases} \dot{S} = (1 - p)\mu - \beta SI - \mu S \\ \dot{I} = \beta SI - \gamma I - \mu I \\ \dot{R} = p\gamma I - \mu R \end{cases}$$

- **S:** denotes susceptible people
- **I:** denotes infected people
- **R:** denotes recovered people
- β: denotes the infection coefficient
- γ: denotes the recovery coefficient

- μ: denotes the birth and death coefficient

By applying the steps we have stated, we obtain:

- $\xrightarrow{\mu} S$
- $S \xrightarrow{\mu}$
- $I \xrightarrow{\mu}$
- $R \xrightarrow{\mu}$
- $S + I \xrightarrow{\beta} 2I$
- $I \xrightarrow{\gamma} R$

5.6 Reverse engineering the logistic equation

Recall the form of the logistic equation:

$$\dot{N} = r_c N(1 - \frac{N}{K}$$

where:

- r_c: is the **birth rate**
- **K:** is the **carrying capacity** of the environment

First we rewrite the equation as:

$$\dot{N} = r_c N - \frac{r_c}{K} N^2$$

By applying the steps we have stated, we obtain:

- $N \xrightarrow{rc} 2N$
- $2N \xrightarrow{\frac{r_c}{K}} N$

Chapter 6

Stochastic Simulation of Chemical Reactions

Recall that a Ordinary Differential Equations is continuous and deterministic, while in contrast chemical reactions can manifest themself randomly. This will led us to the definition of stochastic simulation algorithm for chemical reactions.

6.1 Introducing randomness in chemical reactions

Chemical reactions are not deterministic. Given for example to single molecules in a chemical solution, is difficult to predict in advance when they react. Doing it would require to know the angle and speed of the molecules, to take into account the molecules that compose the solution they are in and so on. Thus we abstract the time of the occurrence of the reaction using a **continuous probability distribution**.

So taking into account these information, using ODEs to represent chemical reaction seems wrong now. This is true, but if we that into account a chemical solution with an high concentrations of reactants, **the law of large numbers allow us to ignore the random aspect of our chemical reaction**. So the rule is:

- **If there are a lot of molecules:** random aspects can be ignored, permitting us to use ODEs to represent the reaction.
- **If there are a small number of molecules:** random aspects become crucial, requiring necessary the use of **descrete variables**.

6.2 Gillespie's Stochastic Simulation Algorithm (SSA)

The Gillespie's Stochastic Simulation Algorithm is an exact procedure for simulating the time evolution of a chemical reacting system by taking proper account of the randomness inherent if such a system.

Consider a set of reactions $\mathcal{R} = \{R_1, ..., R_n\}$, then the SSA:

- Assumes a stochastic reaction constant c_μ for each chemical reaction $R_\mu \in \mathcal{R}$.
- Given an infinitesimal time interval dt, then $c_\mu dt$ is the probability that a particular combination of reactants of R_μ react during the interval.

6.2.1 Explaining c_μ

The constant c_μ is used to compute the **propensity** (also called stochastic rate) of R_μ to occur in the whole chemical solution a_μ as follows:

$$a_\mu = h_\mu c_\mu$$

where h_μ is the number of distinct molecular reactant combinations.

Defined R_μ as:

$$\ell_1 S_1 + ... + \ell_p S_p \xrightarrow{c} \ell_1' P_1 + ... + l_\gamma' P_\gamma$$

The number of distinct reactant combinations of R_μ in a solution with X_i molecules of S_i with $1 \leq i \leq \rho$ is:

$$h_\mu = \prod_{i=1}^{\rho} \binom{X_i}{\ell_i}$$

Note: you can interpret the propensity as the rate of stochastic equations:**the higher is the propensity, the more often the reaction will take place**.

6.2.2 Using propensity as a stochastic rate

Propensity a_μ is used in Gillespie's as a stochastic rate. It is a parameter of a probability distribution used to describe the time of every reaction s.t on average I obtain a frequency which is the same as I got in the differential equation. To do so I will use the propensity in an exponential probability distribution to compute the time between subsequent occurrences of reaction R_μ. The obtained probability distribution will tell me when the next occurrence of the reaction we have model will take place.

6.2.3 What is an exponential distribution

An exponential distribution is a continuous probability distribution that takes place in $[0, \infty]$ describing the **timing between events** in a Poisson process, namely a process in which events occur continuously and independently at a constant average rate. The constant average rate is a parameter.

The exponential distribution is described by a negative probability density function f and by a cumulative distribution function. Both are using the parameter λ as follows:

The probability density function $f(x)$ will tell you how likely the event will take place over time. Obviously is our case we will set $\lambda = a_\mu$ (the propensity). Below the formula:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

The cumulative distribution function that tell you the probability that the event will happen from 0 to x.

$$F(x) = \begin{cases} 1 - e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Note: the mean of an exponentially distributed variable with parameter λ is $\frac{1}{\lambda}$.

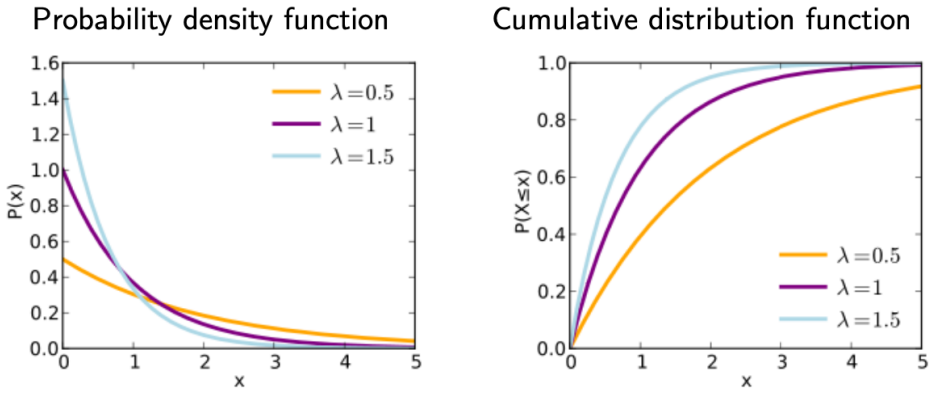


Figure 6.1: example of a graph for a Density Function and a Cumulative Function

For every reaction in the set of reaction I am considering I have a propensity that is associated to a probability distribution. We can interpret reaction as parallel processes that happen recursively and the frequency of the occurrence are computed by exponential distribution

Properties of exponential distribution

There are two important properties of the exponential distribution:

- **The exponential distribution is memoryless:**

$$P(X > t + s|X > s) = P(X > t)$$

This allows a simulation algorithm in which the exponential distribution is used to forget about the history of the simulation.

- Let $X_1, ..., X_n$ be some independent exponentially distributed random variables with parameters $\lambda_1, ..., \lambda_n$, then the equation:

$$X = min(X_1, ..., X_n$$

is also exponentially distributed. We set $\lambda = \lambda_1 + ... + \lambda_n$. This allows a simulation algorithm to use a unique exponential distribution for the whole set of reactions to be simulated.

6.2.4 Introducing the algorithm

Given:

- A set of molecular species $\{S_1, ..., S_n\}$.
- an initial numbers of molecules of each species $\{X_1, .., X_n\}$ with $X_i in \mathbb{N}$.
- a set of chemical reactions $\{R_1, ..., R_M\}$.

Gillespie's algorithm computes a possible evolution of the system.

The **state** of the simulation:

- is a vector representing the multiset of molecules in a chemical solution (at the start it is initialized as $[X_1, ..., X_n]$).
- a real variable t representing the simulation time (at the start it is initialized as $t = 0$).

Then the algorithm iterates the following steps until it reaches a final value indicated as t_{stop}:

- The time $t + \tau$ at which the next reaction will take place. The time is randomly chosen with τ exponentially distributed with parameter $a_0 = \sum_{v=1}^{m} a_v$
- The reaction R_μ that has to occur at time $t+\tau$ is randomly chosen with probability $\frac{a_\mu}{\sum_{v=1}^{M} a_v}$

At each step t is incremented by τ and the multiset representing the chemical solution is updated by subtracting reactants and adding products.

6.2.5 Implementation details

Generation of τ

Recall that τ is randomly chosen at each step. A random number with any probability distribution can be computed froma random number with uniform distribution by applying the **inversion sampling method**. The idea is to use the inverse of the cumulative distribution function.

Given a cumulative distribution function F of a probability distribution $dist$ and a uniformly distributed random variable U, the variable $X = F^{-1}(U)$ is a random variable with distribution $dist$.

In the case of the exponential distribution, the cumulative distribution function for $x \geq 0$ is $F(x) = 1 - e^{-\lambda x}$. Let us now invert the function:

$$F(X) = 1 - e^{-\lambda x} \Rightarrow 1 - F(x) = e^{-\lambda x} \Rightarrow ln(1 - F(x)) = -\lambda x$$
$$\Rightarrow -\frac{1}{\lambda} ln(1 - F(x)) = x \Rightarrow \frac{1}{\lambda} ln \frac{1}{1 - F(x)} = x$$

So we obtain:

$$F^{-1}(Y) = \frac{1}{\lambda} ln(\frac{1}{1 - Y})$$

Since Y is uniformly distributed, also $1 - Y$ is uniformly distributed. This allows us to simplify the definition of F^{-1} as:

$$F^{-1}(Y) = \frac{1}{\lambda} ln(\frac{1}{Y})$$

where:

- τ is exponentially distributed with parameter a_0 can be computed as $\tau = \frac{1}{a_0 ln(\frac{1}{Y})}$ with Y obtained from a standard random number generator.

Choice of reaction R_μ

Recall that R_μ is the reaction we randomly choose to take action at the step τ. First of all R_μ will be choose with probability $\frac{a_\mu}{a_0}$.

We do this by following these steps:

- We generate a random number N uniformly distributed in $[0, a_0$, that is $N = n * a_0$ with $n \in [0, 1)$ obtained by using a standard number generation.
- we start summing $a_1, a_2, ...$

- as soon as the sum becomes greater than N, the number of completed iterations gives you μ

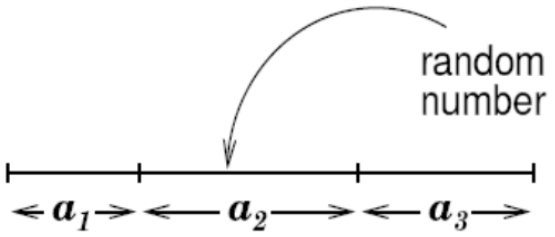


Figure 6.2: Choosing the random number

Note: μ is the smallest integer k satisfying $\sum_{i=1}^{k} a_i > na_0$ with n uniformly distributed in $[0, 1)$

6.2.6 Computational cost of Gillespie's algorithm

The problem of this stochastic approach is the computational cost: because I am executing reaction one by one, I have to repeat the simulation sever time to explore as many behaviour as possible, wasting a lot of time. In the case of large models this may become extremely high, like for example:

- when there are large number of molecules
- kinetic constant are high

In respect to ODEs, this is the only disanvantage.

6.2.7 Variants

To solve the computational drawback, several variants of Gillespie's algorithm have been introduces.

Exact approaches

Exact approaches are variants that improve the computation cost without introducing any approximation:

- Gibson and Bruck proposed the use indexed binary tree priority queue to improve the choice of the reaction R_μ for each step.

- Cao et al. and McCollum et al. proposed dynamical ordering strategies for reaction propensities $a_1, ...$ in order to probabilistically reduce the time needed to choose R_μ at each step.

Approximate approaches

Approximate approaches aims to reduce the computational cost by reducing the number of steps of the overall computation:

- Gillespie proposed the τ-leaping method: the idea is to allow several reactions to take place in a single longer time step, under the condition that reaction rates do no change too much during that time.

- Gillespie et al. proposed the slow-scale Stochastic Simulation Algorithm ssSSA which separates fast reactions from slow reactions. At each step fast reactions are dealt with by assuming that they reach a dynamic equilibrium, so only their stady state is computed. Slow reaction are simulated one by one in the standard SSA.

- Hybrid simulation is a technique which combines ODEs with stochastic simulation: ODEs are applied to molecules occurring in big numbers, stochastic simulation to molecules occurring in small number.

Note: the τ-leaping method is the most common.

Chapter 7

Transition Systems

We want to start considering models that permit us to **study the global behaviour of a system**. Assume we are interested in studying whether a system can reach a bad state:

- If we use ODEs we can state only the "average" behaviour of a system.
- if we us a Stochastic simulator we can only study a number of different possible behaviours.

Both approaches does not guarantee that the system will never reach a bad state. To solve this problem we need to introduce a new way of modeling the system behaviour by using Transition Systems.

Transition Systems are **another model of behavior**, so another way to specify how a value changes over time.

7.1 What is a Transition System

A Transition System is a pair $(S, \rightarrow)$ where:

- S is a **set of states**.
- $\rightarrow \subseteq S \times S$ is the **transition relation**.

Given two states $s, s^{'} \in S$, $(s, s^{'}) \in \rightarrow$, then we can write $s \rightarrow s^{'}$.
If instead we have $s \rightarrow$ it denotes that there exists no $s^{'} \in S$ s.t. $s \rightarrow s^{'}$.
Note that:

- The set of states can be infinity (typically is assumed to be recursively enumerable).

- Transitions describe system state changes.
- A state may have more than one outgoing transitions ($s \rightarrow s^{'}$ and $s \rightarrow s^{''}$ capturing **non-deterministic behaviors**. The fact that s_1 may evolve in either s_2 or s_3 does not necessary mean that there is a random choice between the two possibilities. The state chosen could depend from a timer, a scheduler, a probabilistic choice and so on. In general, non-determinism is an **abstraction** of choice criterion that we simply do not want to model.

7.2 Trace

A trace t in a Transition System is a path, meaning a sequence of states t = $s_0, s_1, ..., s_i, s_{i+1}, ...$ such that for each s_i and s_{i+1} ($i \in fromtheinitialstate\mathbb{N}$ it holds $s_i \rightarrow s_{i+1}$. s_0 is always chose as the **initial state**. Note:

- t = s_0 is the **minimal trace**.
- a trace t is maximal if either t is infinite or $t = s_0, s_1, ..., s_n$ and $s_n \nrightarrow$.

7.3 Reachability

Given a state s in a Transition System $(S, \rightarrow)$ we can say that s is reachable if starting from the initial state s_0 there exists a trace that starts from s_0 and ends at s: $t = s_0, s_1, ..., s_n, s$.

Very often a Transition System is used to verify that in the model we can reach a particular (good or bad) state.

Class of Transition Systems - Kripke Structures A Kirpke Structure is a class of Transion Systems where states are characterized by a set of **atomic propositions** that can either be true or false.

Given a (finite) state of atomic propositions AP, a Kripke Structure K is a Transition System $(S, \rightarrow)$ where $S = \mathcal{P}(AP)$.

- $\mathcal{P}(AP)$ denotes the powerset of AP.
- The interpretation is that an atomic proposition a is contained in a state if and only if it is true in that state

Note: a state s contains the proposition that are true when inside it.

7.4 Transition Systems over a set of variables

Given a set of variables $X = \{X_1, X_2, ..., X_n\}$ and a set of domains $\{D_1, D_2, ..., D_n\}$ s.t. D_i is the domain of X_i, a Transition System over X is a Transition System $(S, \rightarrow)$ where $S = D_1 \times D_2 \times ... \times D_n$.

7.4.1 Transition Rules

Given a Transition System over a set of variables, it can be specified b y giving a set of **transition rules** having the following form:

$$\text{guard} \rightarrow \text{update}$$

where:

- **guard**: is **conjunction of conditions** on the state variables, each having the form X_i op Exp (op is a comparison operator).
- **update**: is a **conjunction of assignments** to state variables, each having the form $X_i' = \text{Exp}$, with X_i' denoting the new values of X_i

The idea is that the transition relation contain a transition between each pair of states s_1, s_2 s.t.:

- s_1 satisfies the guard.
- s_2 can be obtained by applying to s_1 the assignments described in update.

7.5 Labeled Transition Systems

Consider a Concurrent Interaction Systems where we have several components that are in some extent independent, but they can interact with each other. Representing the system as a whole would be overwhelming, requiring lots of variables and rules. Instead it would be more natural to represent it in a **Compositional Way** by separating the variables and rules depending of which component they refer to (similar approach to object oriented programming where we create programs that use several classes, but each one is defined separately).

In order to define this compositionality feature, we need to find a way to define a transition system based on the individual components, **taking into account that they could interact between each other**. This means that in the case of two components that interact with each other, when merging their transition systems I

should be able to unify their rule that explain the interaction between them. In order to recognize these activities we introduce a **labels** and define Labeled Transition Systems.

Labeled Transition Systems (LTS) are an extended version of Transition Systems in which transitions are enriched with labels. Formally, a LTS is a triple $(S, L, \rightarrow)$ where:

- S is a set of **states**
- L is a set of **labels**
- $\rightarrow \subseteq S \times L \times S$ is a **labeled transition relation**. the triple $(s, \ell, s^{'}) \in L$ is usually denoted as $s \xrightarrow{\ell} s^{'}$.

7.5.1 Transition Labels

Transitions labels ℓ describe the action performed by the system (or a single components) during a transition. We distinguish two types of labels:

- Labels that use τ describe an internal action, that is one that is performed in isolation by the single component that contains it.
- Other labels $(a, b, c, ...)$ describe potential actions the system (or component) could perform by interaction with some other component.

7.5.2 Synchronization

There are two ways to model synchronization:

- **Binary synchronization:** it specify the interaction between two components. A transition with label a of one components has to be performed together with a transition with label $\overline{a}$ (same symbol, but overlined) of another component.
- **Global synchronization:** actions that are synchronized among all system component. All components having a transition with the same label a must perform such a transition together.

Note: the synchronization of a number of transitions result in a new τ transition.

Chapter 8

Markov Chains

Sometimes the choice criterion for the transition to pick is probabilistic or due to a stochastic race between poisson processes (race condition). To take care of these cases we need to extend the Transition Systems.

8.1 Discrete Time Markov Chains

Let us extend Transition Systems introducing probabilities.

A Discrete Time Markov Chain is a pair (S, P) where:

- S is the set of **states**.
- $P : S \times S \to [0, 1]$ is the **probability transition matrix** such that for all $s \in S$ in holds:

$$\sum_{s' \in S} P(s, s') = 1$$

The probability transition matrix can also be expressed as a **probabilistic transition relation** $\to \subseteq S \times [0, 1] \times S$ such that $(s, p, s') \in \to$ if and only if $P(s, s') = p > 0$.

Note: if $p = 0$ then the transition is usually omitted.

When the set of states is finite ($S = \{s_0, s_1, ..., s_n\}$) the probability transition matrix can be represented as a square matrix:

$$P = \begin{bmatrix} p_{00} & p_{01} & \dots & p_{0n} \\ p_{10} & p_{11} & \dots & p_{1n} \\ \dots & \dots & \dots & \dots \\ \vdots & \vdots & \dots & \vdots \\ p_{n0} & p_{n1} & \dots & p_{nn} \end{bmatrix}$$

where $p_{ij} = P(s_i, s_j)$ and the sum of each row is equal to 1.

In Discrete Time Markov Chains we usually have a probability distribution of initial states, represent as a vector:

- $[1, 0, 0]$ means that state s_0 is the only initial state
- $[0.5, 0.5, 0]$ means that s_0 and s_1 are equally likely to be initial states

The constraint $\sum_{s' \in S} P(s, s') = 1$ implies that every state has at least one outgoing transition (otherwise the sum would be 0), hence deadlocks correspond to states with a self-loop.

8.2 Paths

A path of a Discrete Time Makov Chain (DTMC) is the same concept behind the trace for a Transition System:

A path π of a DTMC described by the pair (S, P) with initial state s_0 is a possibly infinite sequence of states $\pi = s_0, s_1, ...$ such that for each s_{i+1} with $i \in \mathbb{N}$ in π it holds $P(s_i, s_{i+1}) > 0$.

The probability of a path is simply the product of the probabilities of its transitions:

$$Prob(s_0, s_1, ..., s_n) = \prod_{i=0}^{n-1} P(s_i, s_{i+1})$$

or if we do not know where it ends:

$$Prob(s_0, s_1, ...) = \prod_{i \in \mathbb{N}} P(s_i, s_{i+1})$$

8.3 Probabilistic Reachability

In a Deterministic Time Markov Chain it is possible to compute the probability that the system will reach a given state:

- **Reachability:** property expressing whether a given state can be reached (there exists a path leading to it).
- **Probabilistic reachability:** probability of reaching a given state (probabilities of all the paths leading to it).

Since paths are independent events, their probabilities can be summed.

The probability of reaching state s of a DTMC $(S, \rightarrow)$ from the initial state s_0 is the sum of the probabilities of all paths leading to it. Below the mathematical formula:

$$ProbReach(s_0, s) = \sum_{\pi \in Reach(s_0, s)} Prob(\pi)$$

where $Reach(s_0, s)$ is the set of paths reaching s (it can be infinite).

8.4 Compute probabilistic reachability

Solving probabilistic reachability $ProbReach(s, s^{'})$ amounts to solving a system of linear equations in order to obtain $x_s^{'}$:

$$\begin{cases} 1 \quad if \ s_i = s^{'} \\ \sum_{s_j \in S} P(s_i, s_j) X(s_j) \quad otherwise \end{cases}$$

where P is the probability transition matrix of the DTMC. This can be done by applying iterative computational algebra methods.

8.5 Continuous Time Markv Chains (CTMC)

Now let's extend Transition Systems with stochastic rates:

A Continuous Time Markov Chain is a pair (S, R) where:

- S is a set of **states**
- $R; S \times S \rightarrow \mathbb{R}_{\geq 0}$ is a **transition rate matrix**

The transition rate matrix can be expressed also as a **stochastic transition relation** $\rightarrow \subseteq S \times \mathbb{R}_{\geq 0} \times S$ s.t. $(s, r, s^{'}) \in \rightarrow$ if and only if $R(s, s^{'} = r > 0$ (if $r = 0$ we omit it).

8.5.1 Race conditions

If we have multiple states s' such that $R(s, s^{'}) > 0$ then we can have a **race condition**, that is the "fastest" transition determines the next state of the system. To be sure which transition we take we first have to resolve two questions,

How many time we spend in a state before the transition?

We define the **exit rate of the state s** $E(s)$ which describes the time I spend to stay in s is exponentially distributed which is the sum of the rate of the outgoing transitions:

$$E(s) = \sum_{s' \in S} R(s, s')$$

Which transition is eventually taken?

The choice of the transition to take is proportional to the rate of each transition and is independent from the time at which it occurs.

8.5.2 Discrete Time Markov Chain of a Continuous Time Markov Chain

Since we can ignore time when deciding which transition to take, we can use a Discrete Time Markov Chain to model it. We obtain our DTMC by normalizing the transition rates of the Continuous Time Markov Chain we have, in respect to the exit rate of each state. So given a $CTMC(S, R)$, its embedded DTMC is the $DTMC(S, P)$ where for any $s, s' \in S$:

$$P(s, s') = \begin{cases} R(s, s')/E(s) & if \quad E(s) > 0 \\ 1 & if \quad E(s) = 0 \ \ \& \ \ s = s' \\ 0 & otherwise \end{cases}$$

8.6 Uniformised DTMC

Given a CTMC, what is the probability of the system to be in a state s at a given time? To solve this we introduce the uniformised DTMC.

The idea is that I choose a normalization factor called the **uniformisation rate** q which is not the **exit rate of each state**, but a single factor for all parameters. The uniformisation rate q is set as being greater or equal to all the exit rates of the CTMC rates. Then I construct the uniformised DTMC by recompute each rate r of the CTMC discretizing them into probability $\frac{r}{q}$. Self-loops are added where necessary.

8.6.1 Interpretation of the uniformised DTCM

- a transition in the uniformised DTMC describes a step with duration $\frac{1}{q}$

- q should be chosed big enough to assume that at most one transition can occur during a $\frac{1}{q}$ time interval

Transient probabilistic reachability of a CTMC can now be computed as probabilistic reachability in the uniformised DTMC, bu taking the length of the paths in the DTMC into account.

Chapter 9

Model Checking

Model Checkers are tools specialized in answering question in regard to system we want to analyze.

9.1 Non-deterministic setting

Assume we are in a non deterministic setting: the system is described by using a transition system and question we want answered are modeled using temporal logic specification (a set of logic formulas). Given this parameters the model checker visit the transition system according to the logic formulas and verify if the question is true or false. If the question is false it returns a **counter-example**.

9.2 Deterministic setting

If I have a probability distribution associated to the transition system, then I can extend the approach by instead of checking the satisfiability of a boolean formula, I compute the probability that a formula is satisfied.

9.3 Problems with this approach

The larger the transition system, the more and more become complex; the larger the boolean formulas the more complex the analysis to check its satisfiability. In the end all of our study will amout in checking reachability probability properties.

9.4 Logic in Non-deterministic system

9.5 Computation Tree Logic

Computation Tree Logic (CTL) is one of the main logic for expressing requirements on behaviours. In CTL we can write formulas that can be true or false.

9.5.1 Syntax

Formulas can be declare for states or paths:

- **State formulae:**
  - $\phi ::= true|a|\phi \wedge \phi|\neg\phi|A\psi|E\psi$
  - a is an atomic proposition.
  - ψ is a path formula.
- **Path formulae:**
  - $\psi ::= X\phi|F\phi|G\phi|\phi U\phi$
  - **F:** for "future" states will satisfy the condition
  - **G:** for "global" states (all the ones considered) will satisfy the condition.
  - **U:** for "until" a condition, another condition must be satisfied.

9.5.2 Semantics

- **Quantifiers:**
  - **A:** universal quantifier (any).
  - **E:** existential quantifier (exists).
- **Temporal operators:**
  - **F:** the future states
  - **G:** the global states.
  - **U:** until, it asks if a certain property is true UNTIL another property kicks in.

9.6 Probabilistic Computation Tree Logic

Is an extension of CTL that takes intro account probability distribution.

9.6.1 Syntax

The main difference is that in state formulas we change the quantifiers with **probabilistic quantifiers**. Instead of asking if a formula is true for all states or for some, we ask if the probability that a certain condition is true is $\geq$ than a given probability.

- **State formulae:**
  - $\phi ::= true|a|\phi \wedge \phi|\neg\phi|P\ _p[\psi]$
  - a is an atomic proposition
  - $p \in [0,1]$ is a probability bound
  - $\ \in \{<,>,\leq,\geq\}$
- **Path formulae**
  - $\psi ::= X\phi|\phi U^{\leq k}\phi|\phi U\phi$
  - **X:** stands for "next"
  - $U^{\leq k}$: stands for "bounded until"
  - **U:** stands for "until"
  - $k \in \mathbb{N}$

Chapter 10

Multiset Rewriting and P Systems

A formal specification of chemical reaction and of their behavior can be given in terms of **MultiSet Rewriting (MSR)**.

10.1 Multiset

With **multiset** we intend a variant of the mathematical notion of set in which elements can be repeated. In this way we can interpret chemical solution as multisets representing molecules.

10.1.1 Formalizing multisets

Given a (finite or infinite) support set Σ, we methematically represent a multiset M over Σ in two ways:

- **set of pairs:** $M \subseteq \Sigma \times \mathbb{N}$
- **mapping:** $M : \Sigma \to \mathbb{N}$

Example

Given the set $\Sigma = \{A, B, C\}$ and the multiset $M = \{A, A, A, B, B, C, C, C\}$ over Σ can be represented in two ways:

- **set of pairs:** $M = \{(A, 3), (B, 2), (C, 3)\}$
- **mapping:** $M(A) = 3, M(B) = 2, M(C) = 3$

Note: these representations actually correspond to the representation of chemical solutions we considered in PRISM.

10.1.2 Representing Multisets as strings

Another useful representation of multisets that is very similar to formal grammars is a **string**.

We interpret the set Σ as an **alphabet**, and the multiset M over Σ correspond to strings over such alphabet. Note that in a string representing a multiset, **the order of the symbols does not matter**, so string permutations result in equivalent representation.

Example

Given $\Sigma = \{A, B, C\}$ and the multiset $M = \{(A, 3), (B, 2), (C, 3)\}$ we can represent M as the string: $M = AAABBCCC = A^3B^2C^3$ and all its possible permutations.

Multiset union can be expressed as string concatenation:

$$\{(A, 3), (B, 2), (C, 3)\} \cup \{(A, 2), (B, 1)\} = \{(A, 5), (B, 3), (C, 3)\} = A^5B^3C3$$

10.2 Representing reactions as rewriting rules

Rules are similar to rules in a formal grammar:

A multiset rewriting rule is a pair (u, v) with $u, v \in \Sigma^*$ usually denoted as $\mapsto$. When we apply the rule (u, v) to a multiset $w \in \Sigma^*$ s.t. $u \subseteq w$ what we obtain is the multiset in which u **as been replaced by** v

10.3 Definition of MultiSet Rewriting System

A MultiSet Rewriting System (MSRS) is a pair $S =< \Sigma, \mathcal{R} >$ where Σ is the **alphabet of symbols** and $\mathcal{R}$ is a **set of multiset rewriting rules**.

Given a multiset in Σ^* we can use the mechanism of rewriting rule application to compute **traces** of the multiset rewriting system.

10.4 Interleaving semantics

The behavior of a MSR system can also be described as a Transition System: we can define an (interleaving) semantics for MSR defining inference rules incorporating the mechanism of rewriting rule application.

The interleaving semantics of a MSR system $< \Sigma, \mathcal{R} >$ is the Transition System $\Sigma^*, \rightarrow$ where $\rightarrow \subseteq \Sigma^* \times \Sigma^*$ is the least transition relation satisfying the following inference

rule:

$$\frac{u \mapsto v \in \mathcal{R}}{uw \mapsto vw}$$

10.5 Stochastic MSR

We can extend the stochastic syntax and semantics of MSR to incorporate stochastic rates.

Given an alphabet Σ a stochastic multiset rewriting rule is a tuple (u, v, r) where $u, v \in \Sigma^*$ and $r \in \mathbb{R}^+$, usually denoted $u \mapsto^r v$

Stochastic Multiset Rewriting System

A Stochastic MSR system is a pair $S =< \Sigma, \mathcal{R} >$ where Σ is an alphabet of symbols and $\mathcal{R}$ is a set of stochastic multiset rewriting rules.

Semantics of Stochastic MSR

The semantics of a stochastic MSR system is a Continuous Time Markov Chain $(\Sigma^*, \rightarrow)$ where $\rightarrow \subseteq \sum^* \times \mathbb{R}^+ \times \sigma^*$ is the least stochastic transition relation satisfying the following inference rule:

$$\frac{u \mapsto^r v \in \mathcal{R}}{uw \xrightarrow{r \cdot f(u,uw)} vw}$$

Where $f(u, uw)$ gives the number of instances of u in uw.

10.6 Introducing parallelism

We can defines variantas of the language, in particular regarding the introduction if prarallelism in the application of rewriting rules:

- **simple parallelism:** one or more rule are applied at each step.
- **maximal parallelism:** as many rules as possible are applied at each step.

In this way we can model classes of systems where there are simultaneous events. In particular we are interested in maximal parallelism because its models system where there is a strong formal synchronization.

10.6.1 Parallel semantics of MSR

The parallel semantics of a MSR system $< \Sigma, \mathcal{R} >$ is the transition system $(\Sigma^*, \rightarrow)$ where $\rightarrow \subseteq \sum^* \times \sum^*$ is the lest transition relation satisfying the following inference rules:

$$\frac{u \mapsto v \in \mathcal{R}}{u \rightarrow v}$$

$$\frac{w_1 \rightarrow w_1^{'} \quad w_2 \rightarrow w_2^{'}}{w_1 w_2 \rightarrow w_1^{'} w_2^{'}}$$

$$\frac{w \rightarrow w^{'} \quad u \nrightarrow}{wu \Rightarrow w^{'} u}$$

10.7 Multiset languages

A language where we ignore the sequential ordering of symbols in its words becomes a **multiset language**. A multiset language is a set of multisets of terminal symbols and is generated by a multiset grammar.

This change in the representation of the language causes a significant change in the expressiveness of the languages, becoming more weak. Multiset languages can run on a weak-Turing Machine.

10.7.1 Introducing maximal parallelism in multiset languages

The maximal parallelism re-introduce the expressive power of grammars I lost using multiset languages. General multiset grammar rules applied with maximal parallelism are again able to generate any recursively enumerable language. Having a more complex grammar requires a Turing-complete form of automaton to accept a string from it.

10.8 P Systems

P Systems is a variant of Multiset rewriting with maximal parallelism (P stands for Paul which is the name of the inventor). What set apart P Systems from normal Multiset rewriting is that it has more than one multiset in the set.

The key elements of P Systems are:

- **Membranes:** they creates compartments used to distribute computations.
- **Multisets:** abstraction of chemical solutions that are used as data.

- **Evolution (rewriting) rules:** abstraction of chemical reaction that are used as programs.

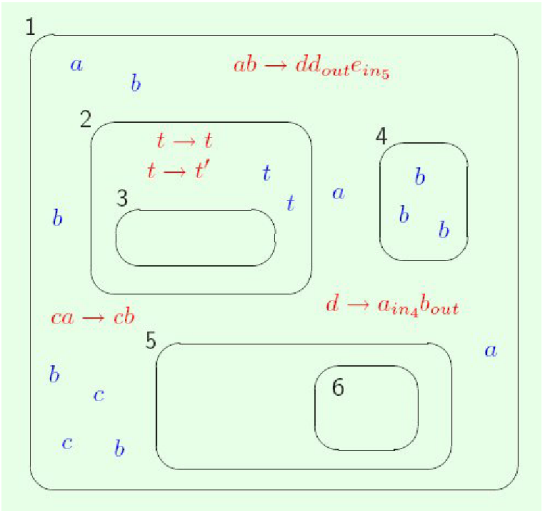


Figure 10.1: Example of a P System

10.8.1 Formal definition of a P System

A P System Π is given by:

$$\Pi = (V, \mu, w_1, ..., w_n, R_1, ..., R_n)$$

where:

- **V:** is an alphabet whose elements are called objects.
- $\mu \subset \mathbb{N} \times \mathbb{N}$ is a membrane structure, s.t. $(i, j) \in \mu$ denotes that the membrane labeled by j is contained in the membrane labeled by i.
- w_i with $1 \leq i \leq n$ are strings from V^* representing multisets over V associated with the membranes $1, 2, ..., n$ of μ.
- R_i with $1 \leq i \leq n$ are finite sets of evolution rules associated with the membranes $1, 2, ..., n$ of μ.

10.8.2 Evolution rules

An evolution rule in the form $u \rightarrow v$ consists of a multiset of objects u (representing reactants) and a multised of messages v (representing products). A message may have one of the following forms:

- a_{here}: means that object a remains in the same membrane. Often we omit *here* and just leave a alone.
- a_{out}: means that object a is sent out of the membrane.
- a_{in}: means that object a is sent into the child membrane I.

Evolution rules are classified in two ways:

- **non-cooperative rules:** the left-hand side consists of a single object (e.g. $a \rightarrow b^2 d_{out}$)
- **cooperative rules:** the left-hand side can be any multiset of objects (e.g. $a^2 b \rightarrow b^2 d_{out}$). A particular case of cooperative rules are catalytic rules, namely rules of the form $ca \rightarrow cb^2$ where c belongs to a special set of objects called catalyst.

10.8.3 Downsides of P Systems

Programming P Systems is very difficult since evolution rules are very basic.

For this reason over the years have been proposed variants of P Systems, obtained by considering different types of evolution rules:

- with rule priorities.
- with promotes and inhibitors.
- with dissolution of membranes.
- symport/antiport rules.
- with active membranes
- and so on...

Chapter 11

Petri Nets

Petri Nets is a modeling languages notation that allow us to apply different analysis approaches. Petri nets have been proposed to model concurrent systems.

11.1 Concurrent Systems

A concurrent system is a system (not strictly from computer science) where there are processes of any type executed at any time. In this system you have a workflow where some processes works in parallel (independent from each other) and others work sequentially (they must wait for the previous machines to finish).

Petri Nets are a graphical model notation for concurrent systems in which you construct a graph that describe resources and events involving those resources. There are two types of nodes:

- **Places:** represent type of resources.
  - **Tokens:** represent instances of a resource of some specific type. Tokes stay inside places. The place where they are describe their type
- **Transition:** represent events that change the state of one or more resources. They connect places to places (like a Multiset Rewriting rule or a chemical reaction). Incoming edges are connected to places that provide resources (tokens) that are consumed by the transition, while output edges represent places for which resources have been produced.

11.2 Firing a transition

Firing a transition means **executing it**. In order to fire a transition you must have enough resources in input, then it will produce the necessary resources for the output places.

11.3 Formal definition of Petri Nets

A Petri Net is a tuple $< P, T >$ where:

- P is the (finite) set of **places**.
- T is the (finite) set of **transitions** s.t. each transition t is a tuple $< I, O >$ where:
  - I is a function s.t. t consumes $I(p)$ tokens in each place p.
  - O is a function s.t. t produces $O(p)$ tokens in each place p.

11.3.1 Markings

Tokens that are placed in the network represent **the state of the network**. The state of the Petri Net is also called the **marking of the Petri Net**. A marking con be seen in two different ways:

- a function m s.t. $m(p)$ is the number of tokens in place p.
- a vector $m =< m_1, m_2, ..., m_n$ where m_i is the number of tokens in place p_i.
- a vector $m =< d_1p_1, ..., d_np_n$ where d_i represents the number of tokens in p_i.

11.3.2 Firing a transition

Given a transition $t =< I, O >$, it can be fired from m iff for any place p:

$$m(p) \geq l(p)$$

If the condition is satisfied then the firing transforms the marking m into a marking $m^{'}$ s.t. for any place p:

$$m^{'}(p) = m(p) - I(p) + O(p)$$

This transformation can be described by the following notations:

- $m \rightarrow m^{'}$

- $Post(m) = \{m^{'} | m \rightarrow m^{'}\}$. $Post(m)$ is the set of markings that can be obtained by firing transitions from m ($Post$ correspond to the transition relation in the Transition Systems)

Initial marking

The marking that we consider to be the first one is colled the **initial marking** m_0.

Reachable markings

Given two markings m and m' s.t.:

$$m \rightarrow m_1 \rightarrow m_2 \rightarrow ... \rightarrow m^{'}$$

Then we can say that $m^{'}$ is reachable from m.

Given the Petri Net N with initial marking m_0, then the function $Reach(N)$ is the set of reachable markings of N s.t.:

$$\text{Reach(N)} = \{m \text{ reachable from } m_0\}$$

11.3.3 Ordering on markings

Markings can be compared using the precedence relation $\preceq$:

- $m \preceq m^{'} \iff \forall p. m(p) \leq m^{'}(p)$.
- $m \prec m^{'} \iff m \preceq m^{'} \wedge m \neq m^{'}$

11.4 What can we use Petri Nets for

The question we can investigate on Petri Nets are these:

- **Boundedness:** is the number of reachable markings bounded?
- **Place boundedness:** in each place the number of tokens I can obtain is bounded or can it grows in an unbounded way?
- **Semi-liveness:** the transition that are present in a Petri Net will they all be fired? There are some transition that can never be fired?
- **Coverability**

11.5 Decidability of a marking

The reachability graph of a Petri Net (aka the Transition System of a MultiSet Rewriting system) can be infinite, **but the reachability property is decidable**. The computation of decidability has been proven to require **exponential time**. This means that Petri Net are not Turing equivalent.

In order to compute reachability in an efficient way, several alternatived have been proposed:

Solution 1

We consider overapproximations of the set of reachable states (based on Place Invariants or on Karp and Miller tree). We compute in polynomial time a set which is an overapproximation of the set of reachable marking. So our solution will include surely ALL the reachable marking, but also it can contain some unreachable ones. So we are only sure that if a marking is not in the overapproximation, then it is not reachable.

Solution 2

Instead of consider reachability, we will answer coverability: "is a marking greater or equal than is reachable?" The problem of coverability is weaker than reachability, but easy to compute and meaningful in the context of Petri Nets.

11.6 Place Invariants

Place Invariants are a way to reason about marking which has some analogy which the mass conservation invariant (matter can never be created or destroyed). This can be used to put constraint on a set of reachable markings.

Formally a Place invariant (or p-semiflow) is a vector i of natural numbers s.t. for any reachable marking m:

$$\sum_{p \in P} i(p) \times m(p) = \sum_{p \in P} i(p) \times m_0(p)$$

11.6.1 Invariants as overapproximations

Given a marking m and an invariant i, we can say that if m is reachable and i is an invariant, then:

$$\sum_{p \in P} i(p) \times m(p) = \sum_{p \in P} i(p) \times m_0(p)$$

The reverse is not true.

Theorem

$\forall$ Petri Net N:

$$Reach(N) \subseteq \{m | m \text{ respects some invariant of } N\}$$

So if a marking does not respect the invariant, **then we surely know that is not reachable**.

Theorem: place invariant and boundedness

If there exists a place invariant i and a place p s.t. $i(p) > 0$, then p is bounded. **The reverse is not true.**

11.7 How to compute Place invariants

11.7.1 Matrix chatacterisation

Let us represent the Petri Net as a matrix where we but places on the rows and transitions on the cons. We construct a matrix that describes the input tokens that are consumed for a given transition and a matrix for those that have been produced:

Consumption Matrix

$$W^- = \begin{bmatrix} I_1(p_1) & I_2(p_1) & ... & I_k(p_1) \\ I_1(p_2) & I_2(p_2) & ... & I_k(p_2) \\ \vdots & \vdots & ... & \vdots \\ I_1(p_n) & I_2(p_n) & ... & I_k(p_n) \end{bmatrix}$$

Production Matrix

$$W^+ = \begin{bmatrix} O_1(p_1) & O_2(p_1) & ... & O_k(p_1) \\ O_1(p_2) & O_2(p_2) & ... & O_k(p_2) \\ \vdots & \vdots & ... & \vdots \\ O_1(p_n) & O_2(p_n) & ... & O_k(p_n) \end{bmatrix}$$

Incidence Matrix

We then define the **incidence matrix** W that represent the global effect of every transition:

$$W = W^+ - W^-$$

11.8 Computing place invariants

We have said that place invariants represent mass conservation, so all the reachable markings will preserve that constraint. Reachable markings are obtained by firing transition. Then every transition that fires has to preserve the overall weight of the matrix.

Then the weight of the tokens that are produced has to be the same as the weight of the tokens that are consumed in order to preserve the overall weight.

So I express invariant by focusing on individual transitions:

$\forall$ transition $t =< I, O >$ we should have:

$$\sum_{p \in P} I(p) \times i(p) = \sum_{p \in P} O(p) \times i(p)$$

which is the same as putting:

$$\sum_{p \in P} (O(p) - I(p)) \times i(p) = 0$$

Now considering the incidence matrix W we have defined previously, any solution i is a place invariant if satisfy the following scalar product:

$$i \times W = 0$$

11.9 Reachability Tree Revised

Another approach to overapproximation is based upon the Karp and Miller Tree. In general the reachability tree that describe the system behavior can be infinite. The approach here is to construct a "smarter" reachability tree that approximate the real reachability tree in a finite way. The approximation consist in replacing infinite paths with a special node.

11.9.1 Monotonicity

The construction of our overapproximated tree is based on a property of monotonicity present in the markings of Petri Nets. Petri Nets induce strongly monotonic transition systems. So what I do is condense markings in which the tokens remains the same.

Example

Given the following path of our tree where each elements reprents a marking and each row a transition:

$$< 1, 0, 0, 0 > \rightarrow < 0, 0, 1, 0 > \rightarrow < 0, 0, 0, 1 > \rightarrow < 1, 0, 1, 1 >$$

From the first three marking we can deduce that p_1, p_3 and p_4 are unbounded. Thus we can rewrite the last marking as:

$$< \omega, 0, \omega, \omega >$$

Note: ω must be regarded as: "any numer of tokens".

Algorithm

This is how we compute the successors of node n:

11.9.2 Stopping a branch

Another small aspect to consider during the construction of the revised tree. On the current marking if I have to generate all the possible next markings considering all the possible transition that can be fired, if by firing a transition I generate a marking which is $\leq$ than a previous marking then I can cancel it, avoiding to continuing generating the marking there.

11.10 Properties of Karp and Miller Tree

11.10.1 Theorem 1

A Karp and Miller tree is always finite

Idea of the proof

If the net is not bounded, int is because of some infinite increasing sequence of marking. Such sequences are detected in a finite amount of time by adding ω in the unbounded places.

11.10.2 Theorem 2

A net is bounded iff there is no node containing an ω in its Karp and Miller tree.

11.10.3 Theorem 3

A place p is unbounded iff there exists a node labeled by m in the Karp and Miller tree s.t. $m(p) = \omega$

11.10.4 Theorem 4

Transition t is semi-live iff there exists a node labeled by m in the Karp and Miller tree s.t. t can fire in m.

11.11 Coverability set

First of all let's state the following properties:

- The set of reachable markings **might be infinite**.
- The set of labels of the nodes of the Karp and Miller tree **are always finite**

We define the Coverability set as a set of markings related to the Karp and Miller tree that will be an overapproximation on the set of reachable markings.

11.11.1 Downward-closure

Let us assume that $\forall i \in \mathbb{N} : i < \omega$. Let m be a marking (possibly containing ω), then its downward-closure is the set:

$$\downarrow m = \{m^{'} | m^{'} \preceq m\}$$

Given the set of markings $S = \{m_1, m_2, ..., m_k\}$ then:

$$\downarrow S = \downarrow m_1 \cup \downarrow m_2 \cup ... \cup \downarrow m_k$$

So the downward-closure operation can be used to construct a set of markings define as all the ones that are smaller than a given one.

11.11.2 Theorems

Let us state the following theorems:

- $\downarrow Cover(N) = \downarrow Reach(N)$
- $Reach(N) \subseteq \downarrow Cover(N)$

hence $\downarrow Cover(N)$ is a finite over-approximation of $Reach(N)$.

This means that $Cover(N)$ is another overapproximation of the set of reachable markings. If a marking is not in $\downarrow Cover(N)$, **then it is not reachable** (sufficient condition).

11.12 Another approach to reachability

In Petri Nets reachability could be achieved not requiring a **exactly that amount** of tokens to reach the marking, we can reach a marking by having **at least** that number of tokens.

The problem thus becomes the coverability problem:

Does there exist a reachable marking which is larger than some marking b?

So given a marking b, to prove that it is reachable I start from the initial marking m_0 and apply the reachability graph and as it grows I check if it includes some marking $m^{'}$ which is greater or equal that b, in which case I can say that I can cover b.

11.12.1 Formal definition

There are two alternative definitions:

- Is there a reachable marking m s.t. $m \succeq b$?
- Does $Reach(N) \cap \{m | m \succeq b\} \neq \phi$

11.12.2 First solution of the problem

We use the coverability set since it is an over-approximation of the reachable states. The coverability set can be easily computed by using the Karp and Miller algorithm.

Case where the marking is not reachable

Consider:

$$\downarrow Cover(n) \cap U = \emptyset$$

implies

$$Reach(N) \cap U = \emptyset$$

Case where the marking is reachable

There is $m \in \downarrow Cover(N) \cup U$, hence there is $m^{'} \succeq m$ which is in $Reach(N)$. However, any $m^{'} \succeq m$ is also in U. Thus there is a $m^{'}$ both in $Reach(N)$ and U.

Note: $m^{'} > m$ since $\downarrow Cover = \downarrow Reach$.

So now we have the following implication:

$$Reach(N) \cap U = \emptyset$$

implies

$$\downarrow Cover(N) \cap U = \emptyset$$

Thus we can say:

Theorem

$$Reach(N) \cap U = \emptyset \iff \downarrow Cover(N) \cap U = \emptyset$$

Which means that in order to check whether a marking in U is reachable, we can use the Karp and Miller approach to compute the $\downarrow Cover(N)$ set and check whether it has a non-empty intersection with U.

11.13 Remark regard our overapproximation methods

Note how all of the method we listed does not limit the state space, like putting an artificial bounds to the number of tokens that can be present. In Prims we require limitation on the state space because the language it uses is more richer than the one from Petri Nets.

Chapter 12

Applications of Petri Nets in Manufacturing

Petri Nets have a wide range of application domains like Performance evaluation, Business process, and Biology. Among them the most common application is in the context of manufacturing.

12.1 Flexible Manufacturing Systems

A Flexible Manufacturing System (FMS) is an application of Petri Nets in the field of manufacturing. FMS are composed by:

- A set of workstations (or machines) where products must be processed.
- A flexible transport system (e.g. rob arms) which load and unload the workstations.

The sequence of operations performed to manufacture a product is called working process (WP). A system resource is an element of the FMS that is able to hold a product (for storage, operation, transport). A FMS can execute more than one WP at the same time:

- It can execute WPs concurrently generating several instances of the same WP utilizing multiple tokens.
- Different WPs shares the same resources by being in a composed net.

12.2 Formalization of WPs

- A Working Process has an initial and final state. Initial and final state cannot use rources
- Choices are allowed in a Working Process, but iterations are not.
- Only one shared resource is allows to be used at each state of the WP, meaning a product can be held by only one resource at a time.

12.3 Deadlocks

Competition among WP for resources can lead to deadlocks. Generally deadlocks are caused by wrong resource allocation policies, causing circular wait situations for a set of resources.

There are three possible approaches to solving the problem of deadlocks:

- **Deadlock prevention:** a control policy is established in a static way in order to guarantee that deadlocks cannot be reached.
- **Deadlock avoidance:** system execution is monitored and at every time the control policy determines how to proceed in order to avoid a deadlock state.
- **Deadlock recovery:** A control policy exists which can recognize if there is a deadlock, in which case it restore the system in a non-deadlock state.

12.4 Deadlock prevention using Siphons

A method to prevent deadlocks is based on identifying siphons.

12.4.1 Siphon

A Siphon is a constraint about places. Given a place p we define the function Pre and $Post$ s.t.:

- $Pre(p)$ returns the transitions having p as an output place.
- $Post(p)$ returns the transitions having p as an input place

intuitively if we have a set of places $S = \{p_1, p_2, ...\}$:

- $Pre(p_1) \cup Pre(p_2) \cup Pre(p_2) \cup ...$

- $Post(p_1) \cup Post(p_2) \cup ...$

We define a siphon as a set of places S s.t. $Pre(S) \subseteq Post(S)$

Theorem for Siphons in FMS

Given a Petri Net modeling a FMS is deadlock free iff $\forall$ reachable marking m and every (minimal) siphon S it holds that $m(S) \neq 0$. Recall that $m(S)$ denotes the overall number of tokens in the set of places S.

Note: this theorems is valid only for Petri Net modeling a FMS.

12.4.2 Description of the method

We have to take into account all possible siphons. We create a place for each siphon in the net with a small enough number of tokens. Every time the production of a new product starts, one token is removed from each of such places (from the very beginning product reserves its right to acquire the resources involved in siphons). As soon as the product reaches a place from where one of the siphones cannot be reached, it releases the corresponding token.

12.4.3 Identifying siphons

In order to make the method usable I need a way to find syphons in my Petri Net which model a FMS. Fortunately siphons can be automatically computed by checking for each subset of places S if the condition $Pre(S) \subseteq Post(S)$ is satisfied.

We can also automatically identify siphons that can become empty by checking for each siphone S' if they do not support a place invariant.

Chapter 13

Discrete-Event Simulations (DES)

Discrete Event Simulation is a generalization of the Simulation we have seen so far. Consider the Stochastic Simulations (e.g. Gillespie's algorithm and PRISM), their limitations lies if the events of a system could be not instantaneous. Considers the case when the frequency of events follow different distributions (e.g. fixed delay, uniform distribution within an interval, ...), in that case we need to generalize the simulation to be able to allow these timing issues in our system.

Discrete Event Simulation (DES) does not rely on the memory-less property of Stochastic Simulations, **we have to keep track of what is going on in our system**. DES implement an **event list** to be aware of the time passing. In an event list we have:

- **State:** is a description of system configuration in terms of a set of variables.
- **Activity:** is a process that involves a sequence of updates of the state variables (events) over time. It has a **duration**.
- **Event:** is an instantaneous update of the state variables. They can correspond to the start and end of an activity.
- **Event notice:** is a description of a future event with the time at which it will happen.
- **Future Event List:** is a list of event notices ordered by time.

The system will be consisting in a number of activities that runs concurrently. Each of these activities that are running and that can schedule events in the future. Each activity can have variables describing the state of it and all the variables of all activities describe the state of the simulation.

An activity can be started at the beginning of the simulation or can start after a delay. Each activity as a duration (which is a different concept that delay) which is the

time in between the start event and end event. Obviously the start event describe the beginning of the execution of that activity, while the end event describe the termination. An activity can have other events.

The behavior of the activity can be specified by some code which describe the computation done by the activity and schedule some events of the activity

13.1 Future Event List

the Future Event List (FEL) is the queue we have keep track of the future events scheduled. Each element of the list is a pair $(t_i, Event_i)$ where:

- t_i is a time stamp.
- $Event_i$ is a description of the event.

It is ordered by increasing time of the time stamps.

Generalization of the simulation algorithm using the FEL

- initialize state variables, the FEL and a **global clock variable** $T = 0$. At the beginning the FEL contains only the starting events of each activities in our system.
- at each step:
  - remove the first event notice $(t, Event)$.
  - handle Event. This could cause the update of state variables, adding one or more event notices to the FEL and removing one or more scheduled events from the FEL.
  - update the global clock $T = t$
- we stop when the clock reach a stop time T_{stop} s.t. $T = T_{stop}$ or the FEL is empty.

13.1.1 Conditional and Primary Events

To handle events that have to wait for a condition in ordered to be enable we need to divide events in two types:

- **Primary events:** events whose occurrence in scheduled at a certain time.
- **Conditional events:** events that are triggered by a certain condition becoming true. **Conditional events are untimed**. For this reason we can store Conditional events at the beginning of the FEL or in an apposite data structure.

13.1.2 Revised Simulation Algorithm

We rewrite the simulation algorithm taking into account the two types of events possible:

- Initialize variables, FEL and T
- Iterate:
  - If there is a conditional event enabled, remove and process it (To does not change), otherwise remove and process the first primary event. Finally update T
- Continue until the global clock reaches the time stop $T = T_{stop}$ or the Future Event List is empty

Chapter 14

Cellular Automata

Agent-Based Modeling is a modeling approach in which system components are represented as agents able to take decisions, perform actions, and interact with other agents or/and the environment.

Usually the behaviors of an agent are specified using a high-level programming language.

Agent-Based Simulation is a form of Discrete Event Simulation that consists in "executing" agents concurrently.

14.1 Considering the environment

Agents can move in a 2D or 3D environment. Agent position and spatial charateristics of the environment will influence the system dynamics (e.g. agents can only interaction with neighbours; there could be obstacles; resources are distributed across the environment; areas could have different charateristics).

14.2 Cellular Automata

Cellular Automata (CA) allow to describe environments in 1D, 2D or 3D. The environment is described as a grid, where each cell of the grid as its own state and behavior described by something similar to an automaton. It's called Cellular Automata because in our case the automaton in our grid are in cells.

Each cell is defined by its own state that can change by means of rules. Cellular Automata can be used to model Complex Systems with spacial structure.

14.2.1 Case of application

As we have stated, in Cellular Automata we are describing an environment as a composition of cells, each with its own behavior. What we want to study is the emergent behavior obtained by the interaction of all this components. So Cellular Automata con model complex systems.

14.2.2 Modeling cellular systems

We want to define the simplest nontrivial model of a cellular system. We base our model on the following concepts:

- Cell and cellular space. We need to define a topology (more that just grids).
- Neighborhood, in the sense that any cell can interact with the cells near it.
- Cell state which describes the cell at that moment.
- Transition rules which define how a cell can change its own state.

Cellular Space

Typically we have 1D, 2D or 3D environments s.t.:

- 1D: describe the environment as a vector of cells.
- 2D: describe the environment as a grid.
- 3D: describe the environment as cubes

Neighborhood

There are two aspect to consider for defining a neighborhood:

- the radius of the neighborhood the corners of each cell describing the adjacent cells (so the nearest ones).

State Set and Transition Rule

The state of a cells depends on the application we are studying. We declare a set of possible state $S = \{s_0, ..., s_{k-1}\}$ that each cell can have.

Transition rule describe how the state of a cell change from one state to another on the basis on the current state and the state of the cells in the neighborhood.

Rules are applied in parallel to all the cells in the environment.

We need to have a complete specification regarding all possible configurations of cells and neighborhood. This is source of exponential explosion of the model, so often you do not need to define each possible rule, but instead define a global property for the neighborhood.

For this reason we define a class of special cases called **totalistic rules**, in which of instead of defining the next state of a cell depending on the precise configuration of the neighborhood, you define the function in base of the sum of the state of the neighborhood. Assume that the state of the neighborhood is represent by a number, then you can simply check its value and determine from it the next state of the cell. This reduce the complexity considering only all the possible aggregations.

A rule is outer totalistic: if it is based on the sum of the labels plus the current state of the set.

Boundary Condition

This aspect vary from model to model. The environment is typically finite, so we have to consider how to apply transition rules to cells that are in the boundary of the environment. So we have to define assumption to how to update the state for these cells.

14.3 Writing a Cellular Automata model

To implement and run a CA experiment you have to:

- Assign the geometry of the CA space
- Assign the geometry of the neighborhood
- Define the set of states of the cells
- Assign the transition rule
- Assign the initial conditions of the CA
- Repeatedly update all the cells of the CA, until some stopping condition is met.

Bibliography