MODELING AND SIMULATION

Report

on

Design and Validation of DES
Petri net modeling

Authors:

Wendi Ding

and

Neel Chakravarty

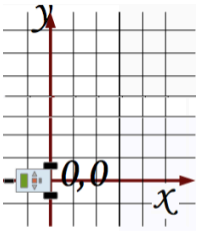
1. Specification

The goal of this section is to model the control of a robot that will follow a list of way points and catch balls in its way. The robot, called Navi, was conceived for the purposes of terrestrial navigation and has several components: wheels, a digger, a light sensor and a compass sensor, 3 actuators (2 for the tractor wheels and one for control the digger). Its controller is a token player that allows the execution of a Petri net model. To realise the desired functions, a file with a list of way points must also be provided.

a) Navi

B,25.0,15.0
L,70.0,80.0
W,60.0,20.0
W,60.0,35.0
W,25.0,30.0
W,25.0,45.0
W,60.0,50.0

b) List of points



c) Lego on the sheet

Figure 1: The robot for terrestrial navigation through points

1.1 Simplified model: The robot only follows some way points

In this simplified model, the robot can follow some way points along a path; a section is defined as a path between two consecutive way points, from a way point W_i to the next way point W_{i+1}. Two particular points are defined: B (for base) and L (for the delivery point). The robot starts from an initial point I_0.

Fig. 2 shows a first version of the PN model that verifies all the properties and correctly represents the specification. The PN model represents the following statements: before moving through the base and the way points, an initialization is executed, mainly for calibrating the compass and loading the file WPlist, which is loaded only once. Then the robot can read the coordinates of the next way point using function ReadWP. After reading and checking, if it is not the last way point, the robot will move to the next way point and when arrived, it reads the WPlist again in order to check the next way point, otherwise, if it is the last way point, the robot will come back to its base B and stop.

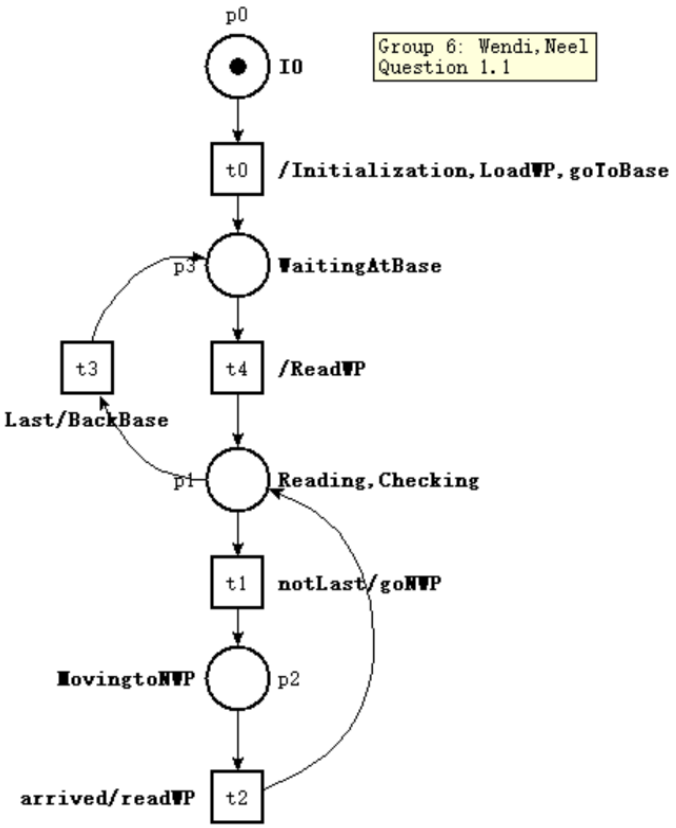


Figure 2: The Petri net for the simplified model

1.2 Improved model: The robot must catch balls

On its way from the base (point B) to the final way point, if the robot detects a ball (the light sensor value is more than 40) in a section $[W_i, W_{i+1}]$, then it lowers its digger to catch the ball (without stopping) and, when it arrives at point W_{i+1}, it brings the ball to the delivery point L. When arriving in L, it raises the digger to release the ball and then comes back to W_{i+1}. If the robot goes through a section without finding any ball, it goes on to the next way point in the list and so on. When arriving at the last way point of the list (it has completed the navigation through all way points), it comes back to the base and stop. Fig. 3 shows the Petri net for the improved model.

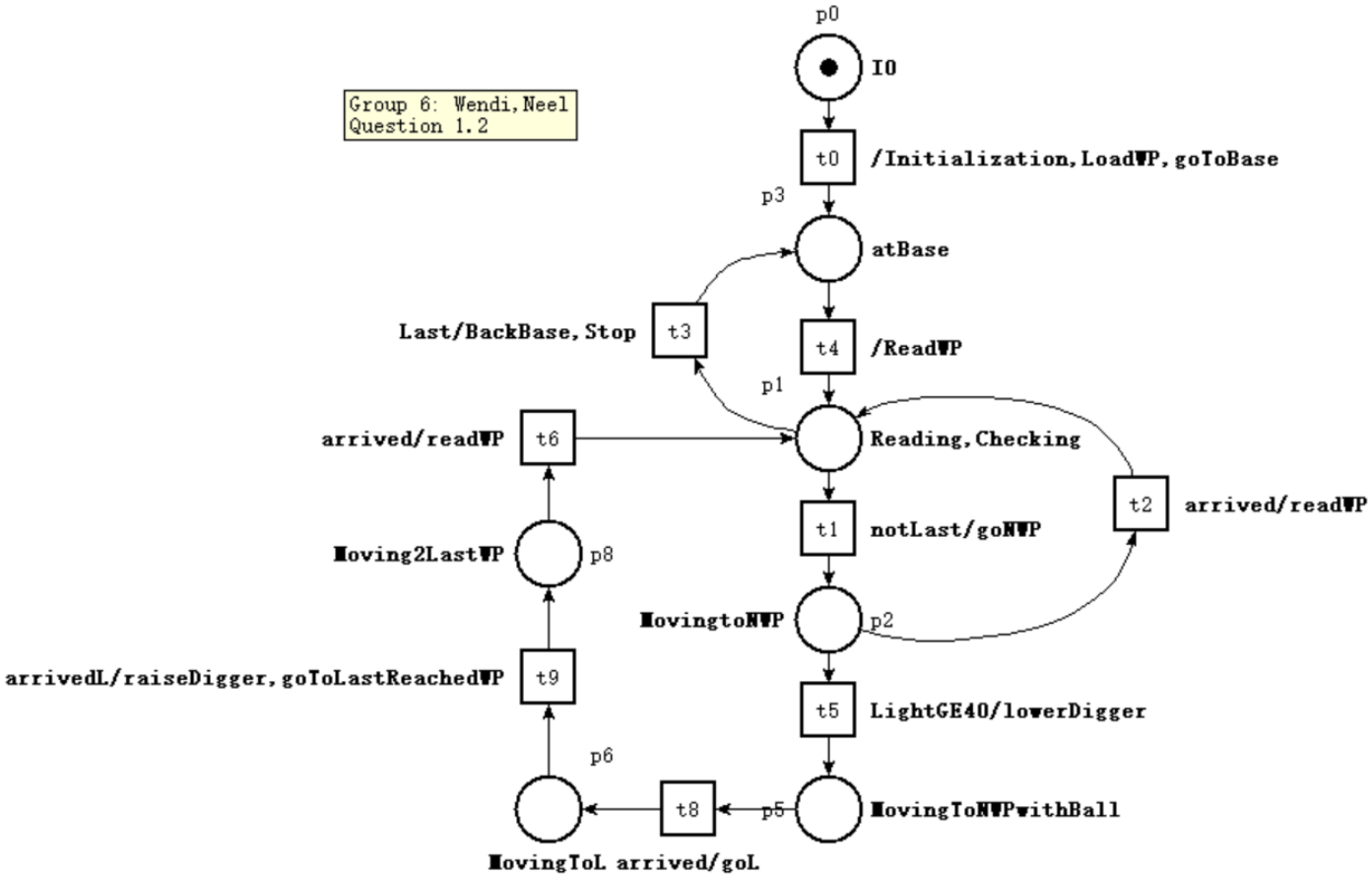


Figure 3: The Petri net for the improved model

A list of all the events that may occur in the system, as well as their pre and post-conditions, are shown in Table 1 below.

Table 1: A list of all the events that may occur in the system with pre and post-conditions

Event	Pre-Conditions	Post-Conditions
Initialization	The robot is started.	The robot is initialized and calibrated, the WPlist is loaded.
ReadWP	The WPlist is loaded. The robot has arrived at its target WP/base.	The next WP is read.
goNWP	The next WP is read and is not the last WP.	The robot moves to the next WP.

BackBase	The next WP is read and the last WP was reached.	The robot moves back to the base B.
lowerDigger	The robot is moving to nextWP and the value of light sensor is greater than 40.	The robot lowers its digger and then moves with the ball.
goL	The robot has arrived at the next WP with a ball.	The robot moves to the delivery point L with the ball.
raiseDigger	The robot has arrived at the delivery point L.	The robot raises its digger in order to leave the ball, then goes back to the last reached WP.
goToLastReachedWP	The robot has arrived at the delivery point L.	The robot moves back to the last reached WP.

2 Lab class - Analysis of the Petri net

The main goal of this section is the analysis of a Petri net. In this section, the PN model is verified and validated using formal methods such as state space analysis and structural analysis, as well as simulation.

2.1 The Mindstorm Lego: interaction with the environment

Navi has a brick Lego NXT, that is the brain of a MINDSTORMS robot and contains the token player that plays the Interpreted Petri net (IPN) model verifying the *cond* conditions and executing the *act* actions, by calling the corresponding predefined programs dealing with the Java classes concerning the sensors and actuators of the NXT brick, which are described on Table 2.

Table 2: Conditions and actions for the robot interaction with the environment

	Function name (LeJos syntax)	`tina` label	Description	Value
conditions	PRISE	caughtLow	ball caught and digger lowered	True/1
	LUMIERE1	noBallDetected	light sensor level $< 40\%$	0-40
		BallDetected	light sensor level $\geq 40\%$	40-100
	ACTION_EN_COURS	noAction	number of actions currently running	0
	FIN_MVMT	arrivedWP	navigation has finished	True/1
	DERNIER_WP	lastWP	last way point reached	True/1
		hasNextWP	last way was not point reached	False/0
	BT_i	recBTi	message bluetooth `i` received, $i \in \{1..4\}$	True/1
		NotRecBTi	message bluetooth `i` is not received	False/0
actions	LEVER_BRAS	riseArm	rise the digger	–
	BAISSER_BRAS	lowerArm	lower the digger	–
	EXIT	exit	stop token player	–
	INIT_NAVE	initNave	init. navigation (calib. compass, read .wp)	–
	ETALON_CAPT_LUMIERE	calibLightSensor	calibrate the light sensor	–
	GO_LAST_RD_WP	goBackRdWP	go to the last Reached WP	–
	GO_LIVRAISON	goDeliveringPt	go to delivering point	–
	GO_NEXT_WP	goNextWP	go to the next way point	–
	GO_BASE	goBase	go to base	–
	BIPER	bip	emit a sound	–
	ENVOI_BT_i	sendBTi	send bluetooth message BT_i, $i \in \{1..4\}$	–

2.2 Petri net using the Lego functions

Using only the tina labels described in Table 1, we redesign our Petri net, as is shown in fig. 4.

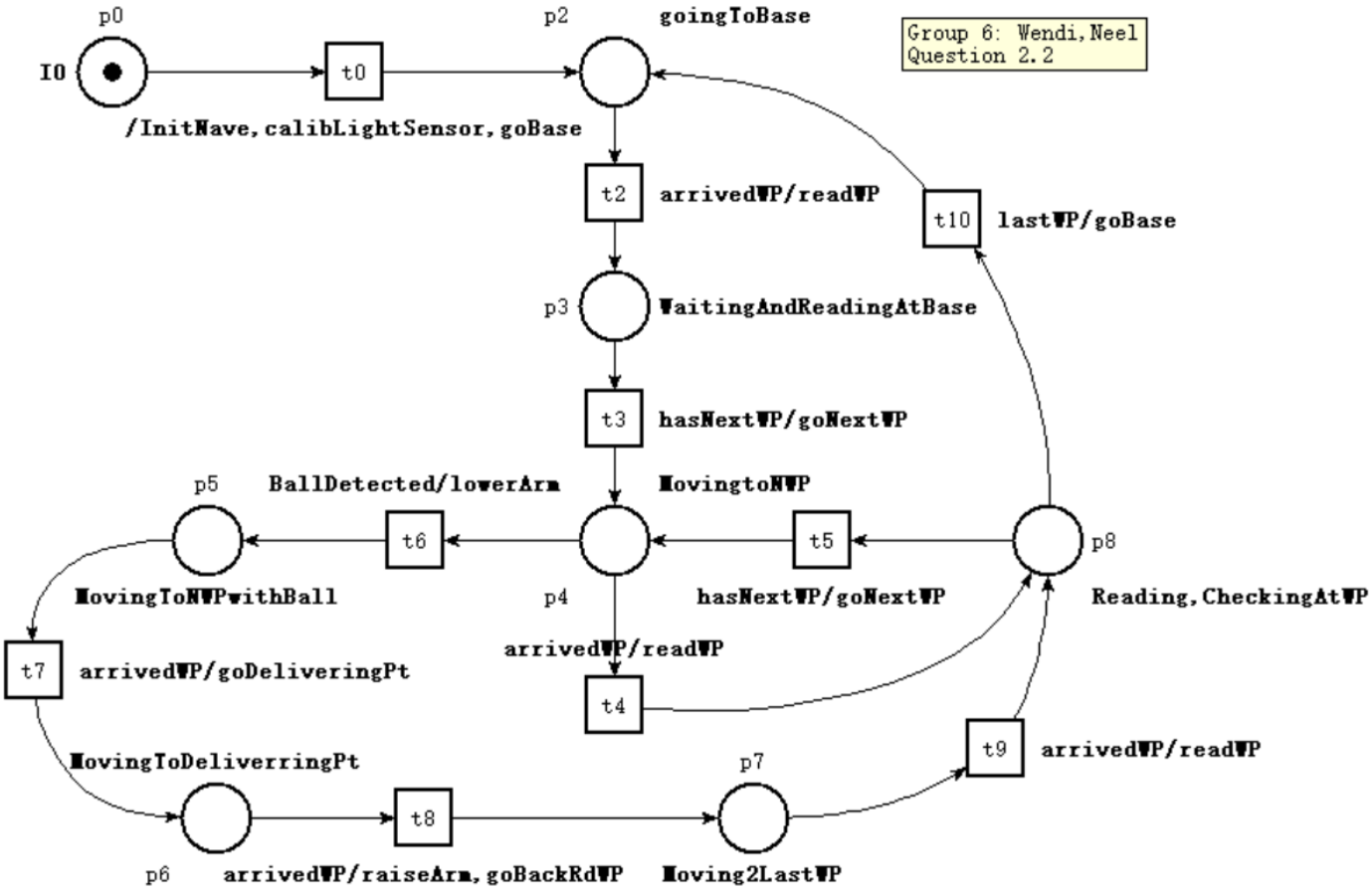


Figure 4: Redesigned Petri net using only the tina labels described in Table 1

2.3 Analysis of the Petri net

2.3.1 Model0

For the model designed on section 2.2 in fig. 4, we verify it in tina using Tools/State Space Analysis with the verbose option. From the general conclusion of state space analysis in fig. 5, we can see that the Petri net is bounded, however it is not live and not reversible. Taking a closer look at the general conclusion, we find that 7 from the 8 places and 9 from the 10 transitions are live, and that no places or transitions are dead. Then the marking graph in fig. 6 (right) shows a bounded net, with all transitions

appearing at least once on the arcs. We notice that transitions t2-t10 are live while t0 is quasi-live. Since t0 corresponds to the initialization and calibration actions of the robot in the Petri net, and in reality the robot is not supposed to go back to the initial state p0 after finishing all the tasks, we conclude that the quasi-liveness of t0 is acceptable.

places	8	transitions	10	net	bounded	Y	live	N	reversible	N

abstraction	count	props	psets	dead	live
states	8	8	?	0	7
transitions	10	10	?	0	9

Figure 5: General conclusion of state space analysis

```
P-SEMI-FLOWS GENERATING SET

invariant

p0 p2 p3 p4 p5 p6 p7 p8 (1)

0.000s

T-SEMI-FLOWS GENERATING SET

not consistent

t10 t2 t3 t6 t7 t8 t9
t10 t2 t3 t4
t5 t6 t7 t8 t9
t4 t5
```

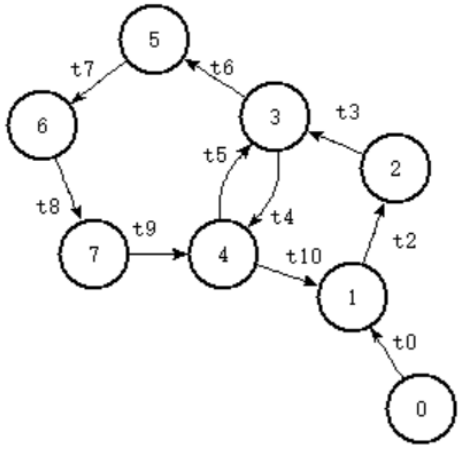


Figure 6: Structural analysis (left) and the marking graph (right) of the Petri net

From the structural analysis (Tools/State Space Analysis with the lts (.aut) option) in Figure 6 (left), we can see that all the positions appear at least once as part of a conservative component, verifying that the Petri net is bounded.

Since the quasi-liveness of t0 is acceptable, it is not necessary to correct the error. So we just make another Petri net model excluding p0 and t0 in a bid to verify that the model without initialization step is bounded, live and reversible, as demonstrated in the next subsection 2.3.1 Model1.

2.3.2 Model for testing

Figure 7 shows a Petri net model for testing, excluding p0 and t0, in order to verify that the model without initialization step is bounded, live and reversible.

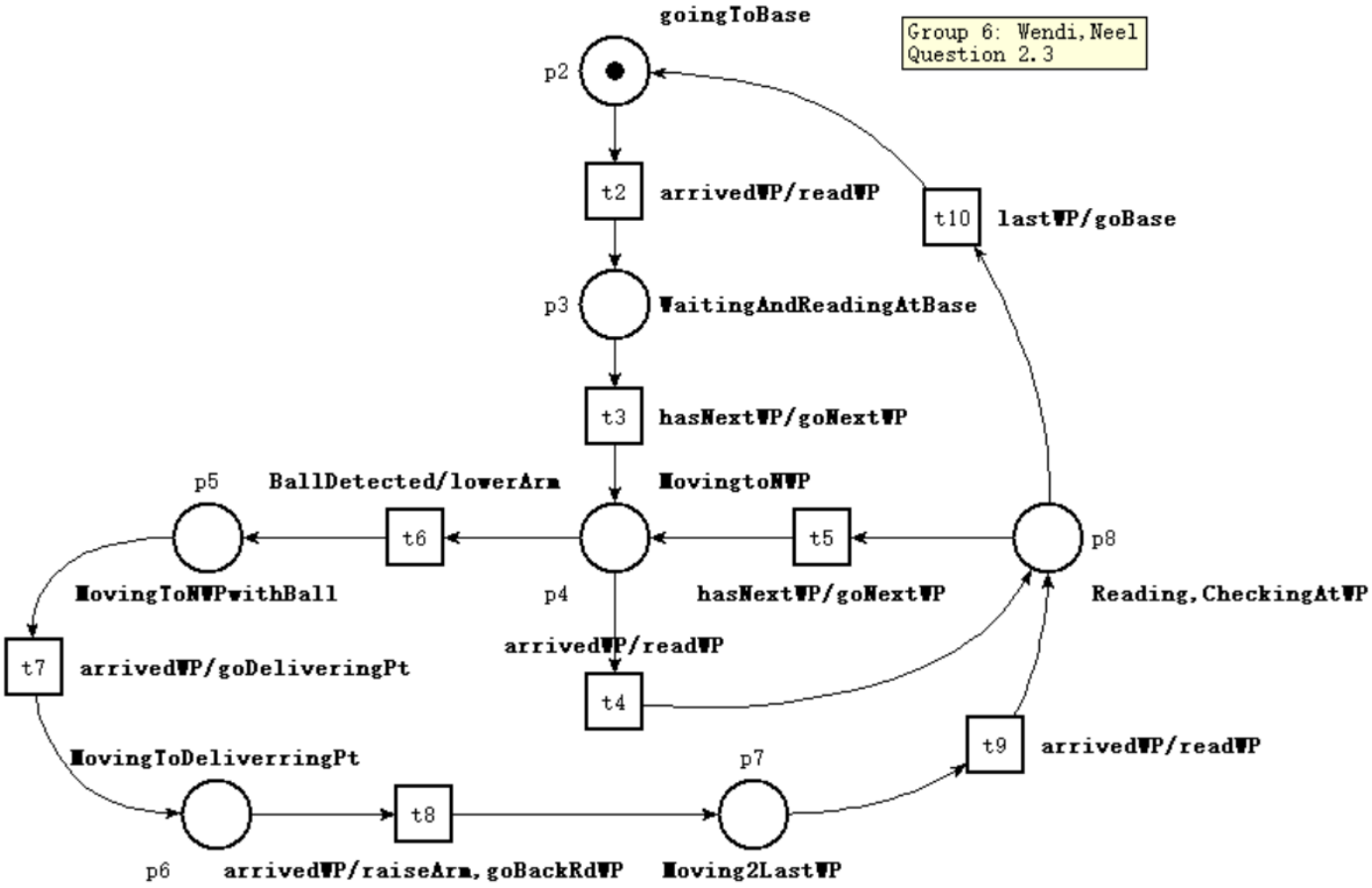


Figure 7: Testing Petri net for verification

The initial marking is set as p2=1. Running the state space analysis and structural analysis, we obtain the following results as shown in fig. 8-9. The analyses show that all states and transitions are live (see general conclusion of state space analysis) and bounded (conservative components of structural analysis and the marking graph). The marking graph also demonstrates that all the transitions of the Petri net can be enabled repeatedly during the simulation, and that initial marking “0”: p2=1 is reachable from any possible markings through a firing sequence. Therefore, we verify that the petri net is bounded, live and reversible.

places	7	transitions	9	net	bounded	Y	live	Y	reversible	Y

abstraction	count	props	psets	dead	live
states	7	7	?	0	7
transitions	9	9	?	0	9

Figure 8: General conclusion of state space analysis for the testing Petri net

```
P-SEMI-FLOWS GENERATING SET

invariant

p2 p3 p4 p5 p6 p7 p8 (1)

0.000s

T-SEMI-FLOWS GENERATING SET

consistent

t10 t2 t3 t6 t7 t8 t9
t10 t2 t3 t4
t5 t6 t7 t8 t9
t4 t5
```

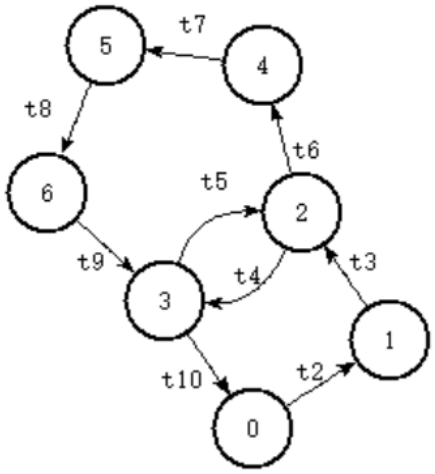


Figure 9: Structural analysis (left) and the marking graph (right) for the testing Petri net

But we have to notice that when verifying a PN model, the labels are not considered. So a Petri net with good properties does not necessarily result in satisfying functions, and we must check the labels and simulate the model in order to validate our Petri net.

2.3.3 Model1

Model0 is modified into Model1 for a functionality consideration. As is shown in fig. 10 we separate the initialization and calibration process and the action going to base B, so that we make sure that initialization, loading of WPs and calibration are finished when the robot starts moving according to the WPlist. In the end, when the last WP is reached and the robot goes back to base B, we add a transition to turn off the robot (exit), so that it will not keep waiting at base all the time when all tasks have been accomplished.

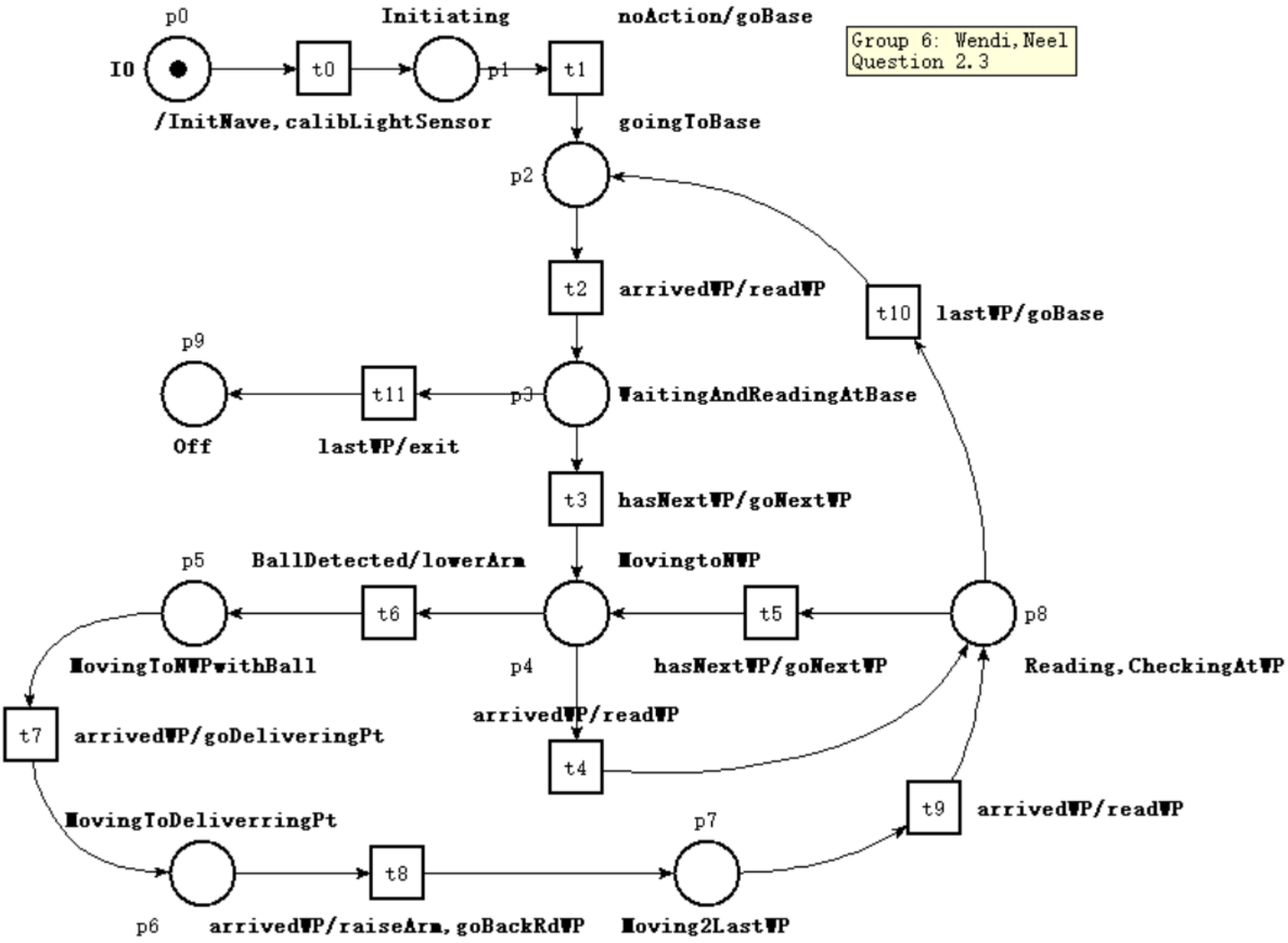


Figure 10: Petri net Model1

places	10	transitions	12	net	bounded	Y	live	N	reversible	N

abstraction	count	props	psets	dead	live
states	10	10	?	1	1
transitions	12	12	?	0	0

Figure 11: General conclusion of state space analysis for Model1

The analyses of Model1 are shown in fig. 11-12. We can see that the Petri net is bounded, not live and not reversible, and that there is one dead state (marking “4”:

p9=1). The reason is that the token can go nowhere when we turn off the robot and it can not recover the initial state without manual interruption.

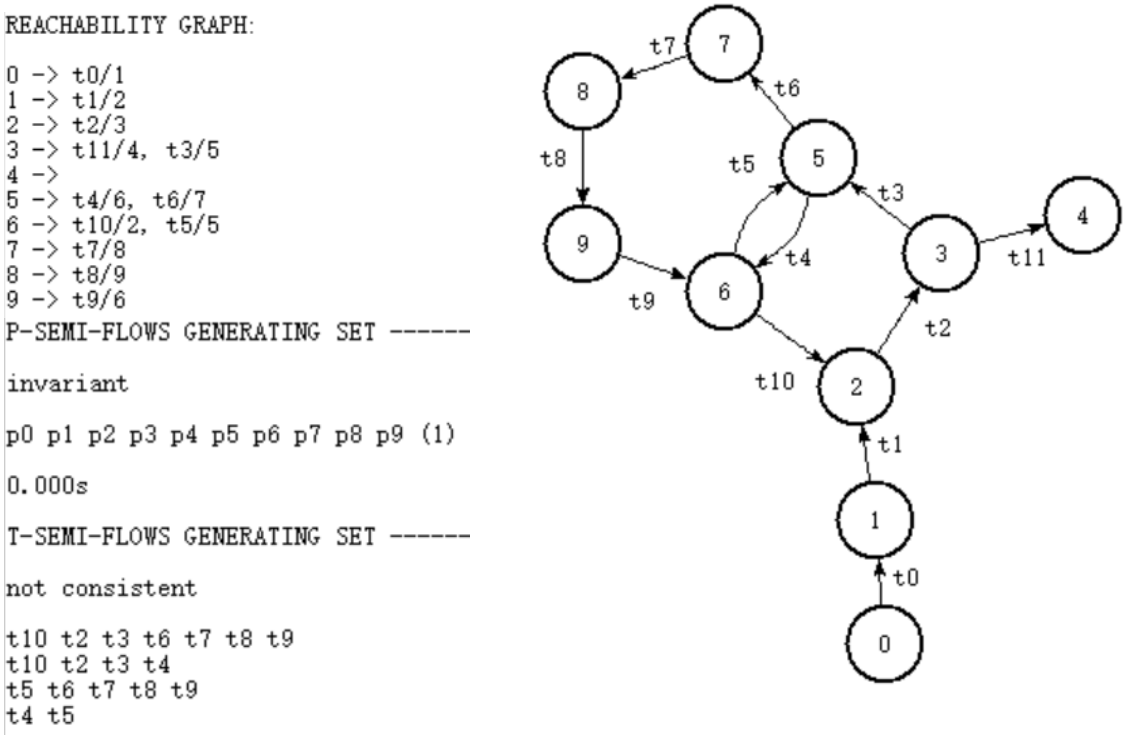


Figure 12: Structural analysis (left) and the marking graph (right) for Model1

2.3.4 Model2

In order to have a bounded, live and reversible Petri net, we can add a manual interruption called Restart which puts the robot back into its initial state: to move the robot from the base B to I0, and to turn it on. The modified model is named Model2 and shown in fig. 13.

Figure 14-15 demonstrate the analysis results of Model2. By now the model is bounded, live and reversible. Any transition can be enabled from any possible marking through a sequence, which is a very satisfying property of Petri net.

However, in reality, since the robot can’t move from base B to I0 and restart itself, transition 12 does not exist in a Petri net functioning as decision and communication

algorithm implemented in a robot, but Model2 can be taken into consideration when we do the validation (2.4) and analysis of cooperation of two robots (2.5).

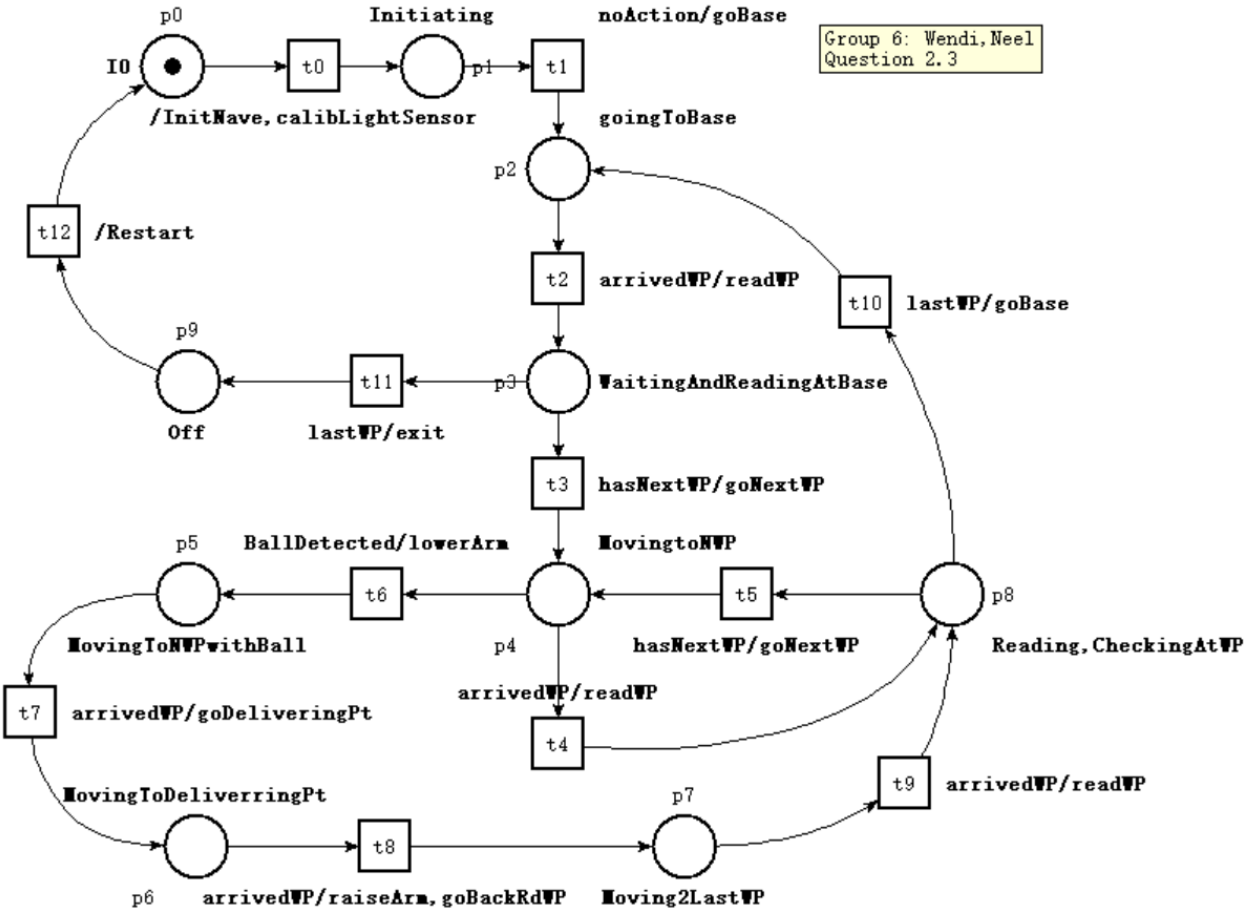


Figure 13: Petri net Model2

places	10	transitions	13	net	bounded	Y	live	Y	reversible	Y

abstraction	count	props	psets	dead	live
states	10	10	?	0	10
transitions	13	13	?	0	13

Figure 14: General conclusion of state space analysis for Model2

```
P-SEMI-FLOWS GENERATING SET ------

invariant

p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 (1)

0.000s

T-SEMI-FLOWS GENERATING SET ------

consistent

t0 t1 t11 t12 t2
t10 t2 t3 t6 t7 t8 t9
t10 t2 t3 t4
t5 t6 t7 t8 t9
t4 t5

0.000s
```

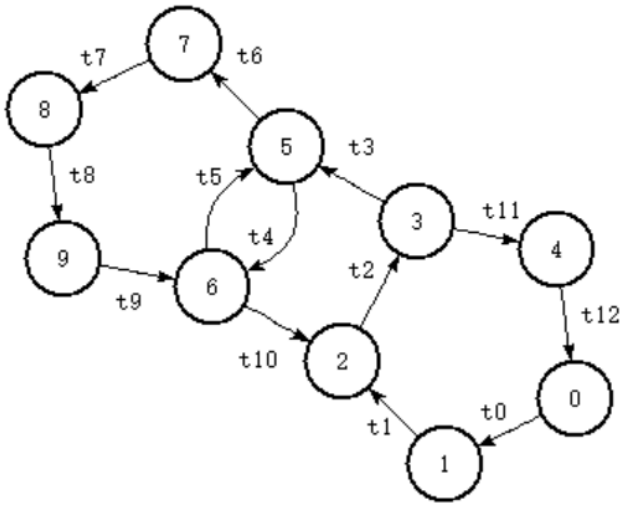


Figure 15: Structural analysis (left) and the marking graph (right) for Model2

2.4 Validate your Petri net

For validating our Petri net, using the stepped simulator in tina, we can simulate a good scenario to ensure that our system matches the specification.

Let us consider a good scenario, where we have a total of four waypoints, and there is a ball present in between the second and the third waypoint. The robot will start at an initial position, and after initialising, it will move towards the base. Then, it must move from the base to the first waypoint, and from the first waypoint to the second waypoint. There is a ball in between the second and third waypoint, so the robot will pick up the ball on its way to the third waypoint. After reaching the third waypoint, the robot will go to the delivery point L to deliver the ball, after which it will return to the third waypoint, and proceed towards the fourth waypoint, which is the last waypoint. So after reaching the fourth waypoint, the robot will return to the base. The initial state of the system is as shown below in Figure 16:

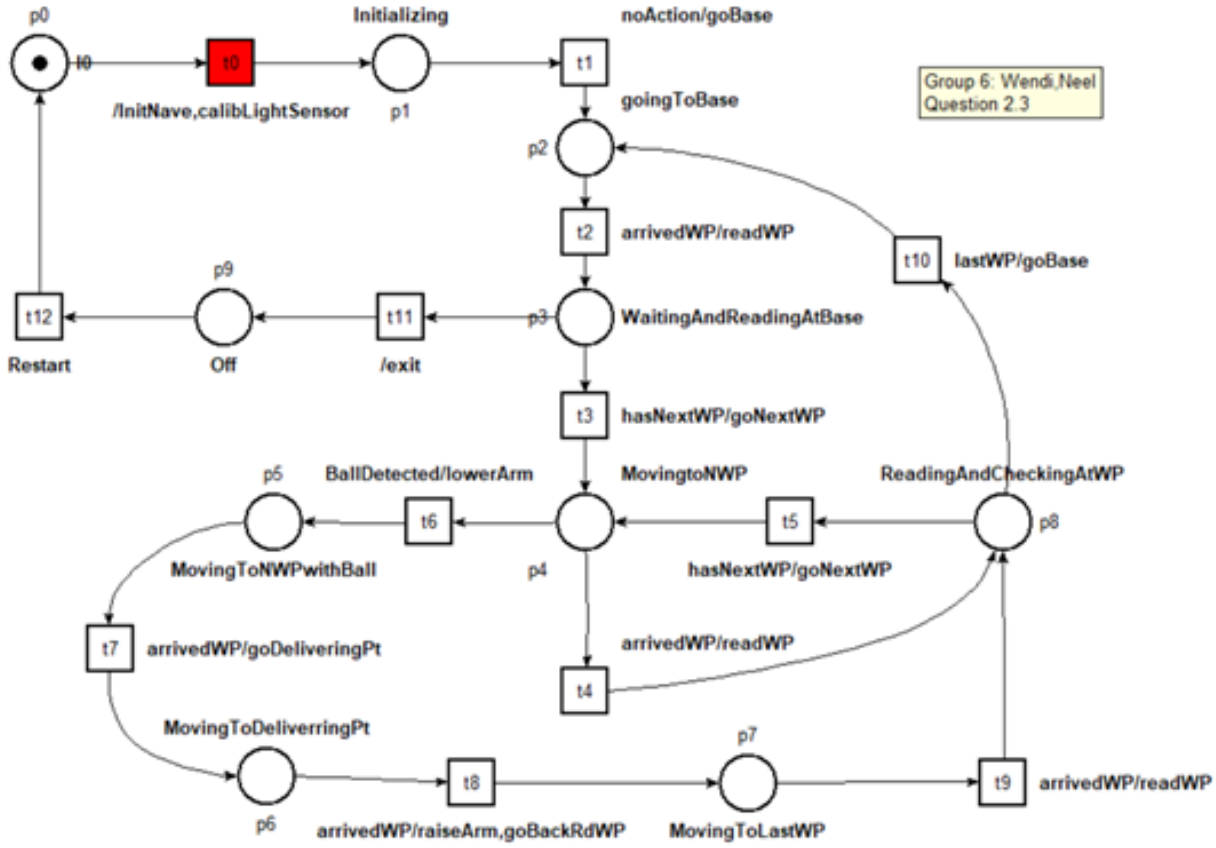


Figure 16: Initial state of the system

As we observe, the first enabled transition t0 is for initializing the robot and calibrating the light sensor. After initialization is completed, we have the next transition t1 enabled for going to the Base. Once the robot has arrived at the base, it will wait there while it reads the next waypoint.

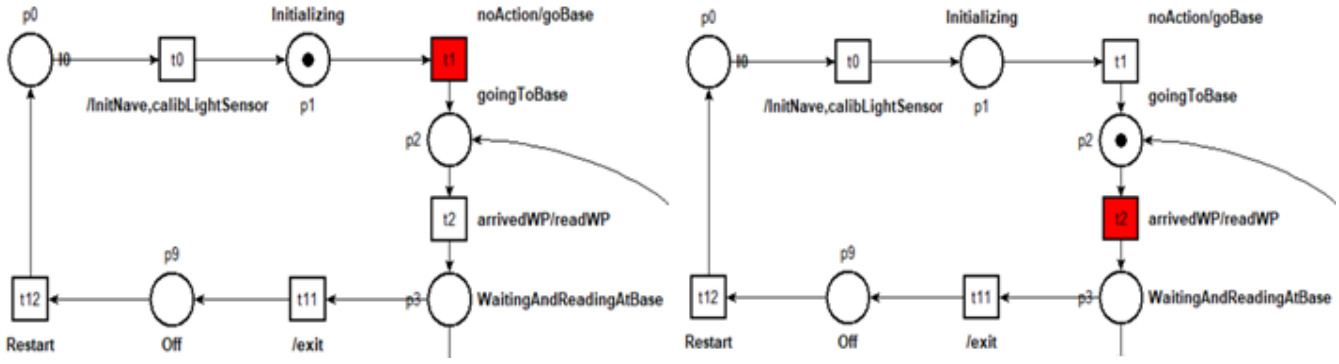


Figure 17: Initialising and going to the base

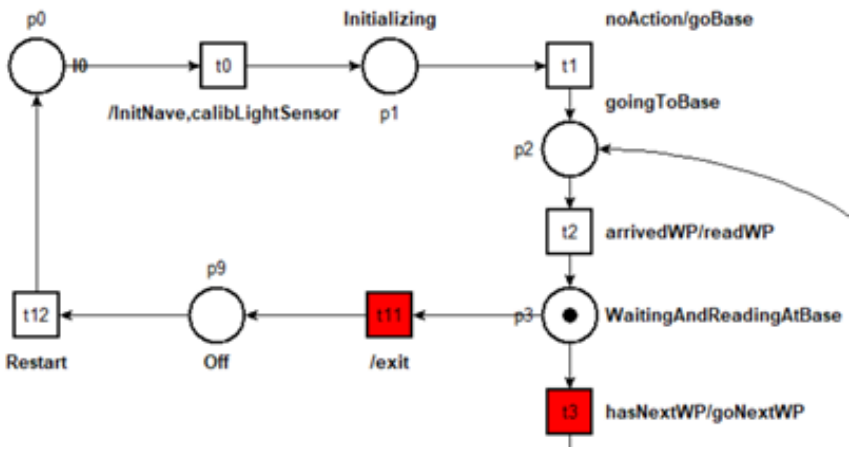


Figure 18: Waiting and reading at the base

In the scenario we are considering, there are four waypoints. So the robot reads the first waypoint, and since the hasNextWP condition will be true, we will fire the transition t3 goNextWP and the robot will start moving towards the first waypoint.

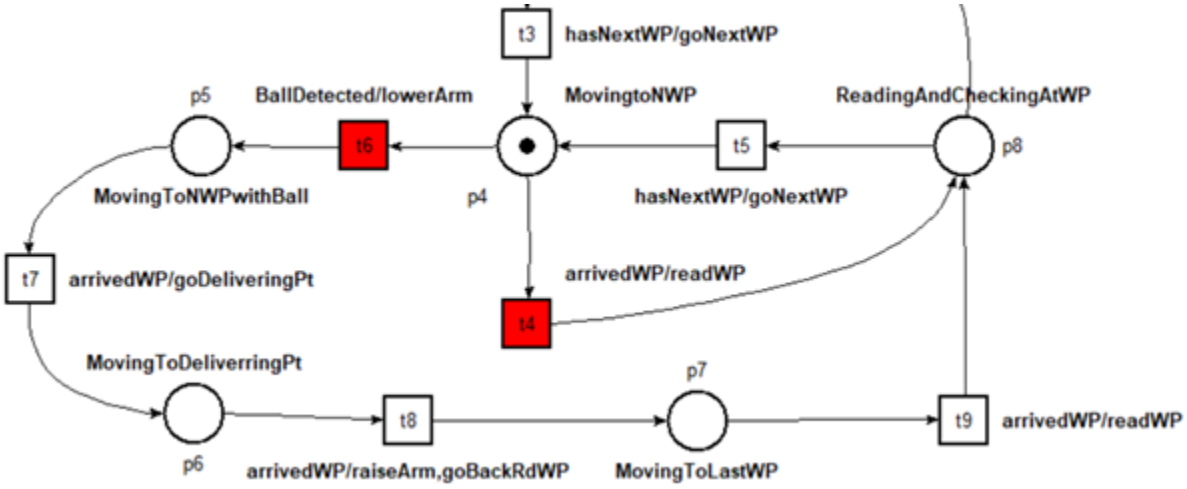


Figure 19: Moving to the next waypoint

So, now we are at the place MovingtoNWP. Here again we have a choice of two transitions. Since there is no ball present, no further action will be taken by the robot until it reaches the second waypoint as no ball is detected. So we will fire the transition t4 arrivedWP/readWP.

Now the robot will read that this is not the last waypoint, so we must start moving to the next waypoint again. So we now fire the transition t5 hasNextWP/goNextWP.

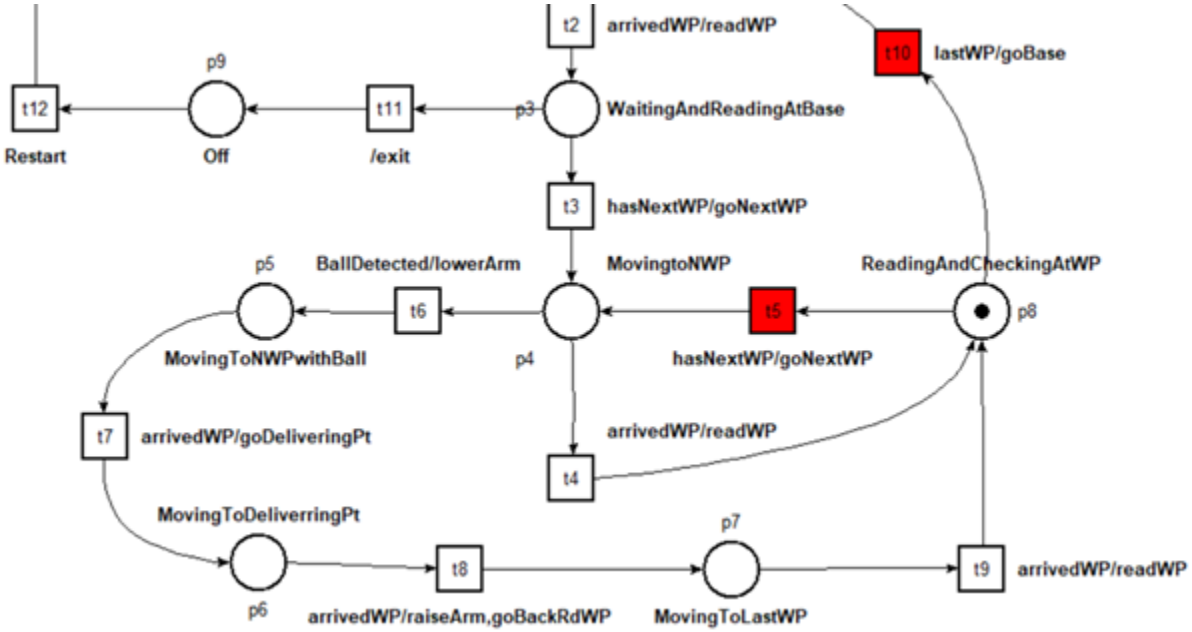


Figure 20: Reading and checking after arriving at the waypoint

Now we are moving from the first to the second waypoint. We are again in the same place as shown before in Figure 19. Once more, there is no ball present here, so no ball is detected, so we follow the same steps until we return to the place MovingtoNWP again.

This time we are moving from the second to the third waypoint. In the scenario we have taken into consideration, the ball is placed in between the second and the third waypoint. So on this occasion, a ball will be detected, hence we fire the transition t6, lower the robot arm to pick up the ball. Now we are at the place MovingToNWPwithBall place, as shown below in Figure 21.

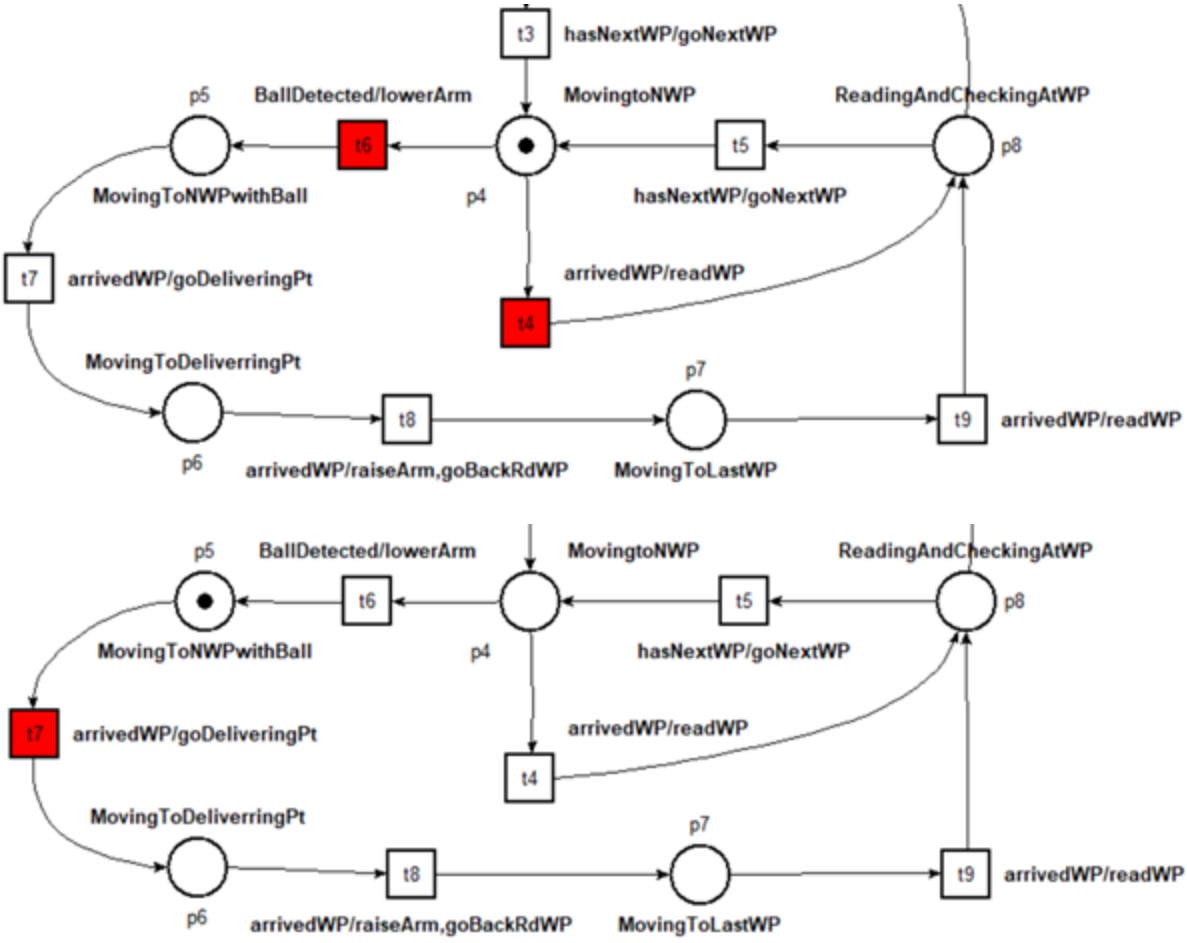


Figure 21: A ball is detected and picked up

Once the robot arrives at the third waypoint, it will start moving towards the delivery point. Thus we fire the transition t7 arrivedWP/goDeliveringPt, so we are now at the place MovingToDeliveringPoint. Once it arrives there and delivers the ball, it will raise its arm and return to the previous waypoint, which is the third waypoint. This set of transitions is shown in Figure 22.

So now that the robot is back at the third waypoint, we will read the next waypoint. So we have to fire the transition t9 arrivedWP/readWP. Once we do this, we are back at the place ReadingAndCheckingAtWP.

Figure 22: Delivering the ball and returning to the previous waypoint

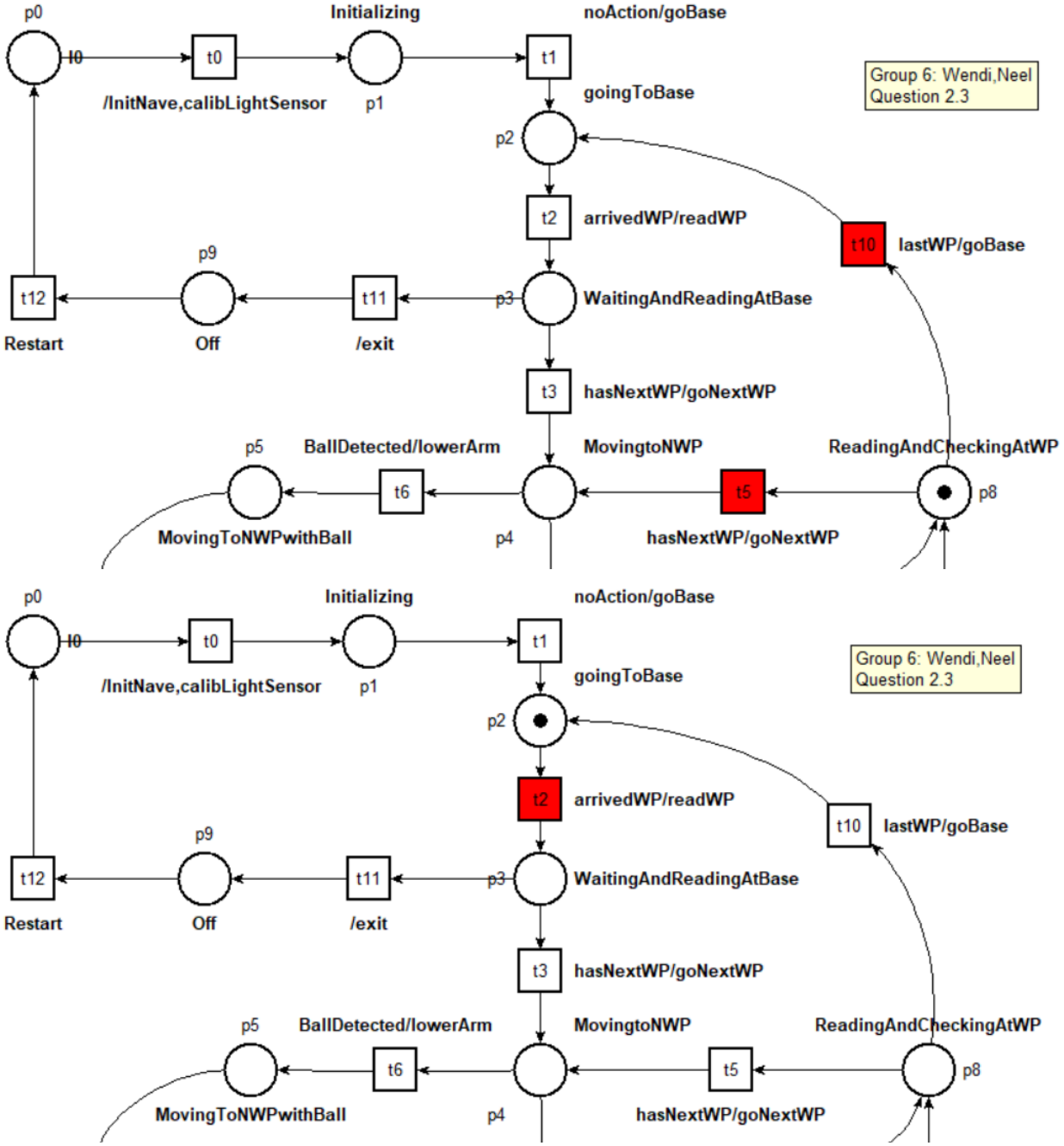


Figure 23: Moving from the last waypoint to the base

We are at the third waypoint now, so we will repeat the steps taken in Figures 20 and 19 respectively to simulate the robot reaching the fourth waypoint. Now we have reached

the last waypoint, so we will fire the transition lastWP/goBase to simulate the robot going to the base as shown in Figure 23.

Once we arrive at the base, the task is finished, since the condition “last WP was reached” (lastWP) is satisfied, the robot will fire the transition t11 lastWP/exit and stop the token player.

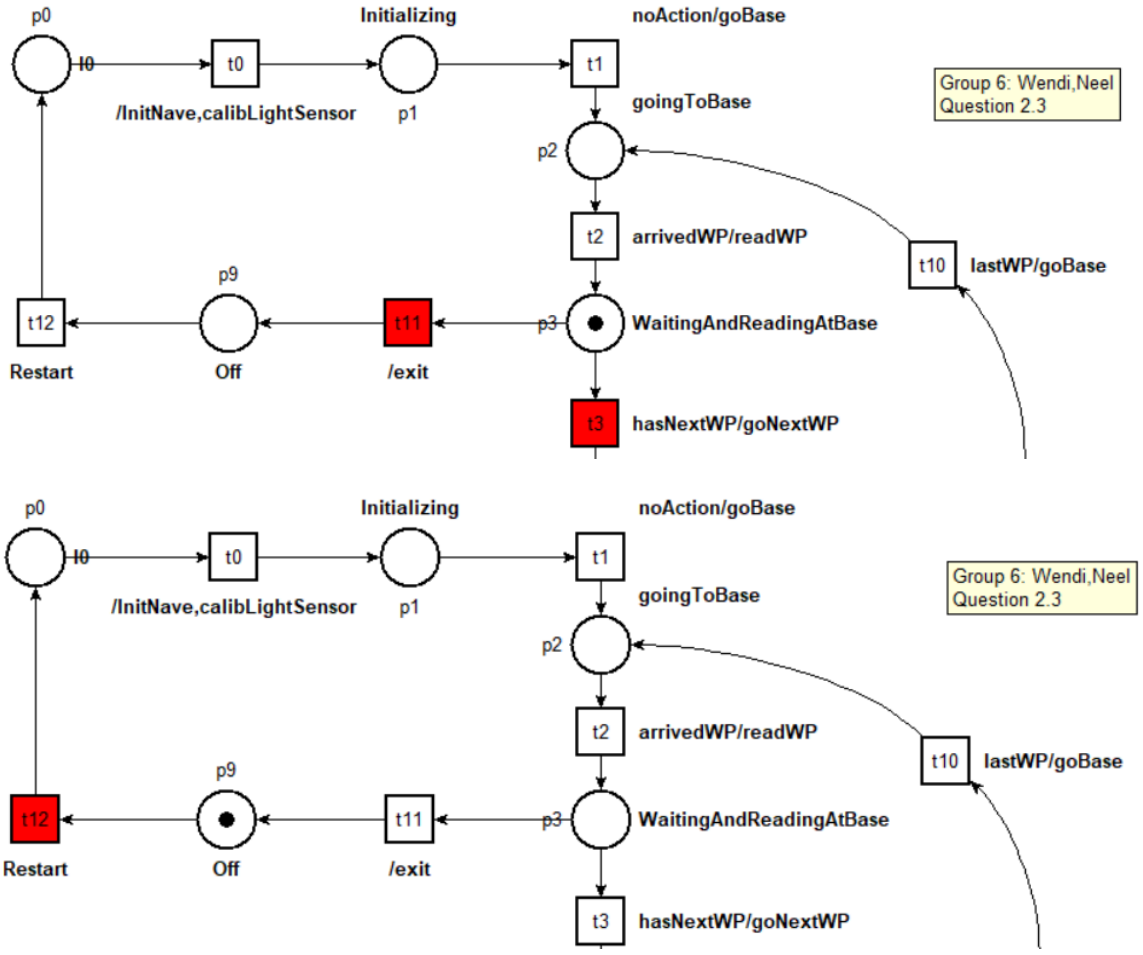


Figure 24: Exit and switch off

Thus, we have managed to validate the system and ensure that it meets the specifications.

2.5 Two cooperative robots

2.5.1 Model description

We want to use two robots in a collaborative way. Hence there are two identical robots, R1 and R2, with the same behaviour than before. We can assume that the WPList of the robots are different and that their itinerary can never cross (to avoid collision). However we assume that they have the same coordinates for the delivery point.

Since the two robots are assumed to be identical, we design a system using the model we created in 2.3, and duplicating the model twice inside one system. So now we have two identical robots operating independently.

Since the paths for the waypoints for each robot are considered to be completely separate, the robots will never come into each other's path when simply moving between their respective waypoints. So without the delivery point L, the two robots are considered as working in parallelism.

However, the situation is different when it comes to picking up the ball and delivering it. The one constraint on the system is that the delivery point for both the robots is the same point L. So, before each robot proceeds to the delivery point with the ball, it must check that the delivery point is free. This is the only point of interaction between the two systems.

This Petri net of two cooperative robots is a model of mutual exclusion, a centralized model that protects critical resources, which is the delivery point L in our case that cannot be occupied by two robots at the same time. The place p20 LFree represents this critical resource, with only one token in initial state, in order to fire either transition t7 or t15, enabling only one of the robots to go to the delivery point at one time. After transition t8 or t21, the robot comes back from the delivery point and the token comes back to p20, representing that the resource is free again.

As seen in the figure below, each robot has to check that the delivery point is free before proceeding there. So if one robot is already moving towards the delivery point, then the transition for the other robot to move towards the delivery point will be disabled, as there will not be enough tokens available.

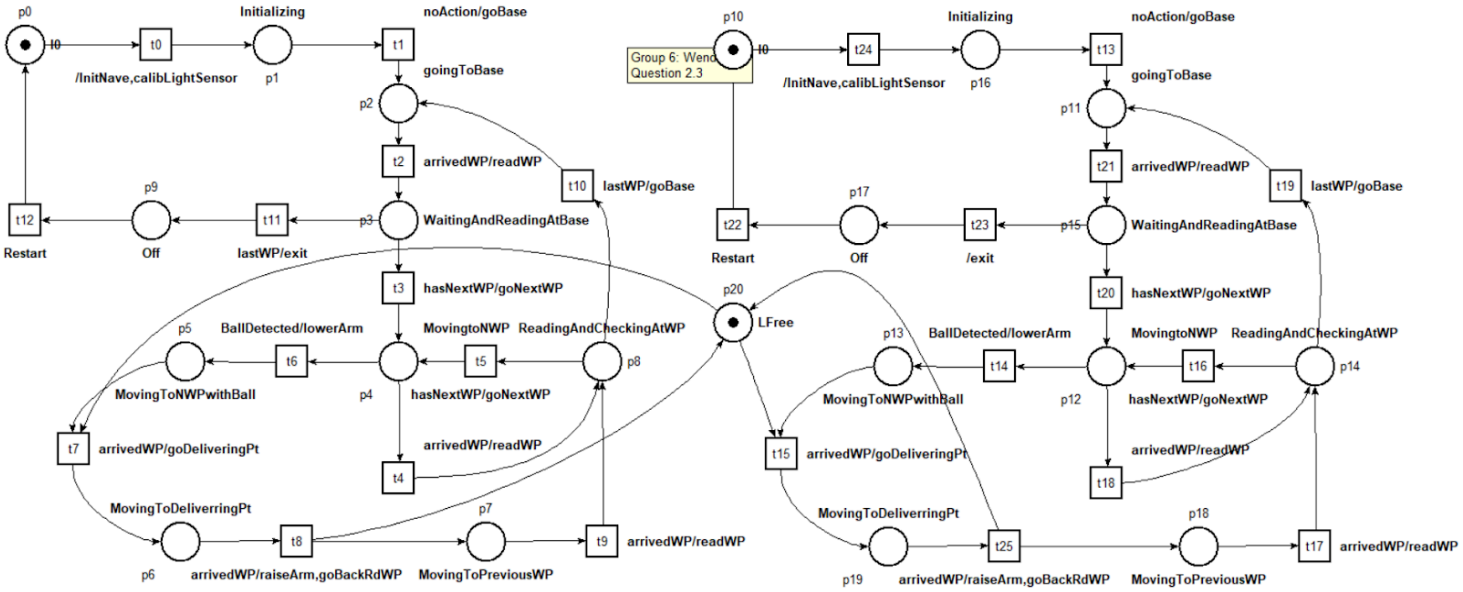


Figure 25: Model for two robots with a common delivery point

Since each robot is implemented only with its own token player, this Petri net representing two cooperative robots is not applicable in reality. It is just used to demonstrate a centralized determination mechanism.

It is also worth noticing that the choice from the place p20 LFree is not determinist, meaning that transitions t7 and t15 can be fireable at the same time in some scenario. To avoid this non-determinism, we can set a priority to robot A to make it prioriably fired when conditions are satisfied on both sides at the same time.

2.5.2 Petri net verification

Since we have taken a bottom-up design method to construct the complex system, the global Petri net does not necessarily have the properties of each module. So we need to always reanalyse the global Petri net. Figure 26-27 show that the global PN is bounded,

live and reversible, with all the places appearing at least once in one of the conservative components.

places	21	transitions	26	net	bounded	Y	live	Y	reversible	Y

abstraction	count	props	psets	dead	live
states	99	21	?	0	99
transitions	256	26	?	0	26

Figure 26: General conclusion of state space analysis for two cooperative robots

```
P-SEMI-FLOWS GENERATING SET ----------------

invariant

p10 p11 p12 p13 p14 p15 p16 p17 p18 p19 (1)
p17 p20 p6 (1)
p0 p1 p2 p3 p4 p5 p6 p7 p8 p9 (1)

0.000s

T-SEMI-FLOWS GENERATING SET ----------------

consistent

t12 t13 t14 t23 t25
t0 t1 t11 t2 t24
t14 t15 t16 t19 t20 t21 t22
t14 t17 t20 t22
t15 t16 t18 t19 t21
t17 t18
t10 t2 t3 t6 t7 t8 t9
t10 t2 t3 t4
t5 t6 t7 t8 t9
t4 t5
```

Figure 27: Structural analysis for two cooperative robots

2.5.3 Scenario validation

To validate our model, once again we will use the stepped simulator to emulate a possible scenario for the robots. Since the model for each robot is exactly the same as seen before, apart from the common delivery point, so the steps up until both the robots try to deliver a ball to the delivery point at the same time will remain unchanged as described previously in the validation of our original model.

In Figure 28 below, we see that both the robots are moving towards their respective next waypoint. Let us assume that for our scenario, both robots detect a ball during this movement.

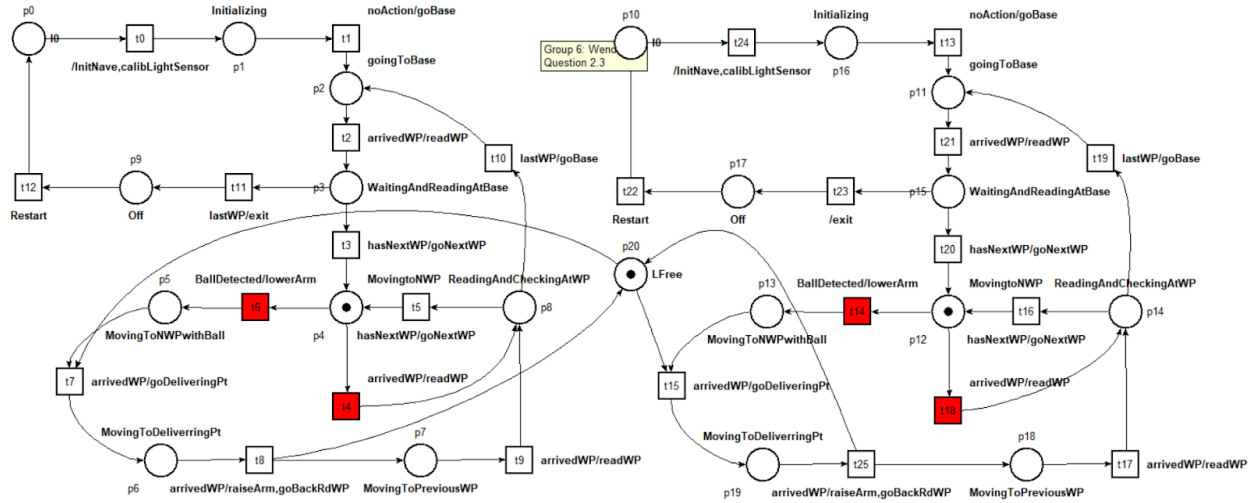
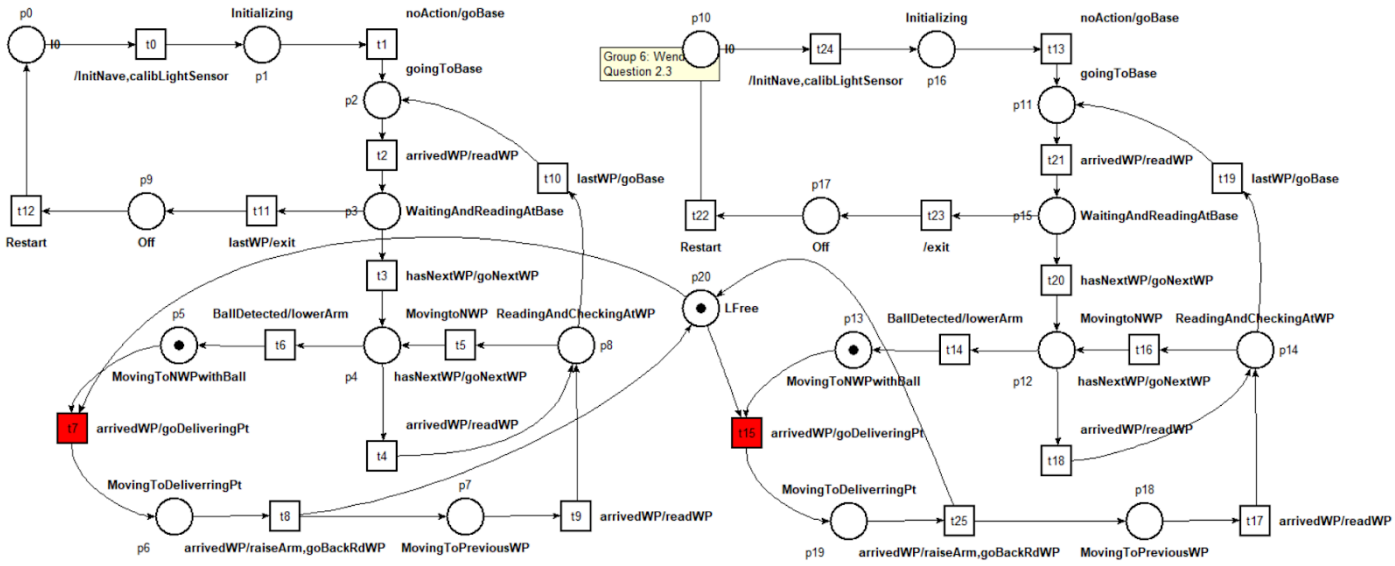


Figure 28: Both robots are moving towards the next waypoint

Since both robots detect a ball, we fire the transitions BallDetected/lowerArm for both the robots. There is no conflict as of yet and both the transitions are fireable as we can see above.

Upon firing both the transitions, we reach the potential point of conflict. As we can see below in Figure 29, the next transitions for each robot cannot be fired simultaneously. This is good because we do not want both the robots to collide when going to the same delivery point. As we can see, there is a single token in LFree, so if one robot fires the transition, the other will not have any available transitions and will have to wait.

Figure 29: Both robots have picked up the ball



For our scenario, we will assume that the first robot reaches its next waypoint first. So we will fire the transition arrivedWP/goDeliveringPt for the first robot. The transition is fireable because the token is present in LFree, which implies that the delivery point is free.

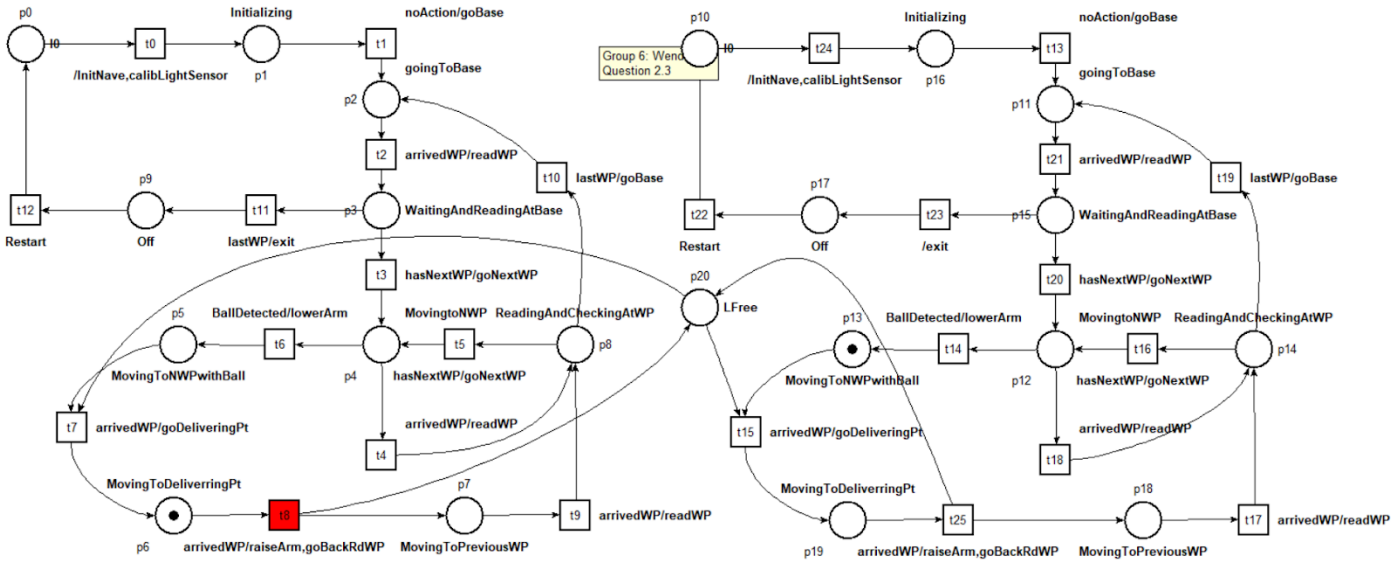
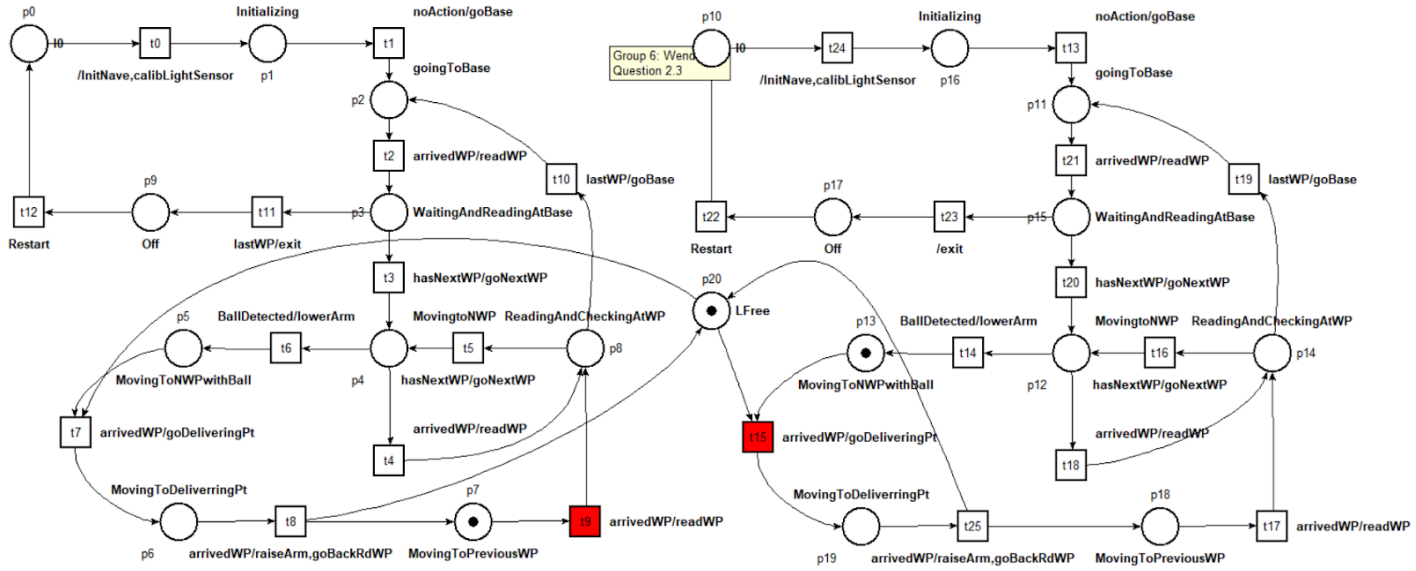


Figure 30: First robot heading to delivery point

So after firing this transition, we can see that since the first robot is moving towards the delivery point, the second robot cannot do the same, the corresponding transition for the second robot is no longer fireable. Now once the robot has completed the delivery, we

Figure 31: First robot heading back to the previous waypoint after delivery



will fire the only fireable transition, arrivedWP/raiseArm,goBackRdWP. So the first robot will head back to the previous waypoint, as seen in Figure 31.

Now that the first robot is heading back, the delivery point has become for the second robot. This would be safe in the scenario we have assumed because the two robots do not have intersecting paths. So let us assume that the second robot has reached the waypoint and was waiting there. So, the next transition, arrivedWP/goDeliveringPt, for the second robot is now available. We now fire that transition.

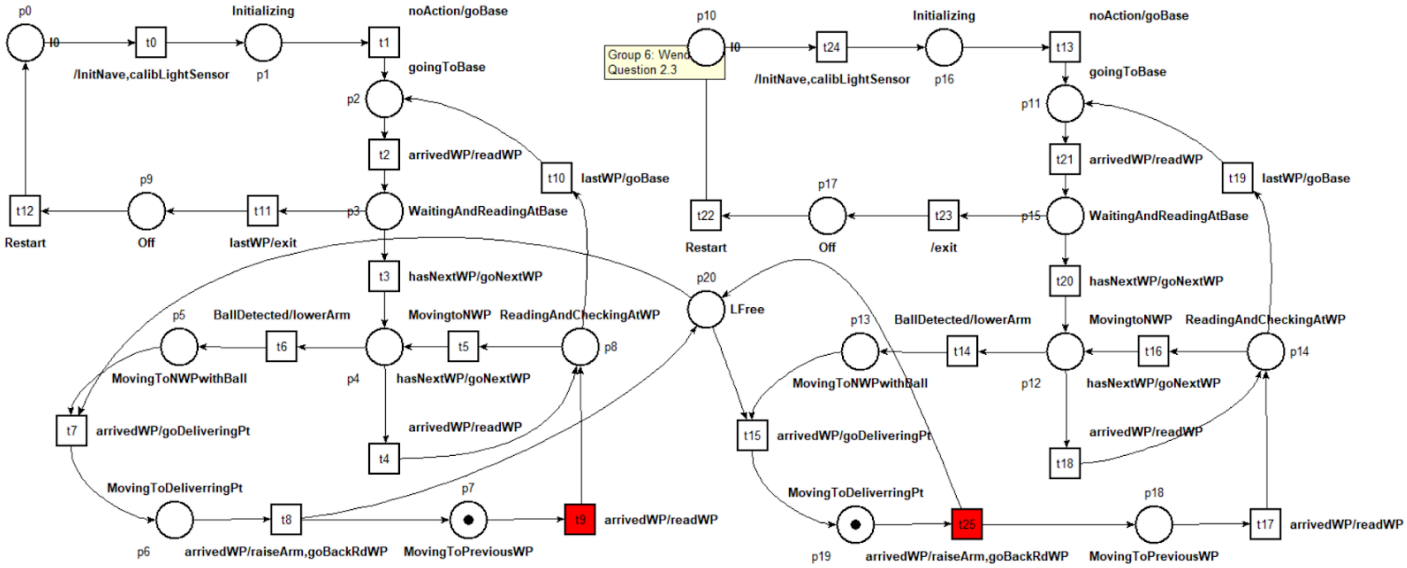
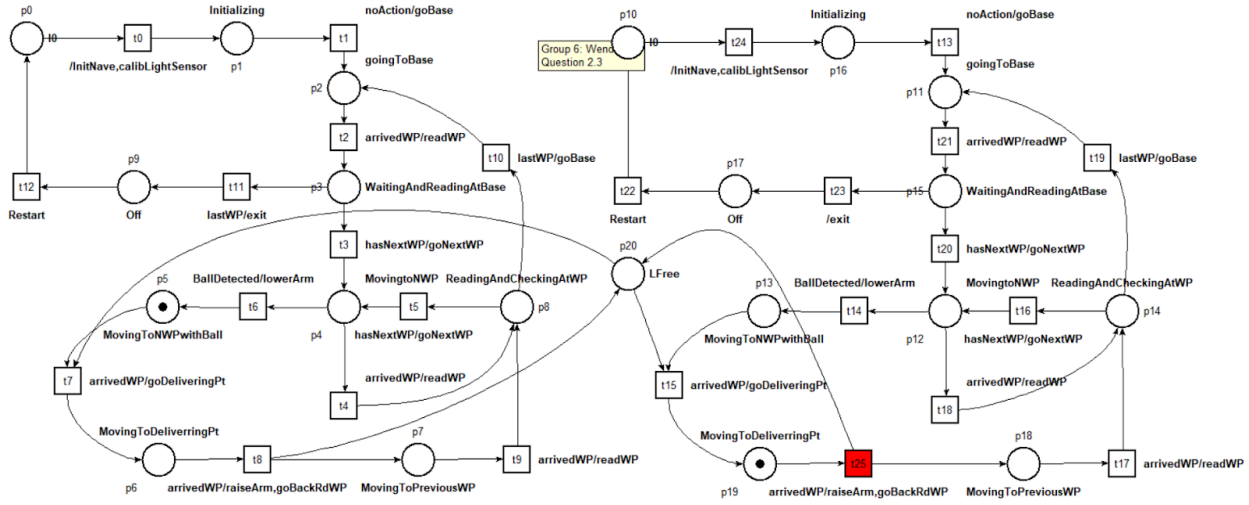


Figure 32: Robot 2 heading to delivery point

Figure 33: First robot waiting after picking up ball



Now that the second robot is moving to the delivery point, theoretically if the first robot reaches its next waypoint, picks up a ball, amd reaches the following waypoint, it would reaches the next waypoint, picks up a ball, amd reaches the following waypoint, it would have to wait there until the second robot delivers its ball and starts returning, only then would the arrivedWP/goDeliveringPt transition for the first robot become fireable again. This scenario would lead to the state shown above in Figure 33.

Let us assume that the second robot has finally delivered the ball, and is now returning to its previous waypoint. So now, the first robot can proceed to the delivery point to deliver its second ball.

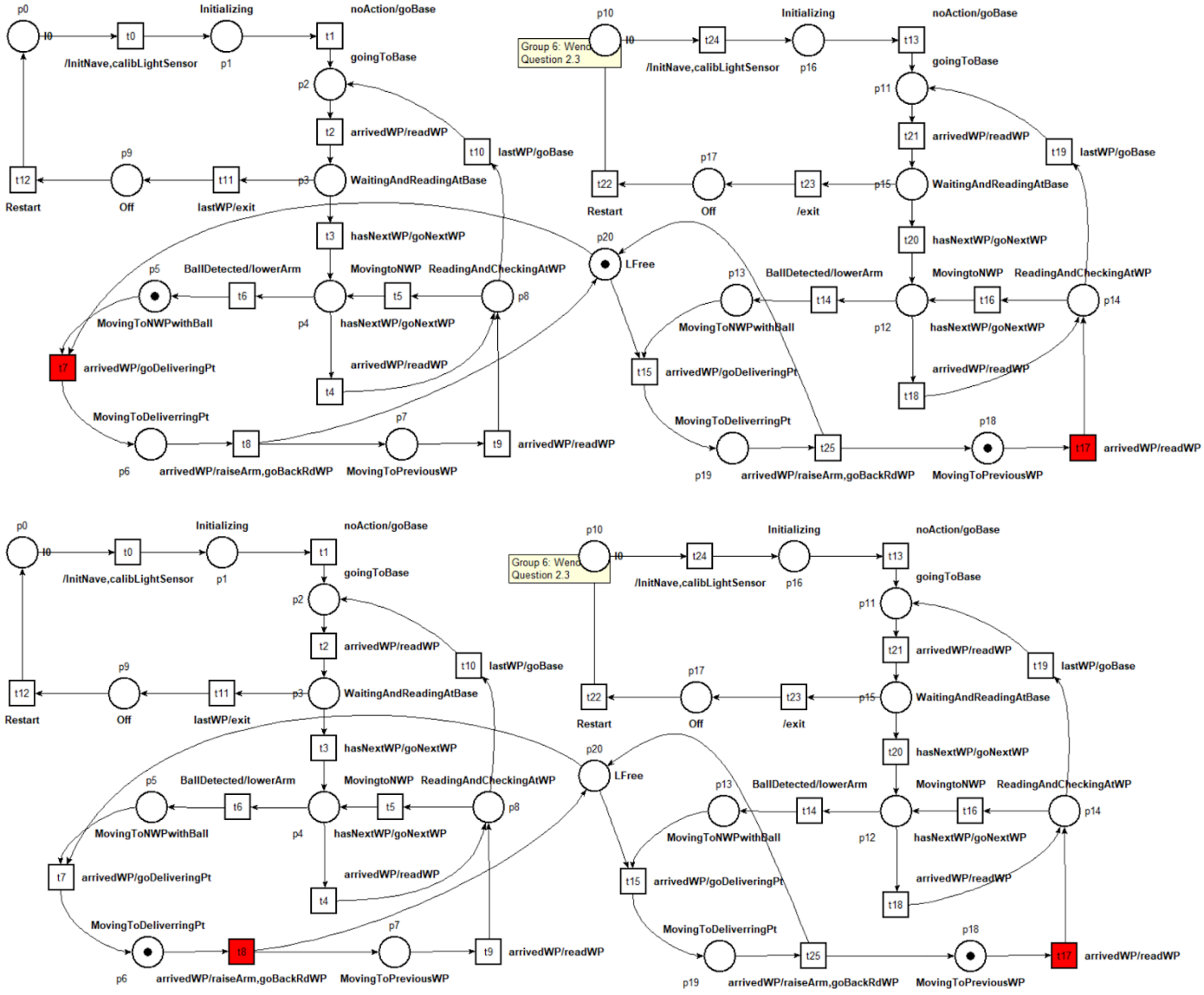


Figure 34: First robot delivering the second ball in its path

Let us assume that now, there are no more balls to be delivered by either of the robots. So now, each robot can proceed along its designated path independently and there will be no more conflict between them. So the steps after this point will simply proceed in the same way for each robot as we simulated earlier for our original model (Figures 19-24).

Thus we have validated our petri net model for two cooperative robots, ensuring that they work as per the specifications.