4AB00. Intermediate assignment

Modeling and simulation of an autonomous vehicle storage and retrieval system

1 Subject

Today's competitive environment, in which deliveries need to be faster and order sizes become smaller, forces material handling providers to progressively develop new and better solutions. A recent development in automated material-handling technology for unit load storage and retrieval is the autonomous vehicle storage and retrieval system (AVS/RS). Figure 1 shows a representation of an AVS/RS for the handling of totes, while Figure 2 illustrates a single tier (floor, level). The storage racks are single-deep and double-sided. Each storage position is of the same size

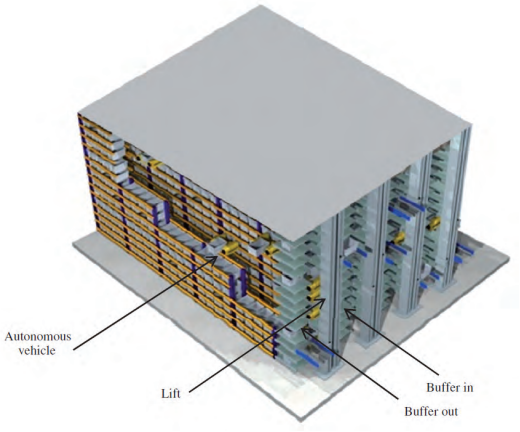

Figure 1: AVS/RS with tier-captive configuration [1]

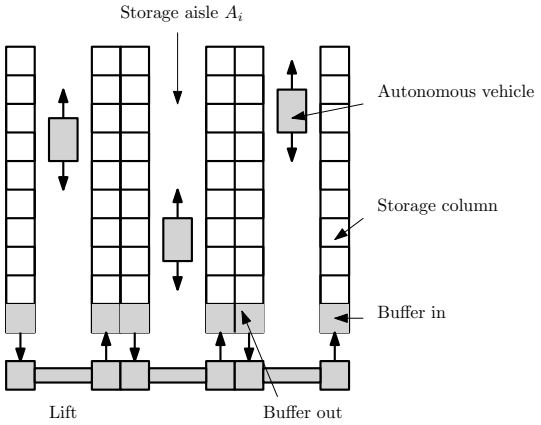

Figure 2: Single tier in AVS/RS [1]

and can hold one tote. Lifts are mounted at fixed locations at one end of each storage aisle. The input/output (I/O) point is located at the first tier beside each lift. Autonomous vehicles are dedicated to a storage aisle within a specific tier (so-called tier-captive configuration). The first position on either side of the storage aisle in all tiers serves as a buffer and is used to manage the transfer of totes between vehicles and lifts (see Figure 2). One buffer (out) handles totes which have been retrieved, the other one (in), located on the other side of the storage aisle, handles the totes to be stored. The presence of these buffers allows the lift and vehicle to work independently of each other.

The throughput performance of AVS/RS systems can be affected by design decisions (such as number of tiers, number of aisles, and depth of aisles) as well as operational decisions (such as tote storage location and order assignment).

2 Goal

The goal of this assignment is to study this new material handling solution by using computer simulation. This study can be restricted to the retrieval process only, i.e., the vehicles perform only single retrieval cycles. Clearly, the retrieval phase is the most critical activity from an organisational viewpoint, as it is directly related to customer service level and — in contrast to the storage phase — it cannot be postponed to a period of low workload.

The objectives of this assignment can be summarized as follows:

- Develop a simulation model of one aisle that consists of `Level` tiers with retrieval process to assess the system performance in terms of throughput and flow time.
- Systematically develop the model process by process, and verify and validate each process using analytical results.
- Investigate design trade-offs, i.e., via different layouts (number of columns and tiers), for this material handling system.

3 Description of the AVS/RS

The AVS/RS is a fully automated system that can store and retrieve unit loads. The unit loads are placed in totes, a type of box, and these totes are placed in a designated aisle and column for storage. When retrieved, an autonomous vehicle places the totes in a buffer at the end of the aisle. There is one autonomous vehicle for every aisle within the

tier. This is called the tier-captive configuration. From the buffer, the totes are picked by a lift that serves all tiers in one aisle, see Figure 2. In this assignment only **one aisle** that consists of `Level` tiers has to be modelled. Each aisle has `depth` number of columns of width `dv` metres and height `dl` metres.

Requests for a tote arrive for each tier, asking for a tote with a random (uniformly distributed) column assigned to it. The inter arrival time of requests is exponentially distributed with mean value `arrive`.

The vehicles transport totes along the aisle (one vehicle per tier). The time taken by the vehicle to process an order depends on the tote's location. The vehicle has to travel to this location, load the tote (it takes a fixed amount of time equal to `lv` seconds), then go back to the end of the aisle and, finally, unload the tote to the buffer (again this takes `lv` seconds). The speed profile of each vehicle is described as follows. First, the vehicle goes with a constant acceleration `av` m/s^2 until it reaches its maximum velocity `vmaxv`, then it moves with this velocity and decelerates with a constant deceleration `av` to approach its destination point. If the tote is located too close to the end of the aisle, it is possible that the vehicle will not reach its maximum speed.

The vehicles unload the totes to a location where it is picked up by the lift. At any given time only one tote per tier is allowed to occupy such a position.

The lift transports the totes from every tier to the ground floor. The processing time depends on the tote's location and the fixed time to load and unload the tote (`ll` seconds for loading and `ll` seconds for unloading). The maximum speed of the lift is `vmaxl` m/s with a speed profile similar to that of the vehicles (constant acceleration/deceleration of `al` m/s^2).

4 Modeling of the AVS/RS

The model consists of several processes communicating via channels, see Figure 3.

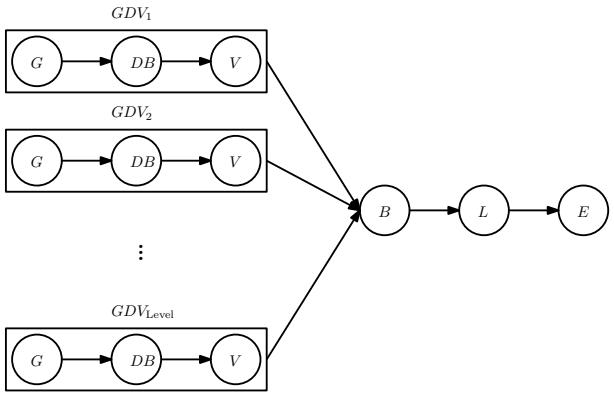

Figure 3: The processes in the model

A brief description of the processes is given below:

G: The generator models the arrival of orders for totes. There is one generator per tier.

DB: The demand buffer stores the orders to be picked up by the vehicles. The vehicle processes the orders under first-come-first-serve policy. It can always receive a tote, but it releases a tote only when the vehicle is ready to pick it up.

V: The vehicle can be modeled as a server with a variable processing time.

GDV: The process consisting of a Generator, Demand Buffer and Vehicle for one tier of the AVS/RS system.

B: The buffer stores the totes from the vehicles awaiting for a pick-up by the lift. The buffer operates under first-come-first-serve policy and has a finite capacity of `bc` totes **per tier**. So, every tier has a buffer location of finite capacity of `bc` totes. All those locations together form the buffer B that accommodates all the totes which are processed by the same lift. When the buffer is not empty (there is at least one tote on one of the tiers), the buffer sends information about the tier of the longest waiting tote to the lift upon request. This information is used by the lift to determine the location of the tote (the number of the tier the tote is located in). Once the lift is ready to pick it up, the tote is released to the lift. There are two types of queues in the buffer: `xs` contains all the totes to be picked up by the lift and there are `Level` counters `n` counting the totes from the corresponding tier.

L: The lift, similar to the process V, is modeled as a machine with a variable processing time. A lift first receives a destination tier. Then the process delays for time required for the lift to go to the tote location and pick up the tote. Then it receives the tote, and next it delays for the time required to go to the ground floor and to unload the tote. After that the lift process is ready to release the tote to the Exit process.

E: In the exit process all the data related to each tote are collected. This process calculates the average throughput and the average flow-time. The number of totes to be processed by the AVS/RS during one simulation run is determined by the constant `Number_of_orders`.

5 Remark concerning report

The report has to contain concise answers to the questions posed. If a chi code has to be presented, present the specification **only of the modified process**. Avoid unnecessary long explanations, however answers to analytical questions without an argument will result in no points.

6 Assignment

When making a Chi specification, one does not build an entire model and then starts debugging. A model is build step by step. During each step proper functionality is checked. We therefore model one process at a time, and connect the process with a Generator and/or Exit process to test proper functionality. In order to test proper functionality we first use analytical methods to determine the expected outcome. Then we run simulations to verify if these outcomes are indeed obtained. If that is the case, we can be more convinced that the process has been modeled correctly.

6.1 Exercise 1: Generator (5 points)

So the first step is to write a proper generator for a tier. To that end, the first model we build consists of (only) a Generator process and an Exit process collecting the totes generated by the Generator. The inter arrival time of requests is exponentially distributed with mean `arrive`. The requested tote should be retrieved from a random (uniformly distributed) column, where there are `depth` columns in total.

a. Determine (analytically) the throughput and flow time that should result from your model with only the generator process and the exit process.

b. Complete the chi specification from the following template:

```
type tote = tuple(real entrytime;
                  int  column);

const real  arrive = 70.0,             # inter arrival time of requests
      int   depth  = 55,               # the number of columns
      int   number_of_orders = 10000;  # the number of orders to process

proc Generator(chan! tote a):
  .....;
  .....;
  tote x;
  while true:
    x.entrytime = time;
    x.column = .....;
    a!x;
    delay .....;
  end
end

proc Exit(chan? tote a):
  tote x;
  real mphi;
  for i in range(1, number_of_orders + 1):
    a?x;
    mphi = (i - 1) / i * mphi + (time - x.entrytime) / i;
    write("tote = %d; Entrytime = %10.4f; Column = %2d; Mean throughput = %8.6f; Mean flowtime = %6.4f\n",
          i        , x.entrytime       , x.column   , i / time               , mphi );
  end
end

model M():
```

```
  chan tote a;
  run Generator(a), Exit(a)
end
```

c. Confirm that your simulations produce the correct throughput and flow time. Also determine the minimal and maximal column returned by your simulation.

d. Run your simulation at least 10 times. What is the range of outcomes for the mean throughput? Determine (roughly, by trial and error) the number of orders required to get good estimates for the throughput (first three non-zero digits are correct).

6.2 Exercise 2: Demand Buffer (5 points)

Next, we include the Demand buffer which is modelled as a standard FIFO buffer with infinite capacity.

a. Determine (analytically) the throughput and flow time that should result from your model with only the generator process, demand buffer, and the exit process.

b. Complete the chi specification from the following template:

```
type tote = tuple(real entrytime;
                  int  column);

const real  arrive = 70.0,             # inter arrival time of requests
      int   depth  = 55,               # the number of columns
      int   number_of_orders = 10000;  # the number of orders to process

proc Generator(chan! tote a):
  .....;
  .....;
  tote x;
  while true:
    x.entrytime = time;
    x.column = .....;
    a!x;
    delay .....;
  end
end

proc Demand_Buffer(chan? tote a; chan! tote b):
  tote x;
  list tote xs;
  .....
  .....
  .....
  .....
  .....
end

proc Exit(chan? tote a):
  tote x;
  real mphi;
  for i in range(1, number_of_orders + 1):
    a?x;
    mphi = (i - 1) / i * mphi + (time - x.entrytime) / i;
    write("tote = %d; Entrytime = %10.4f; Column = %2d; Mean throughput = %8.6f; Mean flowtime = %6.4f\n",
          i        , x.entrytime       , x.column   , i / time               , mphi );
  end
end

model M():
  chan tote a,b;
  run Generator(a), Demand_Buffer(a,b), Exit(b)
end
```

c. Confirm that your simulations produce the correct throughput and flow time.

6.3 Exercise 3: Vehicle as machine (5 points)

Next, we include the Vehicle which as a first attempt is modelled as a standard machine with a deterministic processing time of 40.0 seconds.

a. Determine (analytically) the throughput and flow time that should result from your model with only the generator process, demand buffer, vehicle and the exit process.

b. Complete the chi specification from the following template:

```
type tote = tuple(real entrytime;
                  int  column);

const real  arrive = 70.0,             # inter arrival time of requests
      int   depth  = 55,               # the number of columns
      int   number_of_orders = 10000;  # the number of orders to process

proc Generator(chan! tote a):
  .....;
  .....;
  tote x;
  while true:
    x.entrytime = time;
    x.column = .....;
    a!x;
    delay .....;
  end
end

proc Demand_Buffer(chan? tote a; chan! tote b):
  tote x;
  list tote xs;
  .....
  .....
  .....
  .....
  .....
end

proc Vehicle(chan? tote a; chan! tote b):
  tote x;
  .....
  .....
  .....
  .....
  .....
end

proc Exit(chan? tote a):
  tote x;
  real mphi;
  for i in range(1, number_of_orders + 1):
    a?x;
    mphi = (i - 1) / i * mphi + (time - x.entrytime) / i;
    write("tote = %d; Entrytime = %10.4f; Column = %2d; Mean throughput = %8.6f; Mean flowtime = %6.4f\n",
          i        , x.entrytime       , x.column   , i / time               , mphi );
  end
end

model M():
  chan tote a,b,c;
  run Generator(a), Demand_Buffer(a,b), Vehicle(b,c), Exit(c)
end
```

c. Confirm that your simulations produce the correct throughput and flow time.

d. Run your simulation at least 10 times. What is the range of outcomes for the mean throughput? Determine (roughly/ by trial and error) the number of orders required to get good estimates for the flow time (first three non-zero digits are correct).

6.4 Exercise 4: Vehicle accurately modelled (15 points)

Assume that a vehicle starts from the buffer. If it has to go to column $i \in \{0, \texttt{depth} - 1\}$, it needs to drive a distance $\texttt{dv} \cdot (i + 1)$, pick up the tote, drive back, and deliver the tote to the buffer. For driving the required distance, the vehicle first goes with a constant acceleration `av` m/s^2 until it reaches its maximum velocity `vmaxv`, then it moves with this velocity and decelerates with a constant deceleration `av` to approach its destination point. If the tote is located too close to the end of the aisle, it is possible that the vehicle will not reach its maximum speed.

a. From this information, (analytically) determine the time (in seconds) it takes a vehicle to drive a given distance of x meters. To that end, first determine from the acceleration profile the travelled distance as function of time. From that you can determine the time required to travel a given distance.

b. Next, if loading and unloading a tote both take `lv` seconds, determine (analytically) the time it takes a vehicle, starting from the buffer, to pick up a tote from column $i \in \{0, \texttt{depth} - 1\}$ and deliver it to the buffer.

c. Determine (analytically) the mean and variance of this total travel time for a random (uniformly distributed) column.

d. Determine (analytically) the throughput and flow time that should result from your model with only the generator process, demand buffer, vehicle and the exit process.

e. Complete the chi specification from the following template, incorporating the actual time a vehicle requires for picking up a tote and delivering it to the buffer.

```
type tote = tuple(real entrytime;
                  int  column);

const real arrive = 70.0,             # inter arrival time of requests
      int  depth  = 55,               # the number of columns
      int  number_of_orders = 10000,  # the number of orders to process
      real lv = 3.0,                  # time to load/unload the vehicle
      real dv = 0.5,                  # unit width clearance
      real vmaxv = 1.5,               # maximum velocity of the vehicle
      real av = 1.0;                  # acceleration/deceleration of the vehicle

proc Generator(chan! tote a):
  .....;
  .....;
  tote x;
  while true:
    x.entrytime = time;
    x.column = .....;
    a!x;
    delay .....;
  end
end

proc Demand_Buffer(chan? tote a; chan! tote b):
  tote x;
  list tote xs;
  .....
  .....
  .....
  .....
  .....
end

proc Vehicle(chan? tote a; chan! tote b):
  tote x;
  .....
  .....
  .....
  .....
  .....
end

proc Exit(chan? tote a):
  tote x;
  real mphi;
  for i in range(1, number_of_orders + 1):
    a?x;
    mphi = (i - 1) / i * mphi + (time - x.entrytime) / i;
```

```
    write("tote = %d; Entrytime = %10.4f; Column = %2d; Mean throughput = %8.6f; Mean flowtime = %6.4f\n",
          i        , x.entrytime       , x.column   , i / time               , mphi );
  end
end

model M():
  chan tote a,b,c;
  run Generator(a), Demand_Buffer(a,b), Vehicle(b,c), Exit(c)
end
```

f. Confirm that your simulations produce the correct throughput and flow time.

g. Run your simulation at least 10 times. What is the range of outcomes for the mean throughput? Determine (roughly, by trial and error) the number of orders required to get good estimates for the flow time (first three non-zero digits are correct).

6.5 Exercise 5: Lift as two stage machine (5 points)

In the previous exercises we focussed on modeling a tier. In exercises 5–8 we focus on modeling the Buffer and Lift, starting with the Lift. We model the Lift as a machine, which repeatedly does the following: request a destination tier, move to the desired tier, receive a tote (first delay for `ll`, then communicate), move to the ground floor, deliver the tote (first delay for `ll`, then communicate). In this exercise we assume that moving to the desired tier takes 23.0 seconds, receiving the tote takes 2.0 seconds, moving to the ground floor again takes 23.0 seconds, and delivering the tote also takes 2.0 seconds. Note that, in this case, this implies that the tote should leave the Generator and enter the Lift 25.0 seconds after the Lift has received the destination tier. All of these times are assumed to be deterministic.

a. Assume that after 1.0 second the first tote arrives. Subsequently a tote arrives respectively 10.0, 100.0, and 100.0 seconds after the completion of sending its predecessor to the buffer. Make a lot-time-diagram indicating how long a newly generated job spends in the Generator, and spends in the Lift.

b. Complete the chi specification from the following template:

```
type tote = tuple(real entrytime;
                  int  tier);

const real ll = 2.0;                  # time to load/unload the lift

proc Generator(chan! tote a; chan! int b):
  list real delays=[1.0, 10.0, 100.0, 100.0];
  while size(delays)>0:
    delay delays[0];
    delays=delays[1:];
    b!1;
    writeln("Generator: Informed Lift to go to tier 1 at time %3.1f",time);
    a!(time,1);
    writeln("Generator: Tote has completely left the Generator and entered the Lift at time %3.1f",time)
  end;
  delay(100.0)
end

proc Lift(chan? tote a; chan? int b; chan! tote c):
  tote x;
  .....
  .....
  .....
  .....
  .....
end

proc Exit(chan? tote a):
  tote x;
  while true:
    a?x;
    writeln("Exit:      Tote has completely left the Lift and has been received by the Exit at time %3.1f",time);
  end
end

model M():
  chan tote a,c;
```

```
  chan int b;
  run Generator(a,b), Lift(a,b,c), Exit(c)
end
```

c. Confirm that your simulations produce the correct results, i.e., in correspondence with your lot-time-diagram

6.6 Exercise 6: Buffer with only one tier (5 points)

In case we have only one tier, we could use a standard model for a finite buffer. However, keeping in mind that we need to model in the next exercise `Level` finite buffers in parallel where totes are served at a first-come-first-serve basis, we model the finite buffer slightly differently. We have a counter `n` which counts the number of jobs in the buffer, and we keep a list `xs` of totes that have arrived to the buffer. As long as the buffer contains less than `bc` totes, the buffer can receive jobs. Furthermore, if the buffer contains totes, it is possible to receive a request from the Lift (asking for which tier to move to, which is always 0 in this case). Additionally, if the buffer contains totes it is possible to send the first tote of the list `xs` to the lift.

a. Assume that after 1.0 second the first tote arrives. Subsequently a tote arrives respectively 10.0, 100.0, and 100.0 seconds after the completion of sending its predecessor to the buffer. Make a lot-time-diagram indicating how long a newly generated job spends in the Generator, in the Buffer, and spends in the Lift.

b. Complete the chi specification from the following template:

```
type tote = tuple(real entrytime;
                  int  tier);

const real ll = 2.0,                  # time to load/unload the lift
      int  bc = 1;                    # buffer capacity

proc Generator(chan! tote a):
  list real delays=[1.0, 10.0, 100.0, 100.0];
  while size(delays)>0:
    delay delays[0];
    delays=delays[1:];
    a!(time,0);
    writeln("Generator: Sending tote to Buffer completed at time %3.1f", time)
  end;
  delay(100.0)
end

proc Buffer(chan? tote a; chan! tote b; chan! int c):
  tote x;
  list tote xs;
  int n;
  while true:
    select
      n < bc, a?x:
        xs = xs + [x]; n = n + 1
    alt
      not empty(xs), c!xs[0].tier:
        pass
    alt
      not empty(xs), b!xs[0]:
        .....
    end
  end
end

proc Lift(chan? tote a; chan? int b; chan! tote c):
  tote x;
  .....
  .....
  .....
  .....
  .....
end

proc Exit(chan? tote a):
  tote x;
  while true:
    a?x;
```

```
    writeln("Exit:      Receiving tote from Lift completed at time %3.1f",time);
  end
end

model M():
  chan tote a,b,d;
  chan int c;
  run Generator(a), Buffer(a,b,c), Lift(b,c,d), Exit(d)
end
```

c. Confirm that your simulations produce the correct results, i.e., in correspondence with your lot-time-diagram

6.7 Exercise 7: Buffer with multiple tiers (5 points)

We extend the buffer from the previous exercise to `Level` parallel finite buffers of size `bc` which are modelled as a list of `Level` counters. For this list we use the variable `n`. We keep a common list `xs` of totes that have arrived to all buffers. This list is used to determine which tote should be picked up first by the Lift. For each of the tiers it holds that if the buffer of that tier is not full yet, a tote can be received. Furthermore, if the buffer contains totes, it is possible to receive a request from the Lift asking for the tier to move to. Additionally, if the buffer contains totes it is possible to send the first tote of the list `xs` to the lift.

a. Consider two tiers, and assume that after `Tier` seconds the first tote arrives (for two tiers: after 0.0 for tier 0 and after 1.0 for tier 1). Subsequently a tote arrives respectively 10.0, 100.0, and 100.0 seconds after the completion of sending its predecessor to the buffer. Make a lot-time-diagram indicating how long a newly generated job spends in which Generator, in the Buffer, and spends in the Lift.

b. Complete the chi specification from the following template:

```
type tote = tuple(real entrytime;
                  int  tier);

const real ll = 2.0,                  # time to load/unload the lift
      int  bc = 1,                    # buffer capacity
      int Level = 2;                  # the number of tiers

proc Generator(chan! tote a; int level):
  list real delays=[1.0*level, 10.0, 100.0, 100.0];
  while size(delays)>0:
    delay delays[0];
    delays=delays[1:];
    a!(time,level);
    writeln("Generator %d: Sending tote to Buffer completed at time %3.1f",level,time)
  end;
  delay(100.0)
end

proc Buffer(list(Level) chan? tote a; chan! tote b; chan! int c):
  tote x;
  list tote xs;
  list(Level) int n;
  while true:
    select
      unwind j in range(Level):
        n[j] < bc, a[j]?x:
          xs = xs + [x]; n[j] = n[j] + 1
      end
    alt
      not empty(xs), c!xs[0].tier:
        pass
    alt
      not empty(xs), b!xs[0]:
        .....
    end
  end
end

proc Lift(chan? tote a; chan? int b; chan! tote c):
  tote x;
  .....
  .....
```

```
  .....
  .....
  .....
end

proc Exit(chan? tote a):
  tote x;
  while true:
    a?x;
    writeln("Exit:        Receiving tote from Lift completed at time %3.1f",time);
  end
end

model M():
  list(Level) chan tote a;
  chan tote b,d;
  chan int c;
  run Generator(a[0],0), Generator(a[1],1), Buffer(a,b,c), Lift(b,c,d), Exit(d)
end
```

c. Confirm that your simulations produce the correct results, i.e., in correspondence with your lot-time-diagram

6.8 Exercise 8: Lift accurately modelled (5 points)

Assume that the Lift starts from the ground floor. If it has to go to level $i \in \{0, \texttt{Level} - 1\}$, it needs to travel a distance $\texttt{dl} \cdot (i + 1)$, pick up the tote, travel back, and deliver the tote at the ground floor. For traveling the required distance, the vehicle first goes with a constant acceleration `al` m/s^2 until it reaches its maximum velocity `vmaxl`, then it moves with this velocity and decelerates with a constant deceleration `al` to approach its destination tier. If the tier is located too close to the ground floor, it is possible that the Lift will not reach its maximum speed.

a. From this information, (analytically) determine the time (in seconds) it takes the Lift to a given tier.

b. Next, if loading and unloading a tote both take `ll` seconds, determine (analytically) the time it takes the Lift to pick up a tote from tier $i \in \{0, \texttt{depth} - 1\}$ and deliver it to the buffer.

c. Consider two tiers, and assume that after `Tier` seconds the first tote arrives (for two tiers: after 0.0 for tier 0 and after 1.0 for tier 1). Subsequently a tote arrives respectively 10.0, 100.0, and 100.0 seconds after the completion of sending its predecessor to the buffer. Make a lot-time-diagram indicating how long a newly generated job spends in which Generator, in the Buffer, and spends in the Lift.

d. Complete the chi specification from the following template:

```
type tote = tuple(real entrytime;
                  int  tier);

const int  bc = 1,                    # buffer capacity
      int  Level = 2,                 # the number of tiers
      real ll = 2.0,                  # time to load/unload the lift
      real dl = 0.8,                  # unit height clearance
      real vmaxl = 5.0,               # maximum velocity of lift
      real al = 7.0;                  # acceleration/deceleration of lift

proc Generator(chan! tote a; int level):
  list real delays=[1.0*level, 10.0, 100.0, 100.0];
  while size(delays)>0:
    delay delays[0];
    delays=delays[1:];
    a!(time,level);
    writeln("Generator %d: Sending tote to Buffer completed at time %3.1f",level,time)
  end;
  delay(100.0)
end

proc Buffer(list(Level) chan? tote a; chan! tote b; chan! int c):
  tote x;
  list tote xs;
  list(Level) int n;
  while true:
    select
      unwind j in range(Level):
```

```
        n[j] < bc, a[j]?x:
          xs = xs + [x]; n[j] = n[j] + 1
      end
    alt
      not empty(xs), c!xs[0].tier:
        pass
    alt
      not empty(xs), b!xs[0]:
        .....
    end
  end
end

proc Lift(chan? tote a; chan? int b; chan! tote c):
  tote x;
  .....
  .....
  .....
  .....
  .....
end

proc Exit(chan? tote a):
  tote x;
  while true:
    a?x;
    writeln("Exit:        Receiving tote from Lift completed at time %3.1f",time);
  end
end

model M():
  list(Level) chan tote a;
  chan tote b,d;
  chan int c;
  run Generator(a[0],0), Generator(a[1],1), Buffer(a,b,c), Lift(b,c,d), Exit(d)
end
```

e. Confirm that your simulations produce the correct results, i.e., in correspondence with your lot-time-diagram

6.9 Exercise 9: Entire system (10 points)

In the previous exercises we developed and tested each process individually. Now it is time to combine all processes into one model.

a. Determine the (total) arrival rate of orders for a system with `Level` tiers, where for each tier orders arrive with a mean inter arrival time of `arrive`.

b. Complete the chi specification from the following template:

```
type tote = tuple(real entrytime;
                  int column;
                  int  tier);

const real lv = 3.0,                  # time to load/unload the vehicle
      real dv = 0.5,                  # unit width clearance
      real vmaxv = 1.5,               # maximum velocity of the vehicle
      real av = 1.0,                  # acceleration/deceleration of the vehicle
      real ll = 2.0,                  # time to load/unload the lift
      real dl = 0.8,                  # unit height clearance
      real vmaxl = 5.0,               # maximum velocity of lift
      real al = 7.0,                  # acceleration/deceleration of lift
      int  bc = 1,                    # buffer capacity
      real arrive = 70.0,             # inter arrival time of requests
      int  Level = 9,                 # the number of tiers
      int  depth  = 55,               # the number of columns
      int  number_of_orders = 100000;  # the number of orders to process

proc Generator(chan! tote a; int Tier):
  .....
  .....
  tote x;
  while true:
```

```
    x.entrytime = time;
    x.column = .....
    x.tier = .....
    a!x;
    delay .....
  end
end

proc Demand_Buffer(chan? tote a; chan! tote b):
  tote x;
  list tote xs;
  .....
  .....
  .....
  .....
  .....
end

proc Vehicle(chan? tote a; chan! tote b):
  tote x;
  .....
  .....
  .....
  .....
  .....
end

proc Buffer(list(Level) chan? tote a; chan! tote b; chan! int c):
  tote x;
  list tote xs;
  list(Level) int n;
  while true:
    select
      unwind j in range(Level):
        n[j] < bc, a[j]?x:
          xs = xs + [x]; n[j] = n[j] + 1
      end
    alt
      not empty(xs), c!xs[0].tier:
        pass
    alt
      not empty(xs), b!xs[0]:
        .....
    end
  end
end

proc Lift(chan? tote a; chan? int b; chan! tote c):
  tote x;
  .....
  .....
  .....
  .....
  .....
end

proc Exit(chan? tote a):
  tote x;
  real mphi;
  for i in range(1,number_of_orders+1):
    a?x;
    mphi = (i - 1) / i * mphi + (time - x.entrytime) / i;
    write("tote = %6d; Entrytime = %10.4f; Tier = %2d; Column = %2d; Mean throughput = %8.6f; Mean flowtime =
        %6.4f\n",
          i         , x.entrytime       , x.tier    , x.column    , i / time               , mphi );
  end
end

proc GDV(chan! tote c; int Tier):
  chan tote a,b;
  run Generator(a, Tier), Demand_Buffer(a, b), Vehicle(b, c);
end

model M():
```

```
  list(Level) chan tote a;
  chan tote b,d;
  chan int c;
  run
    unwind Tier in range(Level):
      GDV(a[Tier], Tier),
    end,
    Buffer(a,b,c), Lift(b,c,d), Exit(d)
end
```

by substituting the code for the processes as you derived in Exercises 1, 2, 4, 7, and 8. Note that the type `tote` has been extended in comparison with Exercise 1, so one line of code extra is required in the Generator process to include the tier.

c. Verify if the throughput in your simulations matches with the start rate you determined. Verify that totes reach the exit from each tier. Also verify that totes are generated from each column.

d. Run your simulation 30 times, and record for each run the resulting average throughput and average flow time. Determine the mean and the standard deviation of both the average throughput and the average flow time for these 30 simulations. Finally, determine a 95% confidence interval for both the average throughput and the average flow time.

6.10 Exercise 10: Effect of buffer capacity (10 points)

Via computer simulation analyze the impact of the buffer capacity on the throughput and flow time (choose values of `bc` as 1,2,3,200). For each setting, run your simulations 30 times and determine both mean and standard deviation of the resulting average throughput and average flow time for these 30 simulations, as well as 95% confidence intervals.

6.11 Exercise 11: Looking for the best layout (30 points)

Via computer simulation consider 6 different scenarios (see the table below) that correspond to different layouts of the storage facility with a total capacity of 500 totes with different number of tiers and columns.

`Level`	2	5	10	20	25	50
`depth`	250	100	50	25	20	10

In order to obtain an answer to the questions below: for each scenario, run your simulations 30 times and determine both mean and standard deviation of the resulting average throughput and average flow time for these 30 simulations, as well as 95% confidence intervals.

a. For each scenario, determine by means of simulation the maximal throughput.

b. Assume a required throughput of 6 totes per minute. Compare the resulting flow time for each feasible scenario.

c. Based on your outcomes, select a layout that seems the best for you. If necessary, introduce other performance indicators and modify the code accordingly. The report on this step should contain an argument to select the best layout.

References

[1] G. Marchet, M. Melanci, S. Perotti and E. Tappia, "Analytical model to estimate performance of autonomous vehicle storage and retrieval systems for product totes", *International Journal of Production Research*, vol. **50**(24), 7134-7148, 2012.