Exercise 6-1 **Threads** (H)

Given the following parallel program.

```
x, y <- 0
thread 1 do
   x <- x + 1
   y <- y + 1
end thread

thread 2 do
  x <- x + 2
  y <- y + 2
end thread
```

Define an LTS for this program.

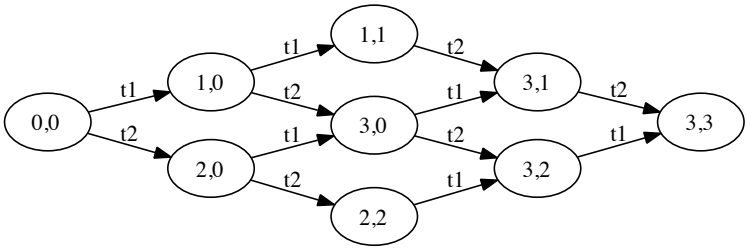
a) Define the sets S and A.

Solution:

$$A = \{\text{t1}, \text{t2}\}$$
$$S = \{(0,0), (1,0), (2,0), (1,1), (3,0), (2,2), (3,1), (3,2), (3,3)\}$$

b) Draw a diagram of the LTS.

Solution:



Exercise 6-2 **Critical section** (H)

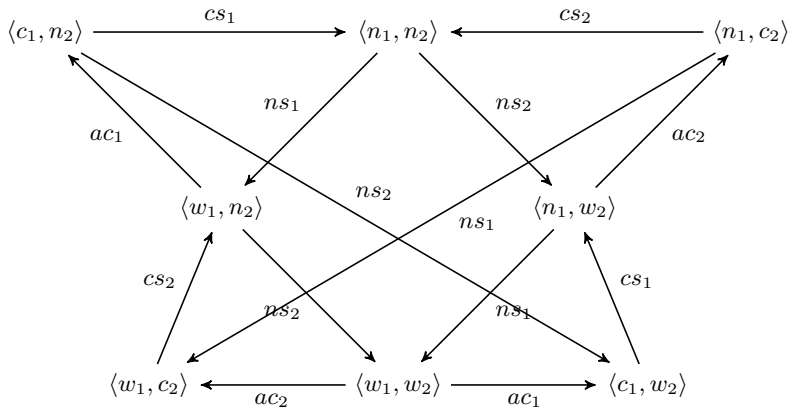
In a concurrent program, a process cycles continuously through two sections of code. The first section, denoted by n, is noncritical, whereas the second section, denoted by c, is critical, i.e., it is required that at most one process may access it. Before executing the critical section, the process visits a state, denoted by w, where it waits for access to the critical section. Informally, an execution path of a process is therefore

$$n \to w \to c \to n \to w \to c \to \cdots .$$

Access to the critical section is granted by a scheduler which may pick any of the waiting processes nondeterministically.

a) Define a suitable LTS that models the program with two processes. (Hint: you may want to label a state with $\langle s_1, s_2 \rangle$, where $s_i \in \{n_i, w_i, c_i\}$, for $i = 1, 2$.)

Solution: We assume that in the initial state all processes are accessing the noncritical section.



Comments:

- The interpretation of the action labels is as follows

  ns_i means "execution of the noncritical section of process i"

  ac_i means "access to the critical section by process i"

  cs_i means "execution of the critical section for process i"
- The states $\langle n_1, n_2 \rangle$, $\langle w_1, n_2 \rangle$, $\langle w_1, w_2 \rangle$, $\langle n_1, w_2 \rangle$ form the classic *concurrency diamond*. With labelled transition systems, the semantics of concurrent processes is *interleaving*—the two processes may finish executing their noncritical sections in either order.
- In any state there is always a nondeterministic choice, except for the states where one process is in the critical section and the other is waiting, i.e., $\langle c_1, w_2 \rangle$ and $\langle w_1, c_2 \rangle$—we may only see the former finishing the execution of the critical section.

b) Execution might not be fair. There are infinite paths starting at $\langle w_1, w_2 \rangle$ where thread 1 never enters the critical section. Specify one such path.

Solution:

$$\underline{\langle w_1, w_2 \rangle} \xrightarrow{ac_2} \langle w_1, c_2 \rangle \xrightarrow{cs_2} \langle w_1, n_2 \rangle \xrightarrow{ns_2} \underline{\langle w_1, w_2 \rangle} \xrightarrow{ac_2} \langle w_1, c_2 \rangle \xrightarrow{cs_2} \dots$$

c) Define a suitable LTS that models the program with three processes (only a sketch is fine).

Solution: Label a state with the triple $\langle s_1, s_2, s_3 \rangle$, with $s_i \in \{n_i, w_i, c_i\}$, for $i = 1, 2, 3$ and proceed as in Question 1. (This model has $27 - (1 + 3 \cdot 2) = 20$ states.)

d) Can you give an upper bound to the total number of states of the LTS that models the program with *twenty* processes?

Solution: An upper bound may be obtained by considering that, in the worst case, each process may be in any local state regardless of the state of the other processes (i.e., ignoring mutual exclusion for the shared resource). Since each process may visit 3 states, an upper bound for the total number of states is 3^{20}. *This is an instance of the problem of state-space explosion—the number of states grows exponentially with the number of components in the system.*

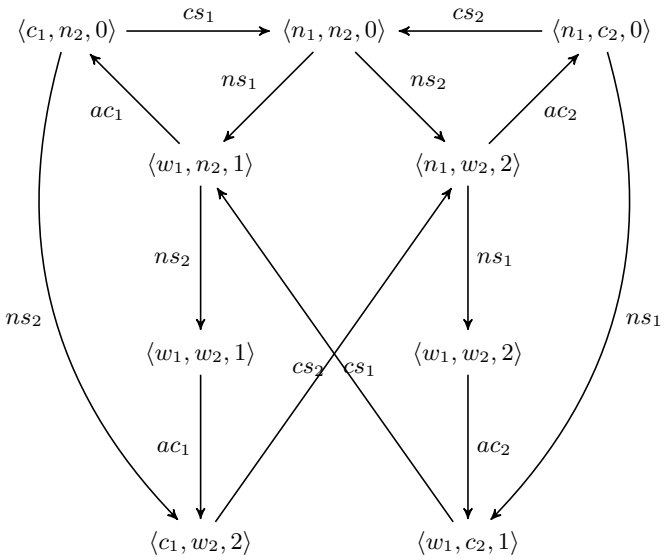
e) Suppose now that the scheduler serves the waiting processes according to a first-come first-served policy. Define a suitable LTS that models the program with two processes.

Solution: The state representation for this LTS is a triple $\langle s_1, s_2, q \rangle$, where s_1 and s_2 are as before (i.e., they model the state of each program) and $q \in \{0, 1, 2\}$ models the state of the scheduler.

- $q = 0$ indicates that there are no processes waiting for access to the critical section.
- $q = 1$ indicates that process 1 has come first.

- $q = 2$ indicates that process 2 has come first.

The LTS is graphically depicted below. Notice that, essentially, this LTS resolves the nondeterministic behaviour of the scheduler by *splitting* the state in which the two processes are both waiting into two distinct states $\langle w_1, w_2, 1\rangle$ and $\langle w_1, w_2, 2\rangle$.



Comparing this model with the LTS of (a), it is interesting to note that this is a fairer system. In this LTS, a process may be waiting forever without getting access to the critical section (see (b)). Instead, in the FIFO system, if one starts from $\langle w_1, w_2, 1\rangle$ then, after ac_1 is observed, it is not possible to observe another ac_1 without having observed a ac_2 before. (This will be formalised problem when we discuss temporal logic specifications.)