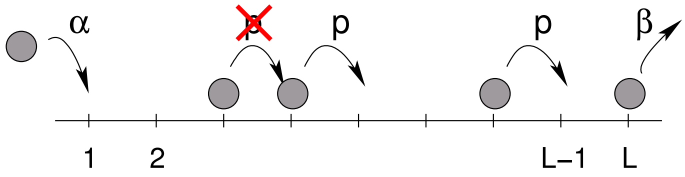
TD3 : Various update schemes for TASEP

We consider a TASEP (Totally Asymmetric Simple Exclusion Process) on a 1D lattice of length L.



- Particles are injected on site 1 with rate α if this site is empty.
- Particles hop to the next site if it is empty with rate p.
- If the last site L is occupied, the particle leaves the system with rate β.

1. Prepare the structure of a python code with

```
import numpy as np
from random import random
import matplotlib.pyplot as plt
```

Define the rates, prepare the main loop over time (we can distinguish k the number of time steps, and $t = k * \Delta t$ the real time).
The state of the system will be represented by a list or array of length L containing Boolean variables (value 0 or 1) indicating if a given site is empty or occupied.

2. **Parallel update :**
At each time step $\Delta t = 1$, all the sites are updated in parallel.
The state of the lattice at time $t + \Delta t$ is obtained from the state at time t.
(a) Write the code that will update the lattice state according to the rules above.

> ***Hint:*** Create 2 lists or arrays of size L, so that the new state can easily be obtained from the old one without erasing data.
> Note that in order to duplicate lists or arrays, you cannot just write
>
> ```
> l1 = l2
> ```
>
> You should rather use
>
> ```
> l1 = l2[:]
> ```
>
> for lists, and
>
> ```
> l1 = np.copy(l2)
> ```
>
> for numpy arrays.

> ***Hint:*** Be careful to distinguish rates and probabilities.

> ***Hint:*** The state of the system can be visualized in a spatio-temporal plot using for example
>
> ```
> # Show only occupied sites
> latplot = [np.nan if jj == 0 else 1 for jj in lat]
> # Spatio temporal plot
> plt.scatter([*range(1,LL+1)],
>   [deltat + tps*jj for jj in latplot],color='b')
> ```
>
> But use this only for a few time steps.

(b) What is the qualitative difference between the case $p = 1$ and $p < 1$?

3. **Random sequential update :**
   Each time step $\Delta t = 1$ is divided into $L + 1$ micro-timesteps $\delta t = \frac{\Delta t}{L+1}$.
   At each micro-step :
   - Choose a link (i,i+1) at random for $i = 0 \cdots L$.
   - If $i = 0$, try to inject a particle with probability $\alpha \Delta t$ if site 1 is empty.
   - If $1 \leq i \leq L - 1$, try to perform a jump if there is a particle in site i, and no particle in site i+1, with probability $p \Delta t$.
   - If $i = L$, and if there is a particle in site L, remove it with probability $\beta \Delta t$.

   (a) With the procedure described above, what is the probability according to which a given particle jumps within one time step Δt ?

   (b) Write the code that will update the lattice state according to the rules above.

   (c) Is there a qualitative difference between the case $p = 1$ and $p < 1$?

   (d) If we choose a link at random at each micro-step, it often occurs that no transition is possible along this link.
   Could we rather choose one of the particles at random ?
   In which context could it be equivalent ?

4. **Update in continuous time :**
   See notes describing the Gillespie algorithm.

   (a) List all the possible transitions, with their rates.

   (b) Implement the Gillespie algorithm for the TASEP.

> ***Hint:*** It will be useful to define a class 'Transition' with attributes 'rate', 'typ', 'link' : For example
>
> ```
> class Transition():
>   """
>   Class for all possible stochastic transitions
>   """
>   def __init__(self,rate,typ,link):
>     self.rate = rate
>     self.typ  = typ
>     self.link = link
>
>   def __repr__(self):
>     return f"t={self.typ}\nr={self.rate}\nl={self.link}"
> ```

> ***Hint:*** Random number from an exponential distribution :
> - either use the 'trick' described in the notes to obtain it from a uniform distribution
> - or use the python function numpy.random.exponential(scale) where scale is the inverse of the rate.