Cyber-Physical Programming

TPC-1 Submission

Henrique Gabriel dos Santos Neto
pg47238@alunos.uminho.pt

Exercise 1. Consider the CCS process $c.(a.0 \parallel b.0)$.
Part 1.1. Informally describe what it does.
Part 1.2. Write its transition system using the semantics provided in the lectures.

Part 1.1.

After receiving a message from channel c, the process will receive 2 messages in parallel from the channels a and b and finish it's execution.

Part 1.2.

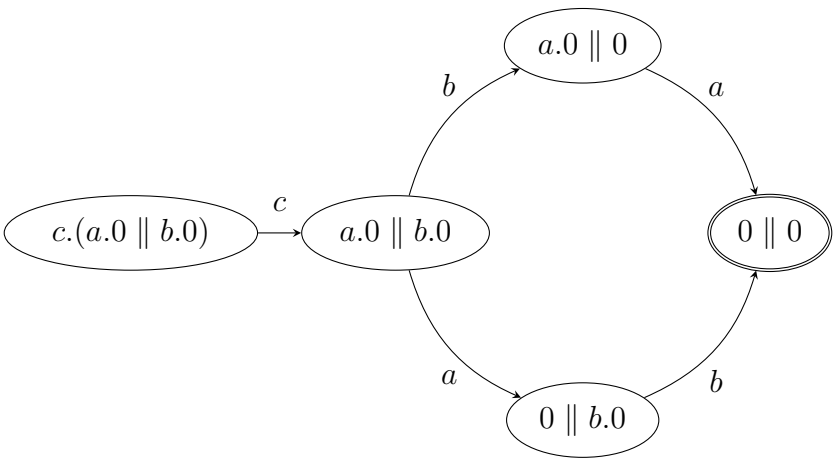
First we need to calculate the system's states and transitions.

$$\frac{}{c.(a.0 \parallel b.0) \xrightarrow{c} a.0 \parallel b.0}\, pre$$

$$\frac{\dfrac{}{a.0 \xrightarrow{a} 0}\, pre}{a.0 \parallel b.0 \xrightarrow{a} 0 \parallel b.0} \parallel_1 \qquad \frac{\dfrac{}{b.0 \xrightarrow{b} 0}\, pre}{0 \parallel b.0 \xrightarrow{b} 0 \parallel 0} \parallel_2$$

$$\frac{\dfrac{}{b.0 \xrightarrow{b} 0}\, pre}{a.0 \parallel b.0 \xrightarrow{b} a.0 \parallel 0} \parallel_2 \qquad \frac{\dfrac{}{a.0 \xrightarrow{a} 0}\, pre}{a.0 \parallel 0 \xrightarrow{a} 0 \parallel 0} \parallel_1$$

With the calculated states and transitions, we can visualize the system as an automaton.



Exercise 2. Consider the *CCS* processes $recX.(a.X + a.a.X)$ and $recX.a.X$.
Part 2.1. Informally describe what they do.
Part 2.2. Prove that $recX.(a.X + a.a.X) \sim recX.a.X$.

Part 2.1.

Both processes receive messages from the channel a indefinitely.

Part 2.2.

We begin by calculating the states and transactions of $recX.(a.X + a.a.X)$

$$\dfrac{\dfrac{\dfrac{}{a.X[recX.(a.X + a.a.X)/X] \xrightarrow{a} recX.(a.X + a.a.X)}\,pre}{(a.X + a.a.X)[recX.(a.X + a.a.X)/X] \xrightarrow{a} recX.(a.X + a.a.X)}\,+}{recX.(a.X + a.a.X) \xrightarrow{a} recX.(a.X + a.a.X)}\,rec$$

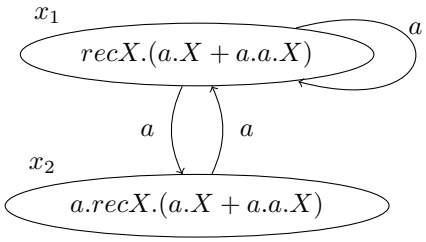
$$\dfrac{\dfrac{\dfrac{}{a.a.X[recX.(a.X + a.a.X)/X] \xrightarrow{a} a.recX.(a.X + a.a.X)}\,pre}{(a.X + a.a.X)[recX.(a.X + a.a.X)/X] \xrightarrow{a} a.recX.(a.X + a.a.X)}\,+}{recX.(a.X + a.a.X) \xrightarrow{a} a.recX.(a.X + a.a.X)}\,rec$$

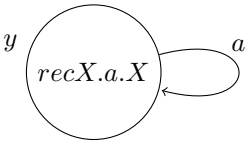
$$\dfrac{}{a.recX.(a.X + a.a.X) \xrightarrow{a} recX.(a.X + a.a.X)}\,pre$$

Following up, we calculate the state and transaction of $recX.a.X$.

$$\dfrac{\dfrac{}{a.X[recX.a.X/X] \xrightarrow{a} recX.a.X}\,pre}{recX.a.X \xrightarrow{a} recX.a.X}\,rec$$

We then arrive on the following transition systems.





To simplify the rules studied on the following step, the states $recX.(a.X + a.a.X)$, $a.recX.(a.X+a.a.X)$ and $recX.a.X$ will now be referenced as x_1,x_2 and y respectfully.

In order to prove that $x_1 \sim y$ (i.e. $recX.(a.X + a.a.X) \sim recX.a.X$) we must guarantee that for every transition from x_1 to another state there exists an equivalent state achievable from the same transition on y and vice-versa. To do this we simply have to exhaust all the possible transitions on both systems.

$x_1 \sim y$

1. $x_1 \xrightarrow{a} x_1 \implies y \xrightarrow{a} y \wedge x_1 \sim y$
2. $x_1 \xrightarrow{a} x_2 \implies y \xrightarrow{a} y \wedge x_2 \sim y$
3. $y \xrightarrow{a} y \implies x_1 \xrightarrow{a} x_1 \wedge x_1 \sim y$

As we can see on expression 1 and 3, we conclude that $x_1 \sim y \implies x_1 \sim y$ which is trivial and therefore we prove that for this set transitions, the states preform similarly. Finally, on expression 2 we conclude $x_1 \sim y \implies x_2 \sim y$, therefore to be conclude that the transitions systems are equivalent we must prove that $x_2 \sim y$.

$x_2 \sim y$

4. $x_2 \xrightarrow{a} x_1 \implies y \xrightarrow{a} y \wedge x_1 \sim y$
5. $y \xrightarrow{a} y \implies x_2 \xrightarrow{a} x_1 \wedge x_1 \sim y$

When analysing $x_2 \sim y$ we arrived at the concluding $x_2 \sim y \implies x_1 \sim y$. Additionally, from the previous analysis we had come to the conclusion that $x_1 \sim y \implies x_2 \sim y$. Given the transient properties of an implication, we can conclude once again that the triviality that $x_1 \sim y \implies x_2 \sim y \implies x_1 \sim y$. Finally, with all the possible transitions from both states verified and validated, we can finally conclude that $x_1 \sim y$ $(recX.(a.X + a.a.X) \sim recX.a.X)$.

Exercise 3. (Hard). Prove that for all CCS processes P and Q we have $P \parallel Q \sim Q \parallel P$.

Similar to what we did in the previous question, to prove that the states are equivalent $(P \parallel Q \sim Q \parallel P)$ we will prove that for every possible transition α we can verify two statements, these being:

(a) $P \parallel Q \xrightarrow{\alpha} W \implies Q \parallel P \xrightarrow{\alpha} W' \wedge W \sim W'$
(b) $Q \parallel P \xrightarrow{\alpha} W \implies P \parallel Q \xrightarrow{\alpha} W' \wedge W \sim W'$

Beginning with statement (a), given the properties of the parallelism we have three possible statements to prove:
1) P can perform a transaction through a channel:

$$P \parallel Q \xrightarrow{\alpha} W \implies P \xrightarrow{\alpha} P' \wedge W = P' \parallel Q \qquad \frac{\dfrac{}{P \xrightarrow{\alpha} P'}\, pre}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}\, \parallel_1$$

$$Q \parallel P \xrightarrow{\alpha} W' \implies P \xrightarrow{\alpha} P' \wedge W' = Q \parallel P' \qquad \frac{\dfrac{}{P \xrightarrow{\alpha} P'}\, pre}{Q \parallel P \xrightarrow{\alpha} Q \parallel P'}\, \parallel_2$$

$$\implies P' \parallel Q \sim Q \parallel P'$$

As we can see we are now in the same state as we started since P' is also abstract processes, meaning we can match it with P and obtain the initial specification of $P \parallel Q \sim Q \parallel P$ proving the equivalency in this branch.

2) Q can perform a transaction through a channel:

$$P \parallel Q \xrightarrow{\alpha} W \implies Q \xrightarrow{\alpha} Q' \wedge W = P \parallel Q' \qquad \frac{\dfrac{}{Q \xrightarrow{\alpha} Q'}\,pre}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'}\,\parallel_2$$

$$Q \parallel P \xrightarrow{\alpha} W' \implies Q \xrightarrow{\alpha} Q' \wedge W' = Q' \parallel P \qquad \frac{\dfrac{}{Q \xrightarrow{\alpha} Q'}\,pre}{Q \parallel P \xrightarrow{\alpha} Q' \parallel P}\,\parallel_1$$

$$\implies P \parallel Q' \sim Q' \parallel P$$

Similarly to previous case, since Q' is abstract we can match it with Q and from also obtain the original statement of $P \parallel Q \sim Q \parallel P$ proving the equivalency in this branch.

3) P and Q can synchronize through a channel:

$$P \parallel Q \xrightarrow{\tau_\alpha} W \implies P \xrightarrow{\alpha} P' \wedge Q \xrightarrow{\overline{\alpha}} Q' \wedge W = P' \parallel Q' \qquad \frac{\dfrac{}{P \xrightarrow{\alpha} P'}\,pre \quad \dfrac{}{Q' \xrightarrow{\overline{\alpha}} Q'}\,pre}{P \parallel Q \xrightarrow{\tau_\alpha} P' \parallel Q'}\,\parallel_\tau$$

$$Q \parallel P \xrightarrow{\tau_\alpha} W' \implies P \xrightarrow{\alpha} P' \wedge Q \xrightarrow{\overline{\alpha}} Q' \wedge W' = Q' \parallel P' \qquad \frac{\dfrac{}{Q' \xrightarrow{\overline{\alpha}} Q'}\,pre \quad \dfrac{}{P \xrightarrow{\alpha} P'}\,pre}{Q \parallel P \xrightarrow{\tau_\alpha} Q' \parallel P'}\,\parallel_\tau$$

$$\implies P' \parallel Q' \sim Q' \parallel P'$$

Just like the previous cases, P' and Q' are abstract processes meaning we can match them with processes P and Q and thus we would be also able to obtain the original expression of $P \parallel Q \sim Q \parallel P$ and thus proving the equivalency in this branch.

Statement (b)'s proof is analogous to the proof of (a).

With both statements proved, we can conclude that $P \parallel Q \sim Q \parallel P$.

Exercise 4. Consider the following scenario. There exist four processes $P_1, \ldots, P_4$, each of them responsible for performing a certain task repetitively. For example $P1$ might read the current velocity, P_2 the current altitude, P_3 current radiation levels, etc ... These processes (re)start their tasks in increasing order (P_1 then P_2 etc ...) but can finish in any order. Additionally process P_1 can only restart its task when all processes $P_1, \ldots, P_4$ finish their current tasks. Let us then consider process $P = (I \parallel S \parallel P_1 \parallel \cdots \parallel P_4) \backslash \{st_1, \ldots, st_4, end\}$ where:

$$
\begin{aligned}
I &= \overline{st_1} \ldots \overline{st_4}.0 \\
S &= rec\ X.\ end.end.end.end.\overline{st_1} \ldots \overline{st_4}.X \\
P_i &= rec\ Y_i.\ st_i.a_i.b_i.\overline{end}.Y_i \qquad (1 \leq i \leq 4)
\end{aligned}
$$

Part 4.1. Explain why process P corresponds (or not) to the description above.

Part 4.2. Note that process S acts a central scheduler which coordinates the processes $P1, \ldots, P4$. Rewrite P so that it does not rely on a central scheduler and explain the reasoning behind your refactoring.

Part 4.3 (Hard). Use the tool *mCRL2* to computationally validate your reasoning process.

Part 4.1.

The process does correspond to process P. In the initial state of execution, processes $P_{1..4}$ and S are awaiting messages from the channels $st_{1..4}$ and *end*. Additionally process I is trying to send a message through st_1. The channels $st_{1..4}$ and *end* are private to P, meaning that the messages can only be transferred within this system. This means that none of the sub-processes can advance spontaneously through the channels $st_{1..4}$ and the *end*, and require a synchronization (τ) with another process when receiving from these channels. So with this is mind, we can conclude that the first thing that happens is a synchronization on processes P_1 and I, which in informal terms means we started the P_1 task. Following this, I will sequentially synchronize with P_2, P_3 and finally $P4$ for the same reasons specified previously (st_1, st_2 and st_3 are private to P) resulting in the start of the remaining tasks. Finally, at the end of all synchronizations I will end its execution (0).

When the processes start, they will perform their respective a and b receptions and finally attempt transmit a message through channel *end*. Since all of the sub-processes are running in parallel the order of theses transactions in non deterministic, resulting in several possible ways of executions. Nonetheless, on each process P_i there will be a transmission of an *end* message once it's tasks is complete. These transmission requires a synchronization with another process because, just like the $st_{1..4}$ channels, *end* is a private channel to the system P. The only possible process that can accept messages from *end* is the scheduler S which at the start of it's execution awaits for four messages from the channel *end*. This way, when every $P_{1..4}$ sub-process finishes their a and b tasks, they will synchronise with process S through

the *end* channel and then await a new message from their respective *st* channel (because of their recursion).

As a result of existing four P_i processes, there will be a total of four *end* messages which will be received by process S. We can guarantee that each process only sends one *end* message in each cycle since, as mentioned before, after transmitting an *end* message each process will stagger while awaiting for a new $st_{1..4}$ message and therefore won't send anymore messages. When all processes are accounted for, the system will be extremely similar to the initial state, since as said previously, the processes $P_{1..4}$ would have restarted their execution and would be awaiting a synchronization on the channels $st_{1..4}$. The difference now is that instead of theses synchronizations occurring with process I, they will occur with process S. After receiving all of the *end* messages, S will synchronize with P_1 P_2 then P_3 and finally $P4$ though the channels $st_{1..4}$, restarting the system, and finishing it's task by restarting it's execution. As a result, this systems guarantees the specifications of the description since all processes restart and the synchronization introduced at the end of all processes guarantees that $P1$ restarts before all the other processes.

Part 4.2.

To remove the central scheduler, a possibility would be to have the processes communicate their *start* and *end* messages in a ring/token like system, were both messages are chained across the processes. This means that in the *st* channels we make a process start the following task as soon as it receives it's starting message. Therefore, after $P1$ starts, it sends a start message to $P2$, which after starting sends it to to $P3$ and so on until the last process has been started.

Additionally, the *end* channel is now divided into multiple end_x channels so that when a process i ends, it will first synchronize with the previous process with it's respective $end_{(i-1)}$ channel and will then proceed to synchronizing with the next process through it's own end_i channel. For this chain of *end* messages to useful, the first and last processes have to be slightly specialized. The first process won't have to wait for a previous one, since there is no previous. The last process won't send a *end* message after synchronizing with it's predecessor and will instead send a start message in st_1 to P_1 in order to restart the process. As a result we can guarantee that the system start each process by the same order as before (P_1, P_2,P_3 and finally P_4) and we can also guarantee that for P_1 to receive a new start message and as a result restart the entire system, all of the other processes would need to have finished and synchronized through their *end* channels. The systems is as follows:

$$
\begin{aligned}
I &= \overline{st_1}.0 \\
P_1 &= rec\ X.\ st_1.\overline{st_2}.a_1.b_1.\overline{end_1}.X \\
P_i &= rec\ Y_i.\ st_i.\overline{st_{(i+1)}}.a_i.b_i.end_{(i-1)}.\overline{end_i}.Y_i \qquad (i = 2 \vee i = 3) \\
P_4 &= rec\ Z.\ st_4.a_4.b_4.end_3.\overline{st_1}.Z
\end{aligned}
$$

Process P would then be:

$$P = (I \parallel P_1 \parallel P_2 \parallel P_3 \parallel P_4) \backslash \{st_1, st_2, st_3, st_4, end_1, end_2, end_3\}$$

Part 4.3.

In order to validate the previous statements, the modified system was specified in *mCRL2*. The resulting specification is presented in fig. 1.

In *mCRL2* we cannot specify channel the same way we do in *CCS*. Instead to emulate the behaviour we have to specify each synchronizing channel (st_i and end_i) with three actions which were define with a pattern. The first contains an underline before the channels name ($\_st_i$ and $\_end_i$), and signifies an input from the channel. The second contains an underline after the channels name ($st_i\_$ and $end_i\_$), and signifies an output to the channel. The third only contains the channel name and consists of the synchronization of the input and output actions. The program is then specified only to allow the synchronizing actions in addition to the jobs a_i and b_i.

```
act
 st1, st2, st3, st4, _st1, _st2, _st3, _st4, st1_, st2_, st3_, st4_,
 end1, end2, end3, end1_, end2_, end3_, _end1, _end2, _end3,
 a1,a2,a3,a4, b1,b2,b3,b4;

proc
 I  = st1_;
 P1 = _st1 . st2_ . a1 . b1          . end1_ . P1;
 P2 = _st2 . st3_ . a2 . b2 . _end1 . end2_ . P2;
 P3 = _st3 . st4_ . a3 . b3 . _end2 . end3_ . P3;
 P4 = _st4        . a4 . b4 . _end3 . st1_  . P4;

init
allow(
   {st1, st2, st3, st4, end1, end2, end3, a1, a2, a3, a4, b1, b2, b3
   , b4},
   comm(
    { _st1|st1_ -> st1, _st2|st2_ -> st2,_st3|st3_ -> st3,_st4|st4_
   -> st4, _end1|end1_ -> end1, _end2|end2_ -> end2, _end3|end3_ ->
   end3},
    I || P1 || P2 || P3 || P4
    )
);
```

Figure 1: *mCRL2* specification

The tool then allows us to test properties, simulate system runs and also draw the system's automaton. On a system of this scale the drawings are nearly impossible to read, so we will rely mostly on the properties tools and simulations. Nonetheless, a drawing of the automaton is still provided in fig. 2 in order to give the general view of the process.

In other to prove the specified requisites we will use *mCRL2* notation for $\mu - calculus$ in order to specify and prove each desired property. We will consider that a process P_i starts after receiving it's st_i message and ends after receiving it's b_i message (which in processes $P_{2..4}$ consists of waiting or synchronising on the channel $end_{(i-1)}$).

Well begin by proving that the processes start in the other P_1, P_2, P_3 and finally P_4. In other terms, we need to guarantee that st_2, st_3 and st_4 occur before st_1, that st_3 and st_4 occur before st_2 and finally that st_3 occurs before st_4. We can specify this with the expression:

```
[!st1*.st2]false && [!st1*.st3]false && [!st1*.st4]false &&
    [!st2*.st3]false && [!st2*.st4]false && [!st3*.st4]false
```

As we can see, there is a recurring patterns of `[!a*.b]false` which specifies that all finite paths (referenced by the `[]` brackets) consisting of several possible actions different that a ending with a b action are invalid (`false`).

Next, we shall prove that P_1 only restarts after all other tasks have ended. To describe this we can specify an expression that dictates that st_1 can only happen after a st_i action $(2 \leq i \leq 4)$ if it's corresponding $end_{(i-1)}$ has occurred. This way we arrive at the following expression.

```
[true*.st2.!end1*.st1]false && [true*.st3.!end2*.st1]false &&
    [true*.st4.!end3*.st1]false
```

Just like the previous statement we can also see a pattern, this time consisting of `[true*.b.!c*.a]false`. This pattern specifies that all finite paths which performe several possible arbitrary actions (`true*`) followed by action b followed by several possible actions different from c ($!c*$) that end with action a are invalid (`false`).

Finally we need to prove that every process can finish in any order. This proof can be extensive, since if solve directly it would required us to specify that all 4! possible orders the processes can finish are valid. However we can instead prove that all runs where each process finishes (i.e action b occurs) is valid since this will include these cases where the processes finish out of order and as such will imply the previous statement. The resulting expression is:

```
[true*.b1.true*]true && [true*.b2.true*]true &&
    [true*.b3.true*]true && [true*.b4.true*]true
```

We can provide each expression to the tool through it's IDE or using console commands. When using the IDE we grant a name to the property we want to prove and as result, if the statement is valid the tool will present the message

`The property <name> on this specification evaluates to true`

and present a possible witness to the statement. When using the console we must first covert the property to a *Parameterised Boolean Equation System* (PBES) using the *mCRL2*'s package tool `lps2pbes` and finish by using a `pbes2bool` to validate the PBES statement. If the property is valid the message "`The pbes is valid`" will be presented.

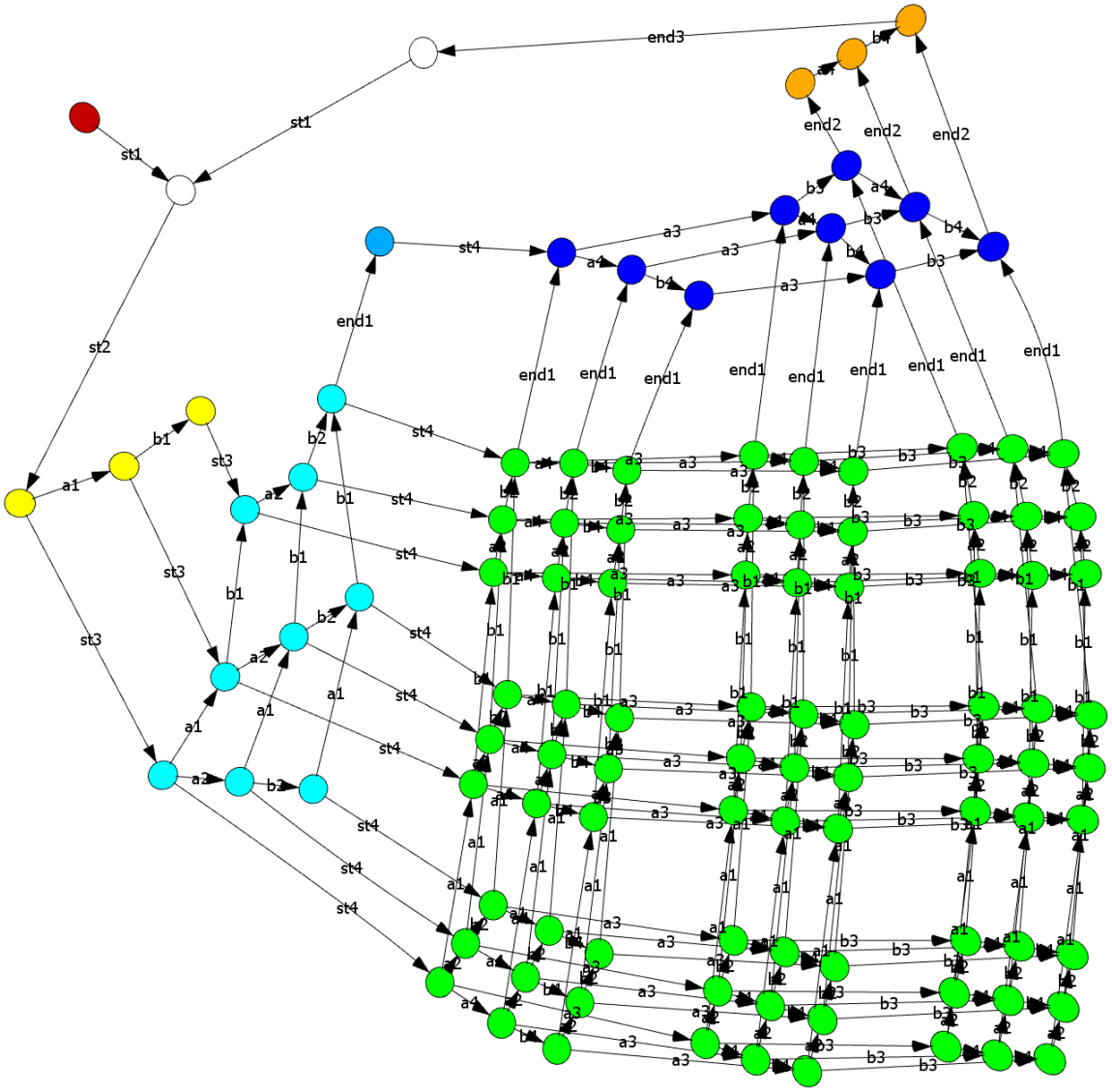


Figure 2: Automaton

The states have been colored in order to make analysing it easier. Every state after st_2 (i.e P_2 has been start) has been colored yellow, then every state after st_3 has been colored cyan. On the order hand every state after end_1 (i.e P_2 has synchronized with P_1) has been colored dark blue and every state after end_2 has been colored dark orange. Additionally we can see a red state, which corresponds to the initial state of the system, and several green states which correspond to all of the possible cases were all the processes have started and are running in parallel or waiting for a synchronization from the previous process.

With this drawing, it is possible after an extensive analysis to in fact see some of the previously specified properties. For example, if we focus on all the start messages which preform color changes on the states, we can observe that the order of in which the processes are started is correct, etc. However it's evidently what had been said. It is extremely difficult and impractical to analyse a graph of this scale and so, it is not reliable.