Tokens Part 2

We have now addressed when to fire a transition and what to do with input event receipt status and tokens when a transition fires. However, the current representation works only for a single robot.

What if we want many robots station keeping? We would need to keep track of what each robot is doing individually as they will be in different locations.

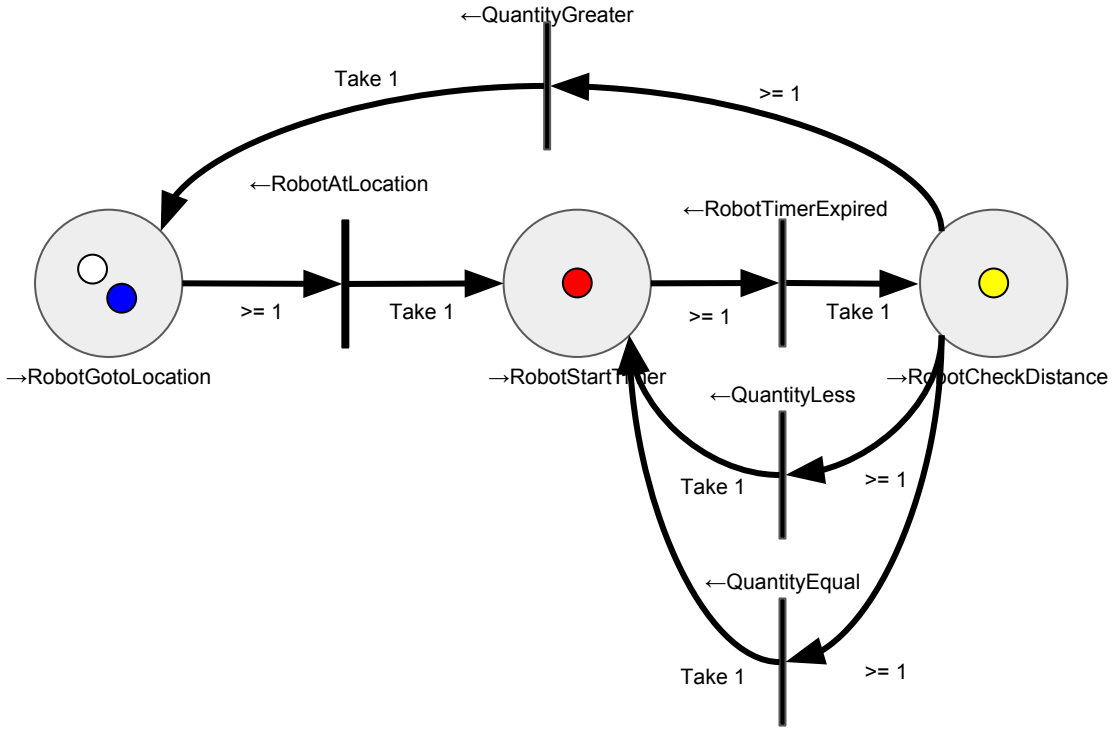
We have now addressed when to fire a transition and what to do with input event receipt status and tokens when a transition fires. However, the current representation works only for a single robot.

What if we want many robots station keeping? We would need to keep track of what each robot is doing individually as they will be in different locations.

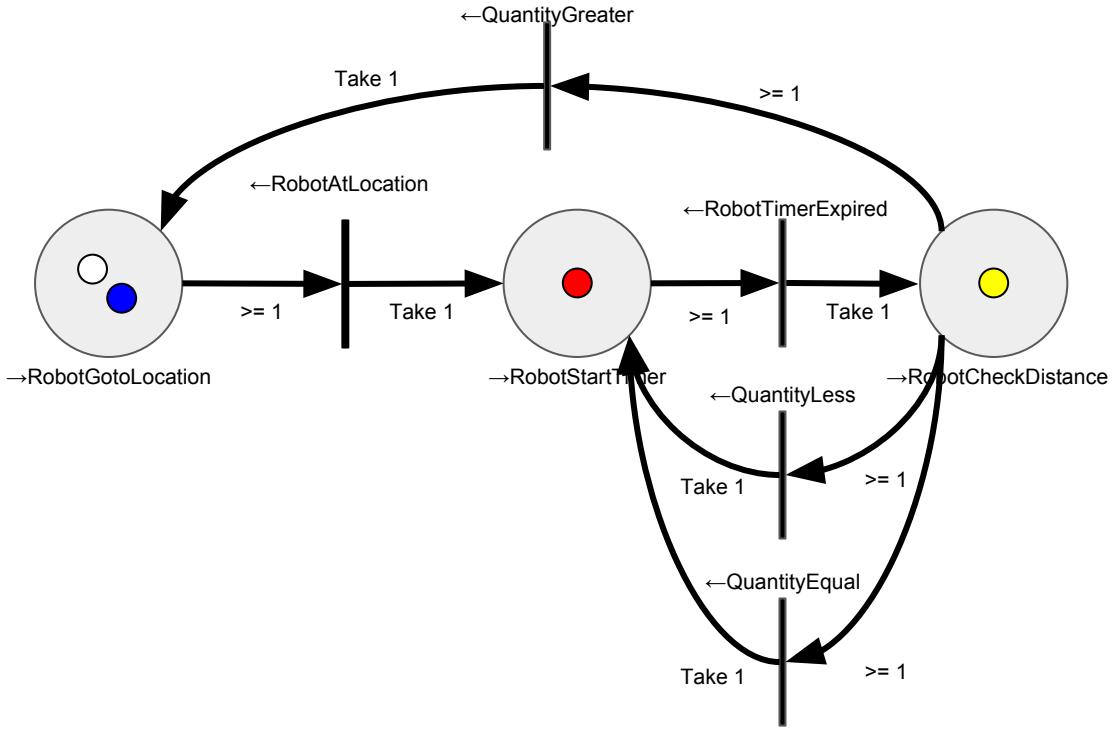
To address this we use the concept of “colored tokens,” allowing tokens to optionally store information, such as the ID of a robot. To make it easy to visualize, we will assign each robot ID a color and color its token accordingly.

If a token has no robot ID attached to it, we call it a “generic token” and color it black. Generic tokens are used in parts of plans which don’t involve robots.

Here we can see the station keeping plan running with tokens for White, Blue, Red, and Yellow robots. No generic tokens are used.



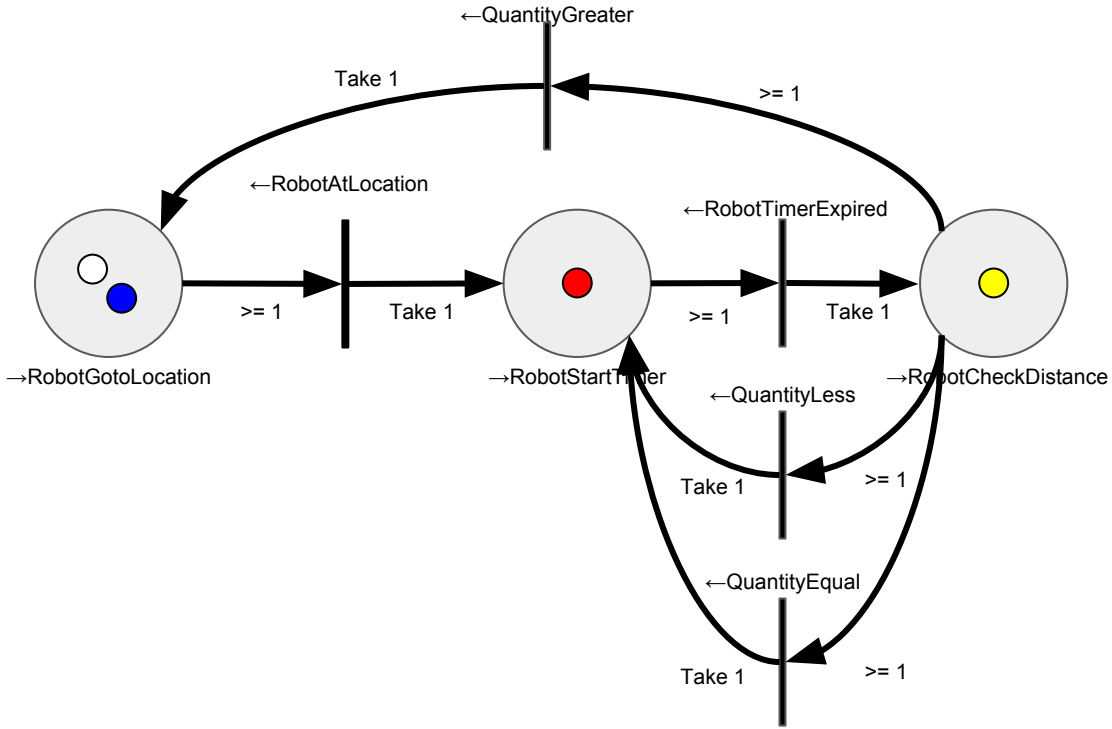
QUIZ 6-1: What is white robot doing?



Quiz 6-1 Solution

White robot is moving to the station keep location

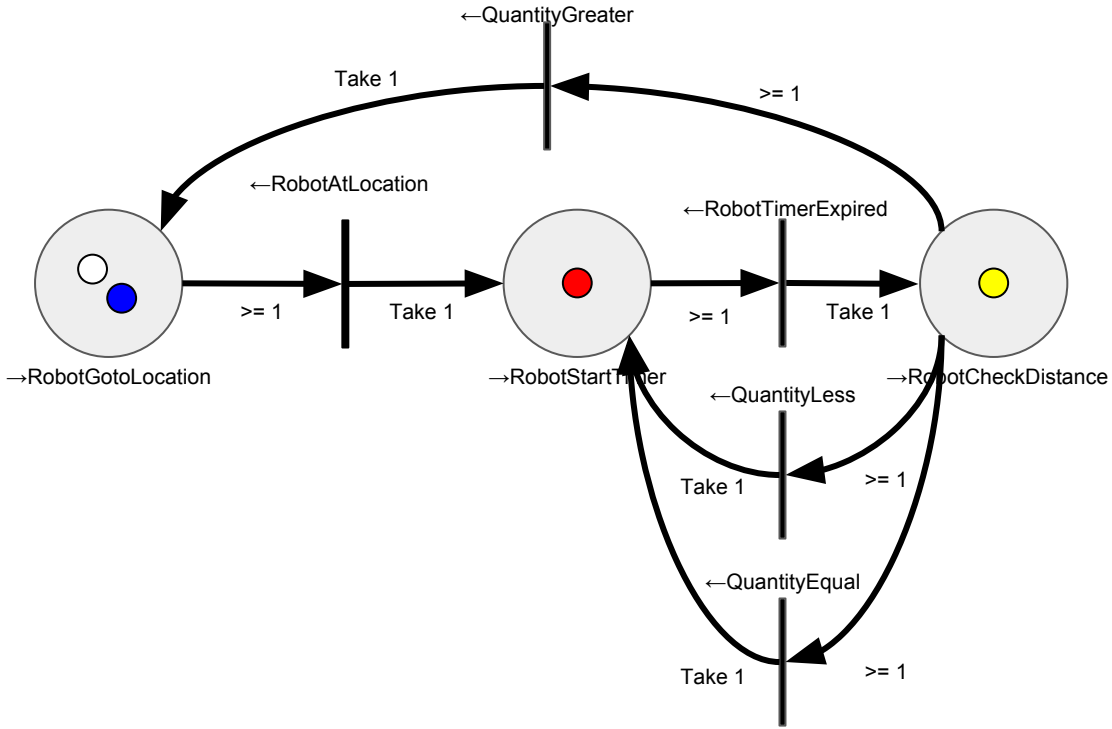
QUIZ 6-2: What is blue robot doing?



Quiz 6-2 Solution

Blue robot is moving to the station keep location

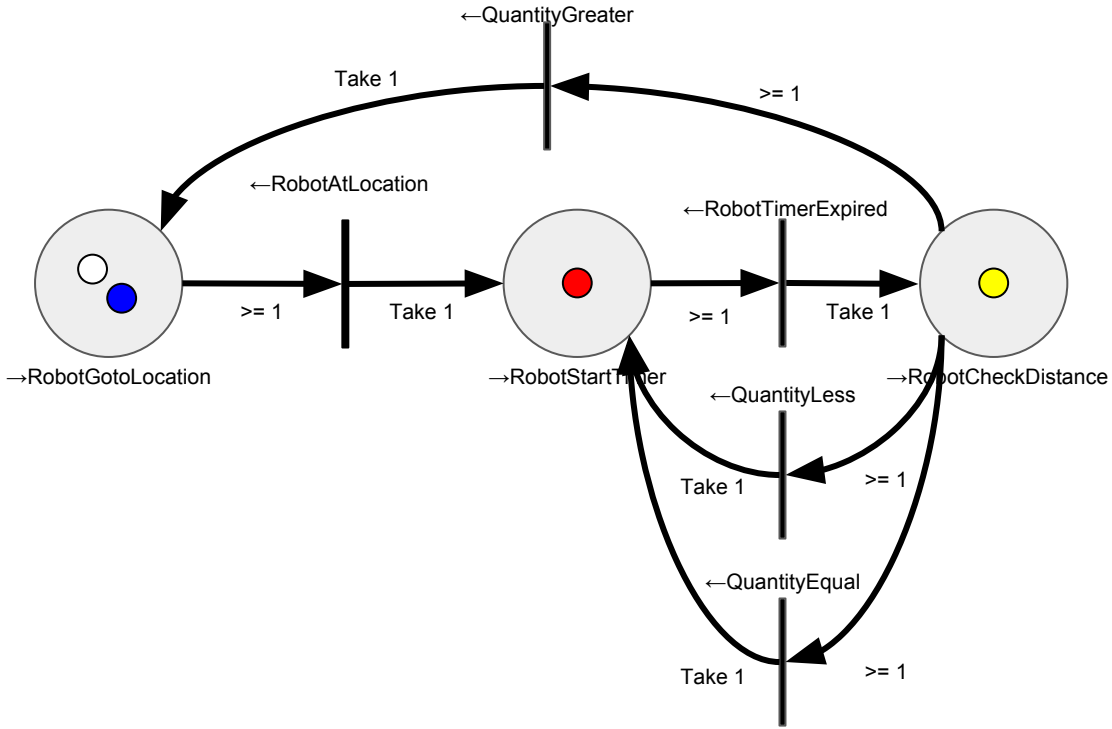
QUIZ 6-3: What is red robot doing?



Quiz 6-3 Solution

Red robot is waiting

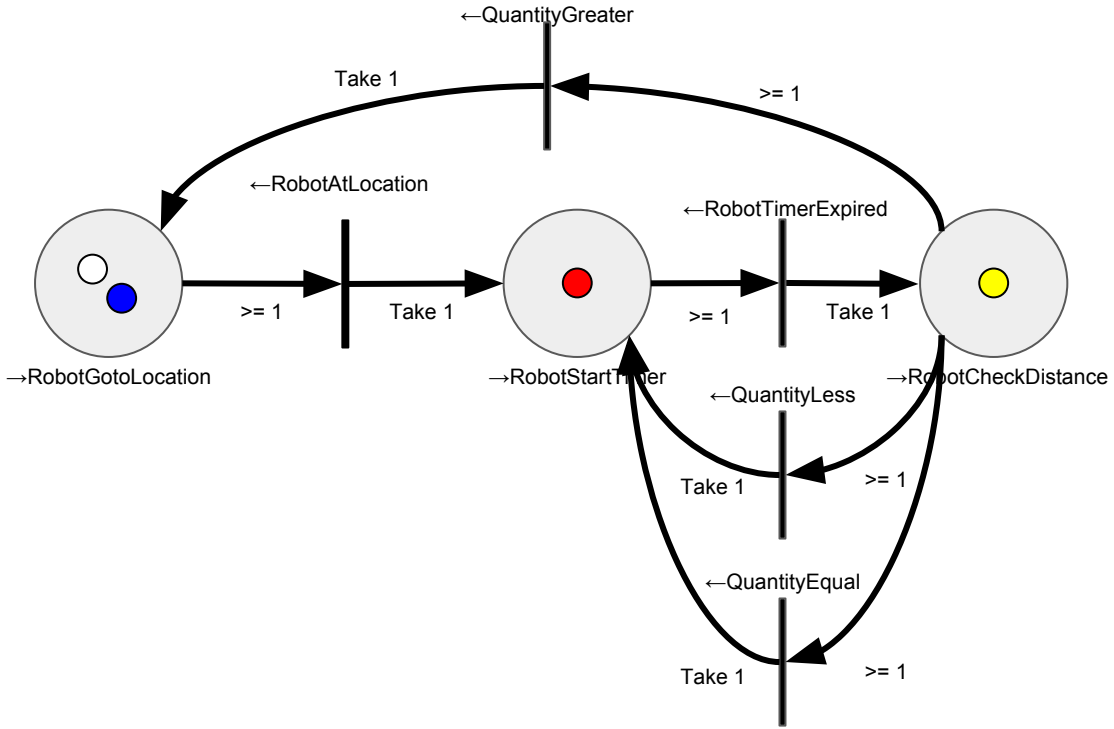
QUIZ 6-4: What is yellow robot doing?



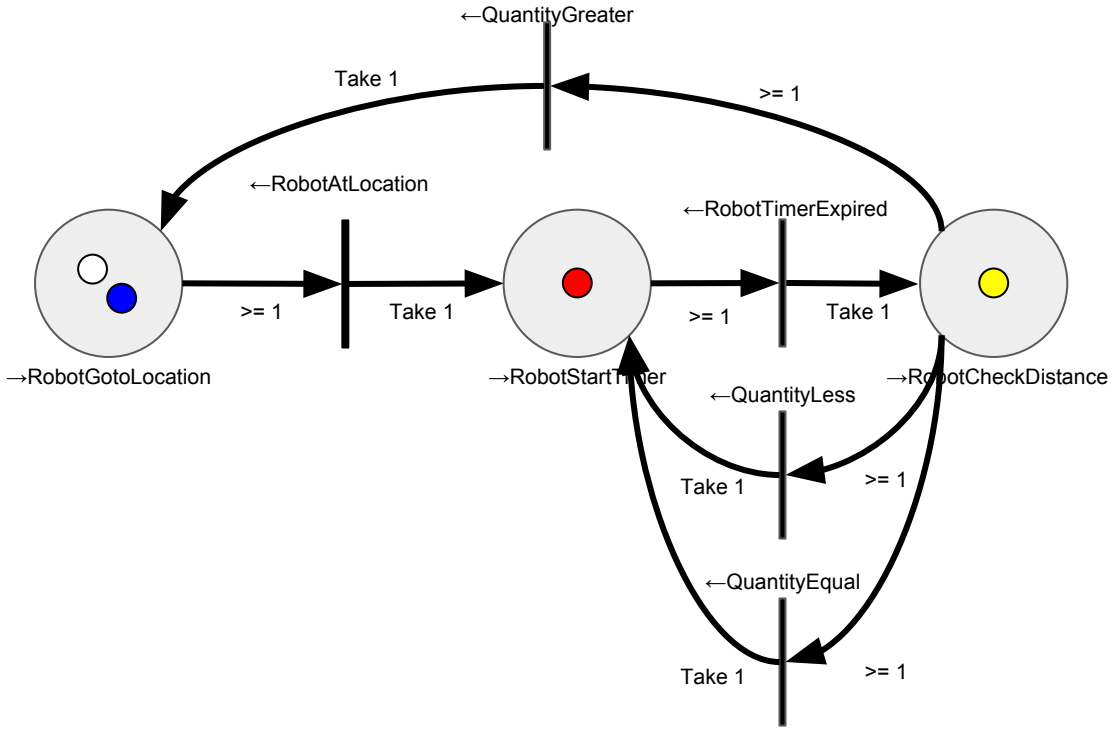
Quiz 6-4 Solution

Yellow robot is checking its distance from the station keep location

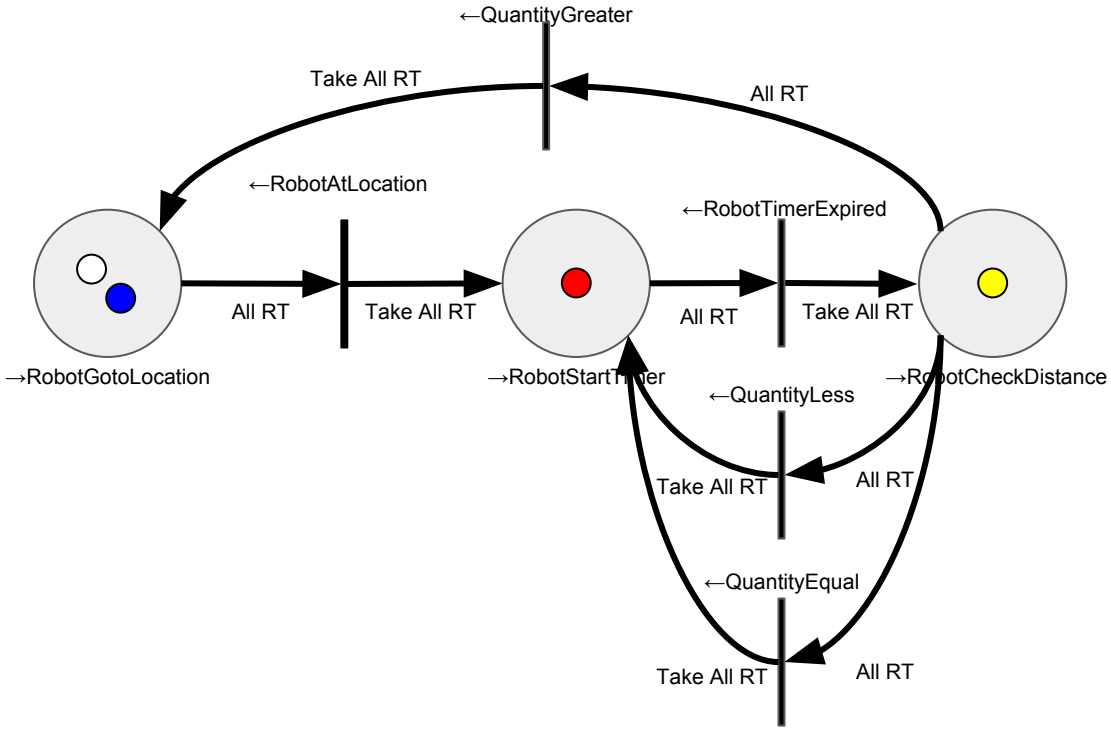
However, this representation causes a problem with the edge requirements as there is no way of distinguishing between tokens. Consider this: when Blue robot finishes moving to its location and generates a ←RobotAtLocation input event, how will we know to just move the Blue token?



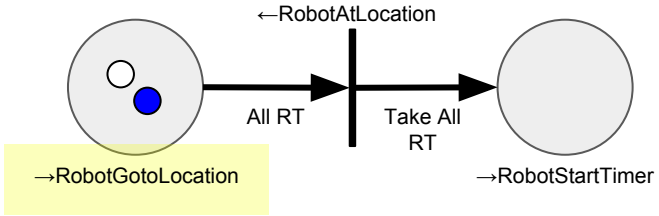
We solve this with the concept of “Relevant Tokens” (RT). An input event can list any number of tokens which are relevant to their generation. When Blue robot finishes moving to its location, a ←RobotAtLocation input event will be generated with “Blue robot token” as a relevant token. We also modify the edge requirements to only manipulate the set of relevant tokens (RT).



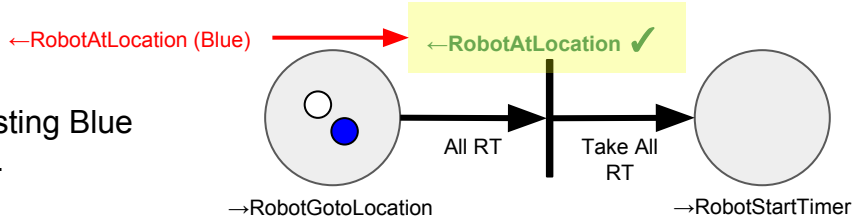
Let’s work through an example looking at a small portion of the plan.



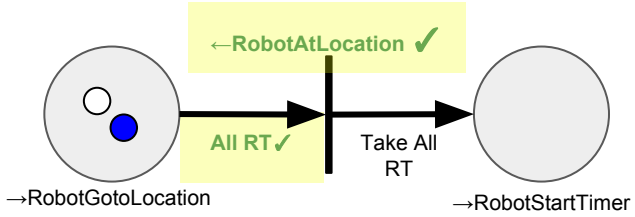
1. Blue and White robots begin moving to their locations.



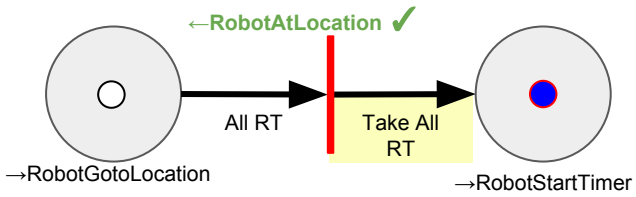
2. Blue robot finishes moving and generates a ←RobotAtLocation, listing Blue robot token as a RT. We mark ←RobotAtLocation as being received.



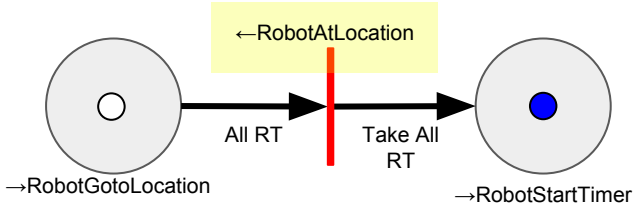
3. All input events on the transition have been received, so we check if all its in edge requirements are satisfied. The only in edge requirement is “All Relevant Tokens,” so we look at the list of relevant tokens for each input event. The only input event is ←RobotAtLocation, and its relevant token list is simply Blue robot token. Blue robot token is in “←RobotGotoLocation,” so the in edge requirement is satisfied.



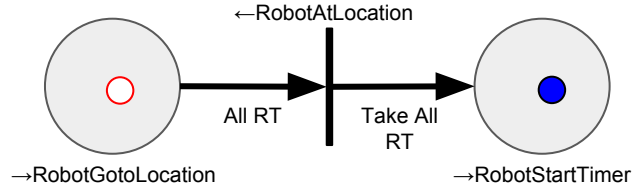
4. All input events have been received and all in edge requirements have been satisfied, so the transition fires. The out edge requirement is “Take All Relevant Tokens,” and the list of relevant tokens is simply Blue robot token. Blue robot token is removed from all places connected to the transition by an in edge and added to the place connected to the out edge. So the blue token is removed from “→RobotGotoLocation” and added to “→RobotStartTimer.”



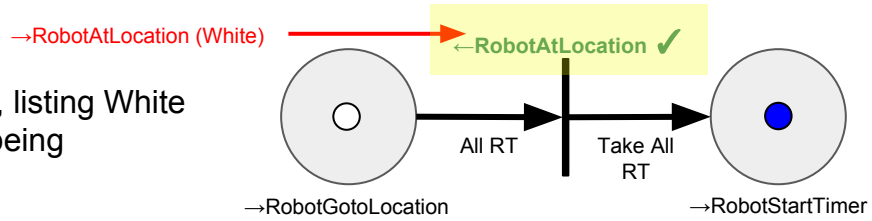
5. We have manipulated the tokens, so now the input events are marked as “not received” and we are done firing the transition.



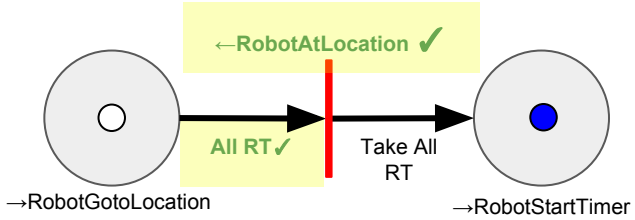
6. White robot continues moving to its location.



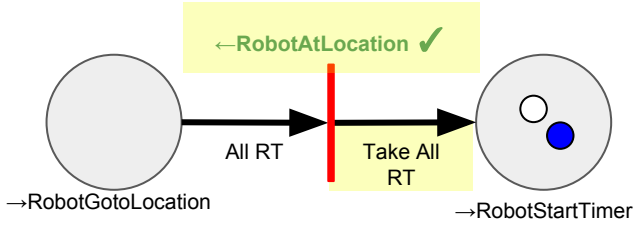
7. White robot finishes moving and generates a ←RobotAtLocation, listing White robot token as a Relevant Token. We mark ←RobotAtLocation as being received.



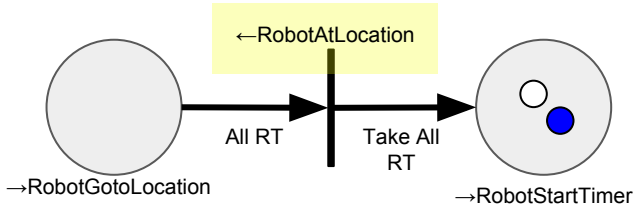
8. All input events on the transition have been received, so we check if all its in edge requirements are satisfied. The only in edge requirement is “All Relevant Tokens,” so we look at the list of relevant tokens for each input event. The only input event is ←RobotAtLocation, and its relevant token list is simply White robot token. White robot token is in “→RobotGotoLocation,” so the in edge requirement is satisfied.

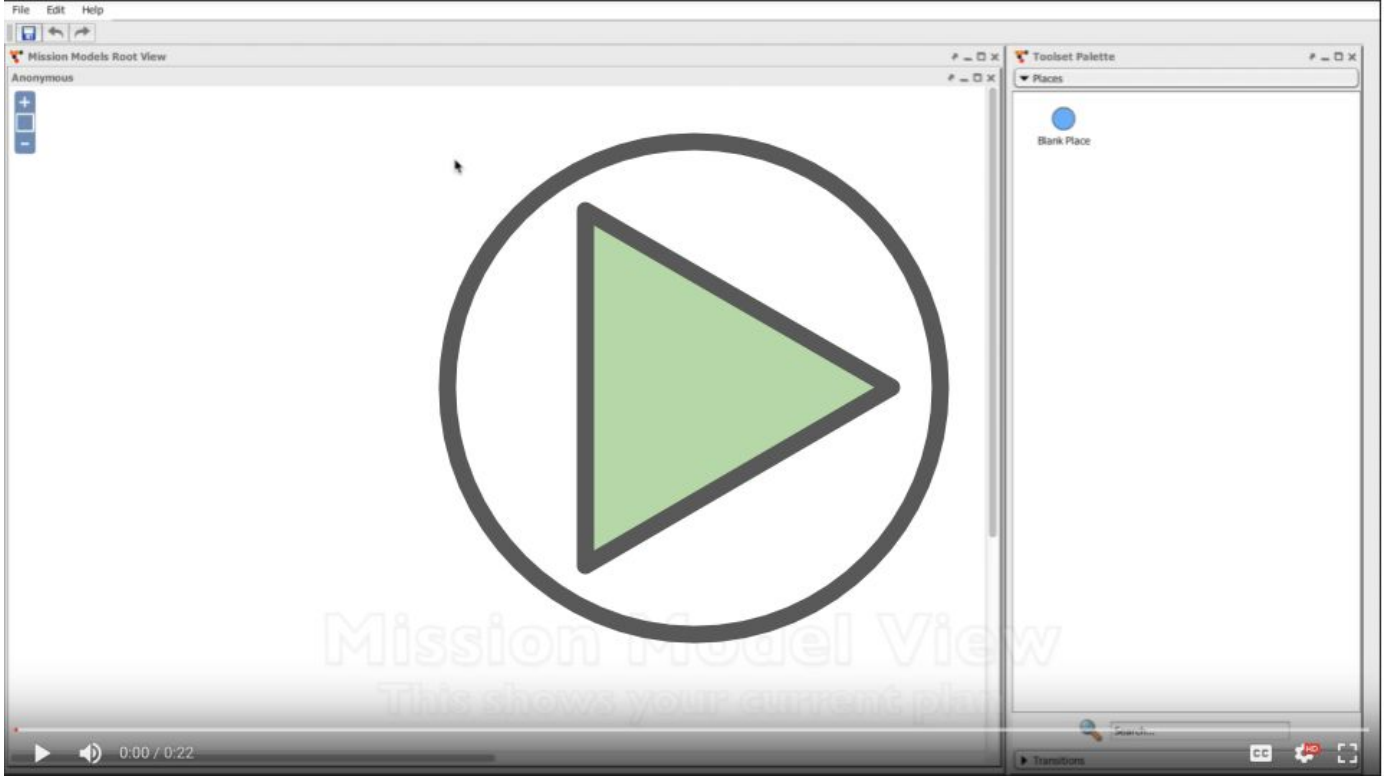


9. All input events have been received and all in edge requirements have been satisfied, so the transition fires. The out edge requirement is “Take All Relevant Tokens,” and the list of relevant tokens is simply White robot token. White robot token is removed from all places connected to the transition by an in edge and added to the place connected to the out edge. So the white token is removed from “→RobotGotoLocation” and added to “→RobotStartTimer.”



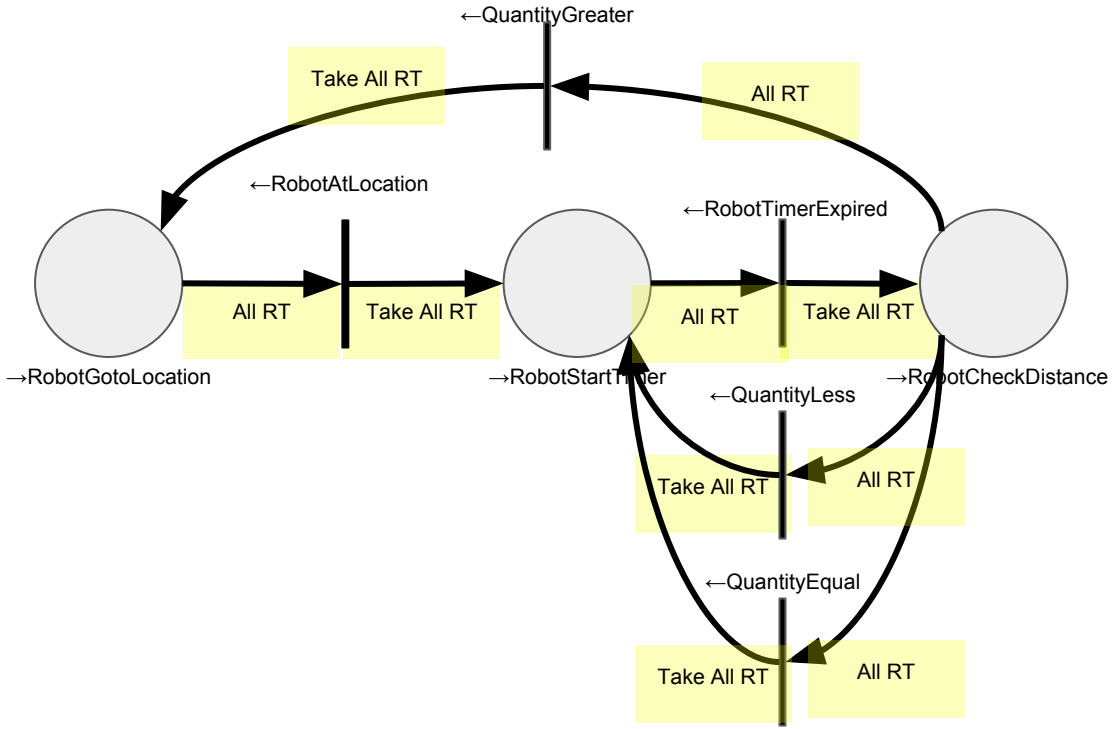
10. We have manipulated the tokens, so now the input events are marked as “not received” and we are done firing the transition.





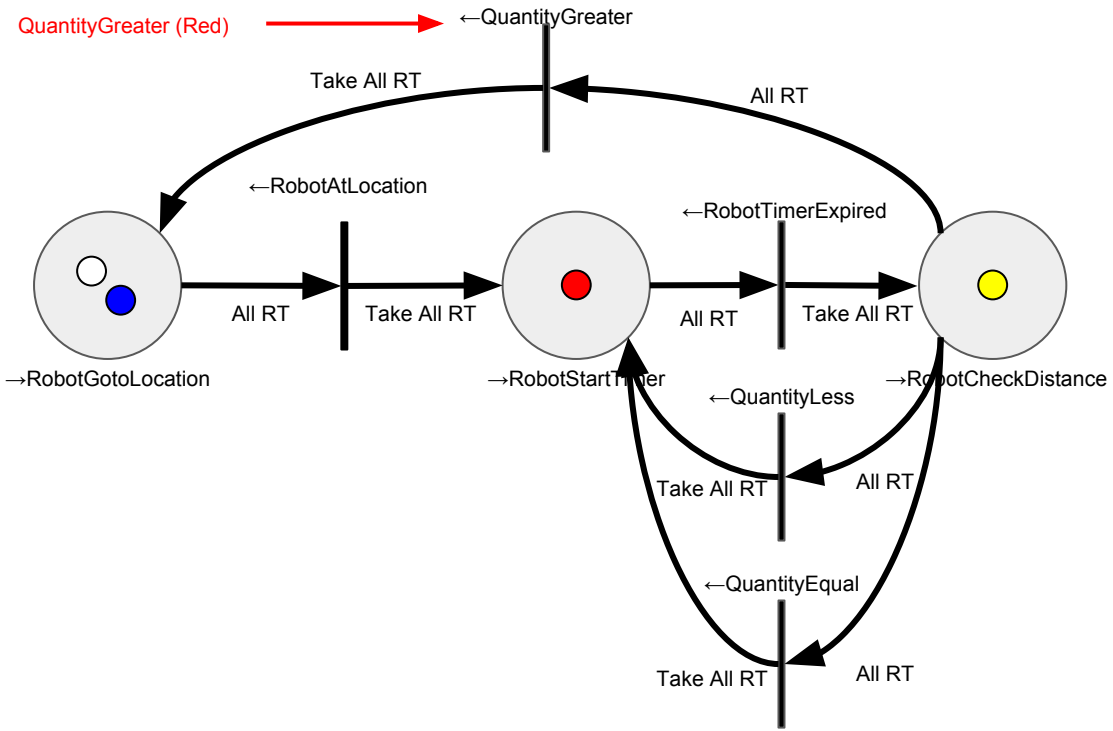
Watch “Using Relevant Tokens”: This video will show you how to use new RT in and out edge requirements.

Job 6-1: Update the edge requirements to use RT



QUIZ 6-5: What would happen if the following input event was received?

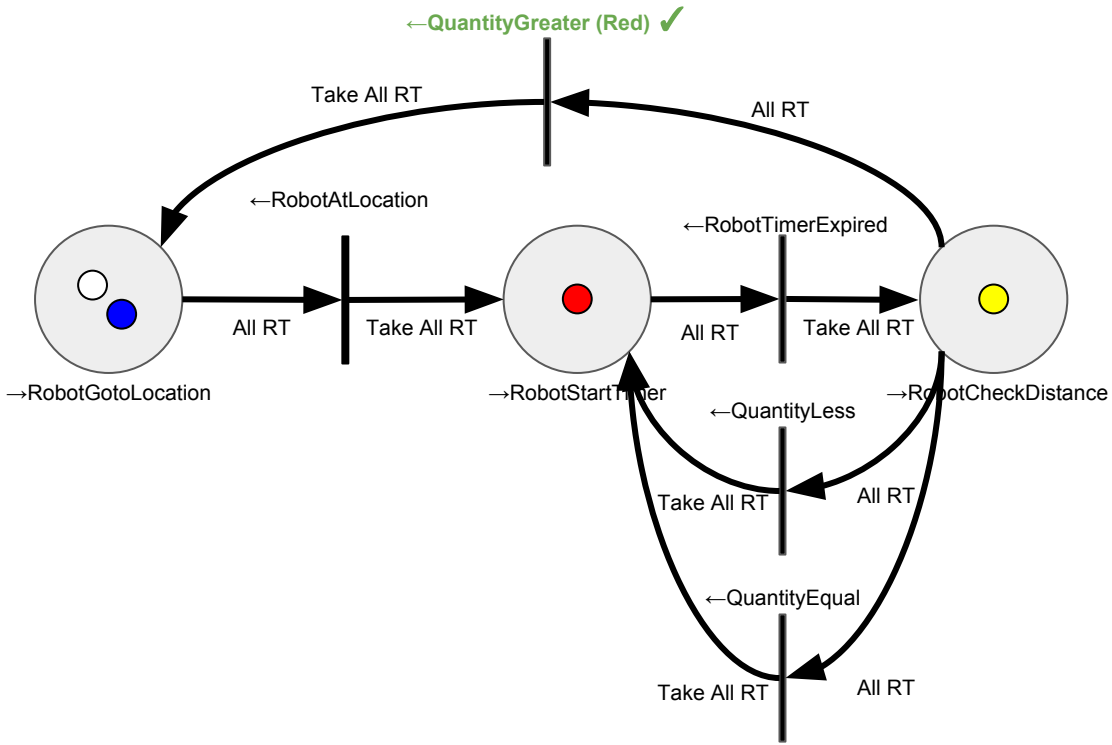
Quantity Greater, Relevant Tokens = Red robot token



Quiz 6-5 Solution Part I

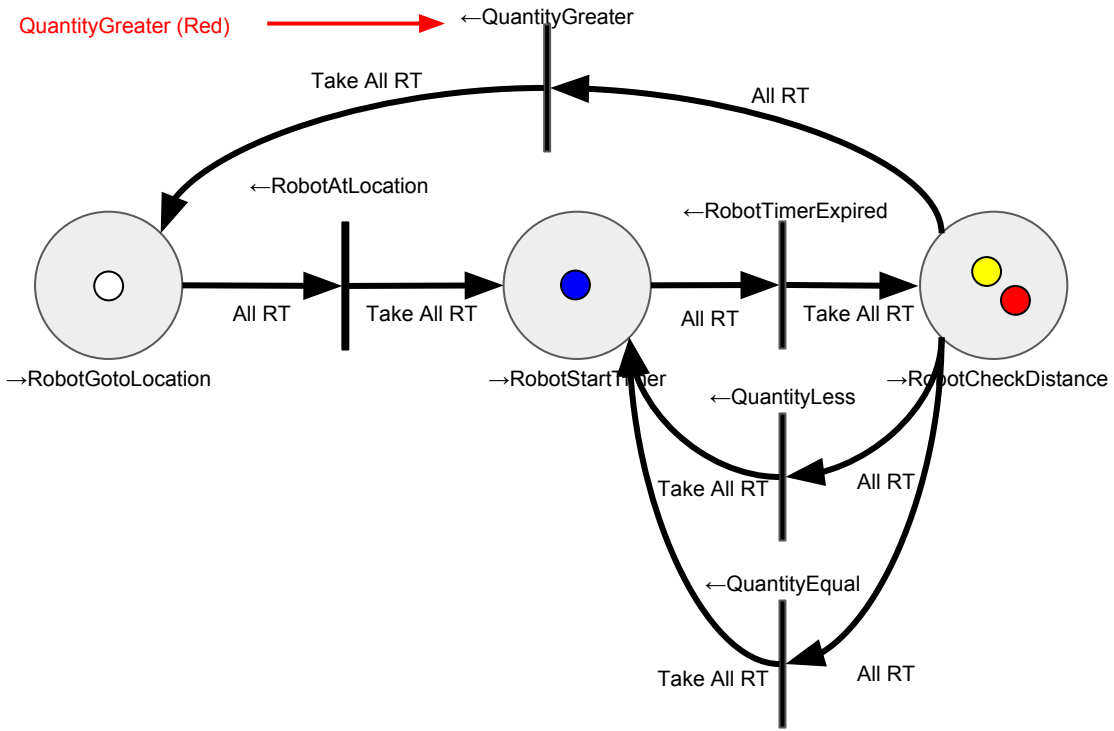
←QuantityGreater would be marked as received, but the transition would not execute because the red token is not in the place with →RobotCheckDistance

Quiz 6-5 Solution Part II



QUIZ 6-6: What would happen if the following input event was received?

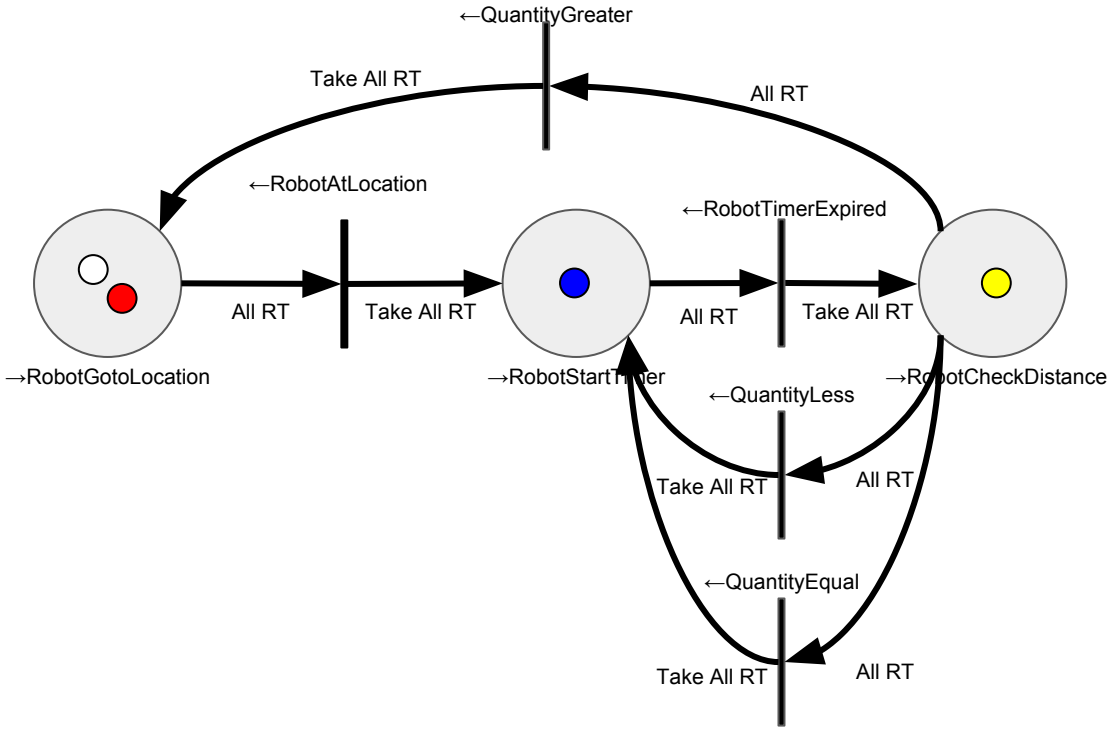
Quantity Greater, Relevant Tokens = Red robot token



Quiz 6-6 Solution Part I

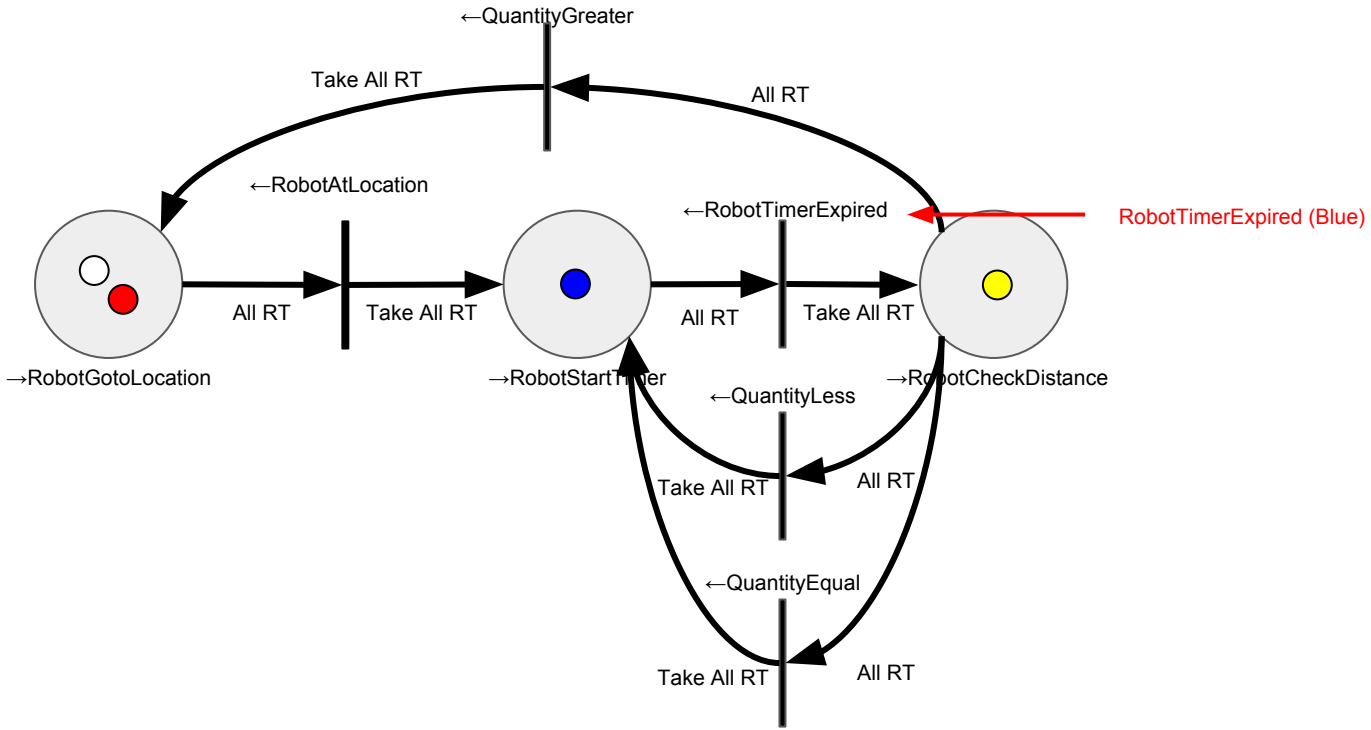
The transition with ←QuantityGreater would execute and move the red token from the place with →RobotCheckDistance to the place with →RobotGotoLocation

Quiz 6-6 Solution Part II



QUIZ 6-7: What would happen if the following input event was received?

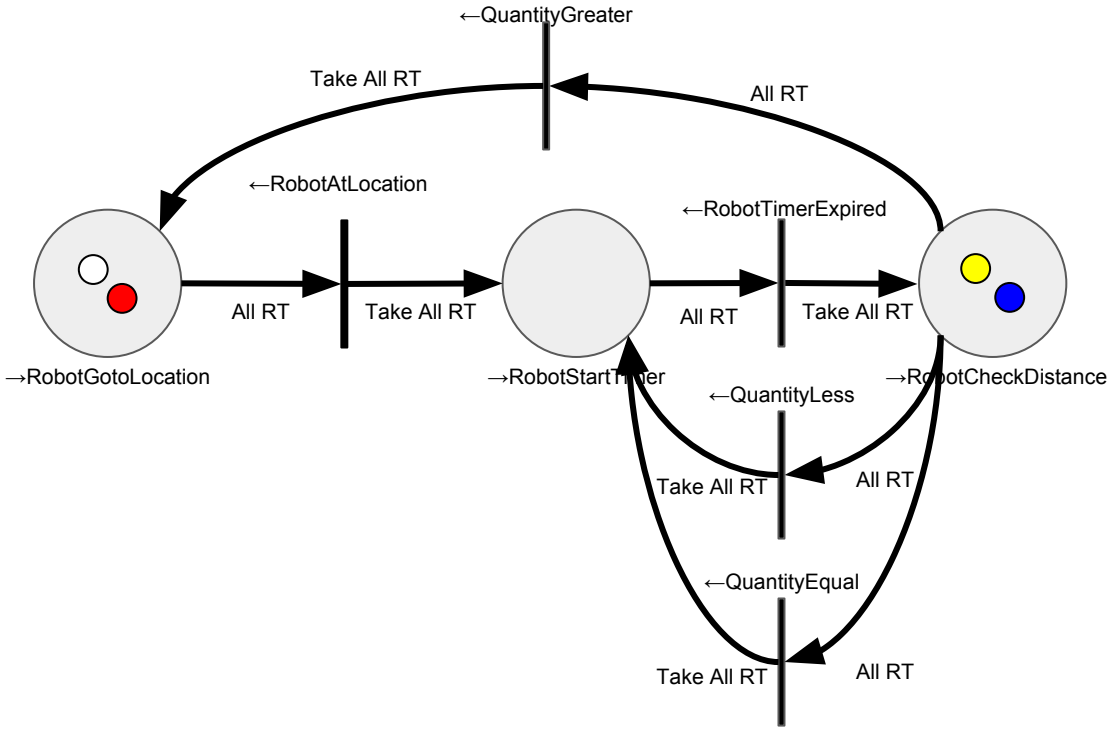
Robot Timer Expired, Relevant Tokens = Blue robot token



Quiz 6-7 Solution Part I

The transition with ←RobotTimerExpired would execute and move the blue token from the place with →RobotStartTimer to the place with →RobotCheckDistance

Quiz 6-7 Solution Part II



We have now discussed how to move specific robot tokens through the SPN, but have not talked about how the actual commands are sent to robots to make them do things.

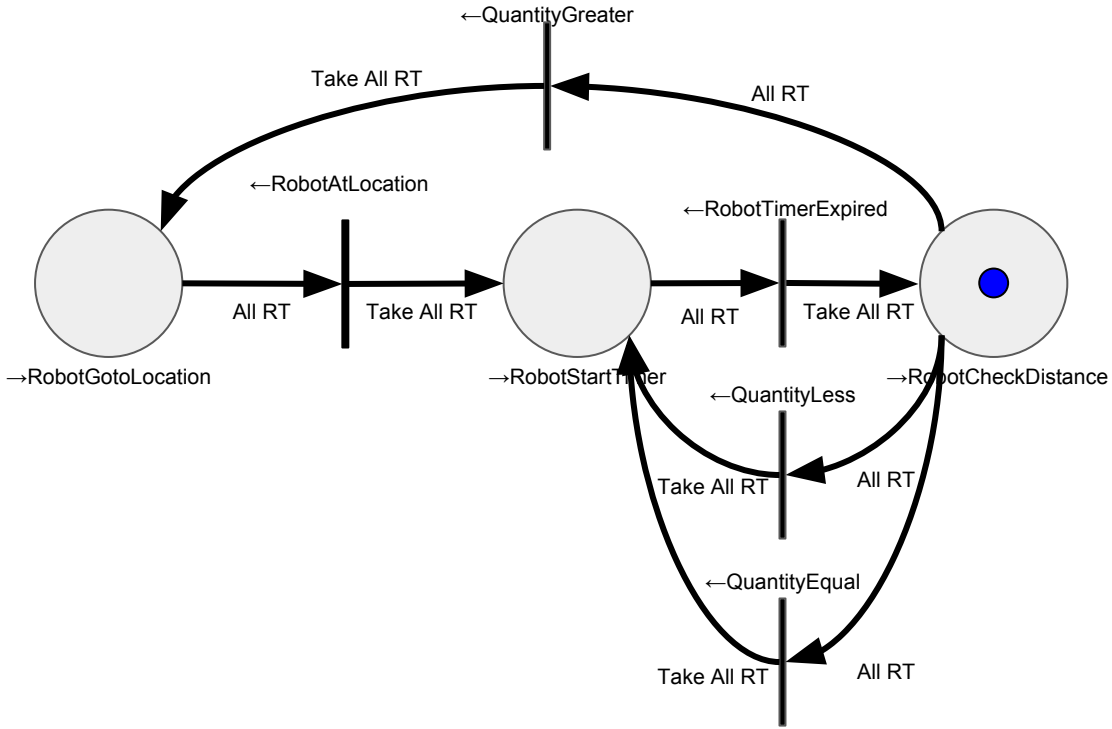
When a token or set of tokens enter a place, each output event on the place is activated by the tokens. Output events do different things depending on what tokens activate it.

Consider the output event →RobotGotoLocation:

- If activated by a robot token, the robot is told to move to the location
- If activated by a generic token, nothing is done

QUIZ 6-8: What would happen in the following scenario?

Blue robot token enters the place with "→RobotCheckDistance"

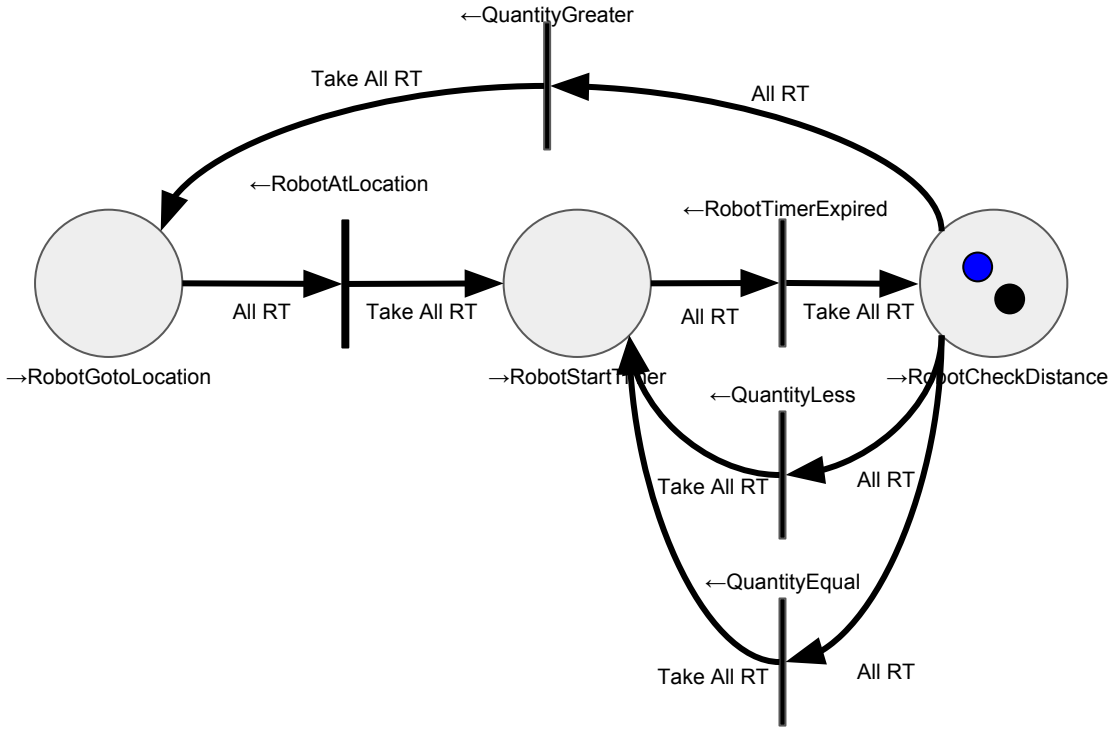


Quiz 6-8 Solution

Blue robot is commanded to check its distance from the desired location, and generate a ←QuantityGreater, ←QuantityLess, or ←QuantityEqual accordingly. The generated input event will list blue robot as a relevant token.

QUIZ 6-9: What would happen in the following scenario?

Blue robot token and 1 generic token both enter the place with "→RobotCheckDistance"



Quiz 6-9 Solution

Blue robot will be commanded as in the previous example.

The generic token will be ignored.

Now we will discuss how SPNs start and how SPNs end.

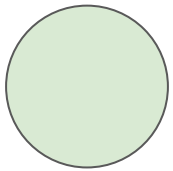
Start Place

Each SPN has exactly one start place where the following tokens are placed when the SPN is started

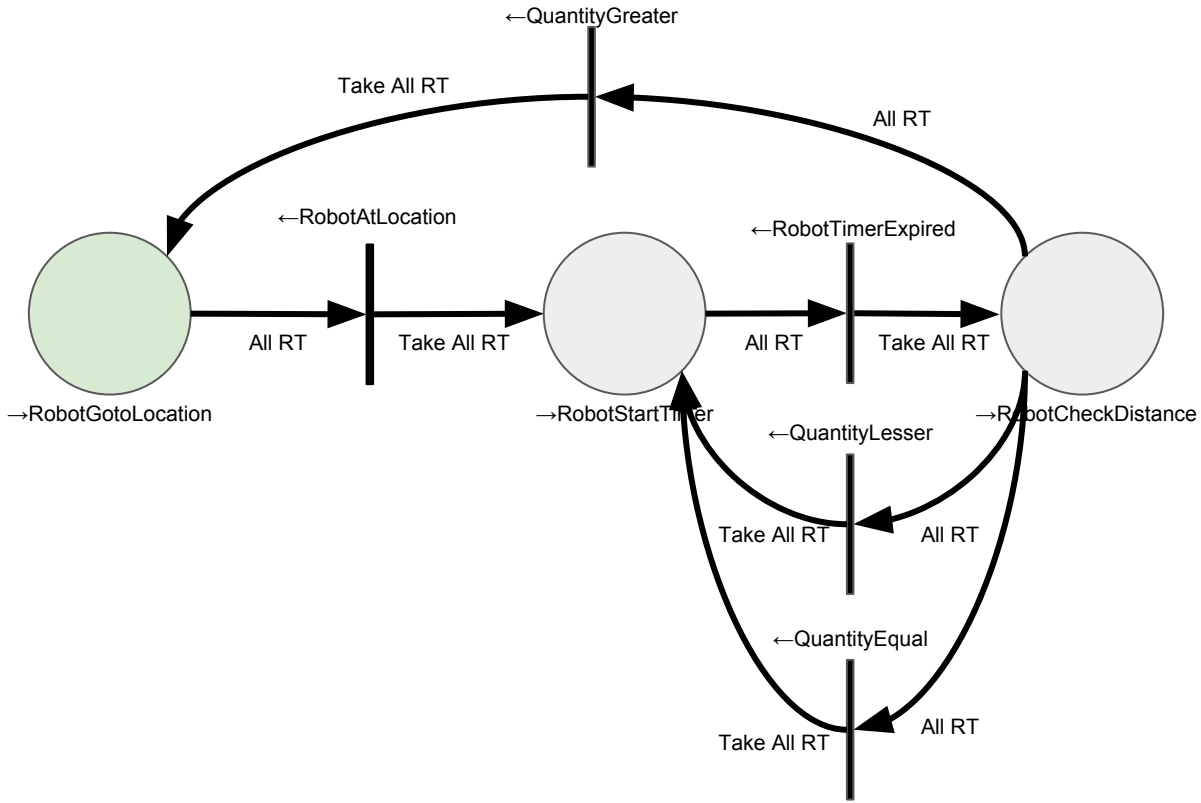
- 1 Generic token
- 1 Robot token for each existing robot

If the start place has any output events, all of the tokens activate them.

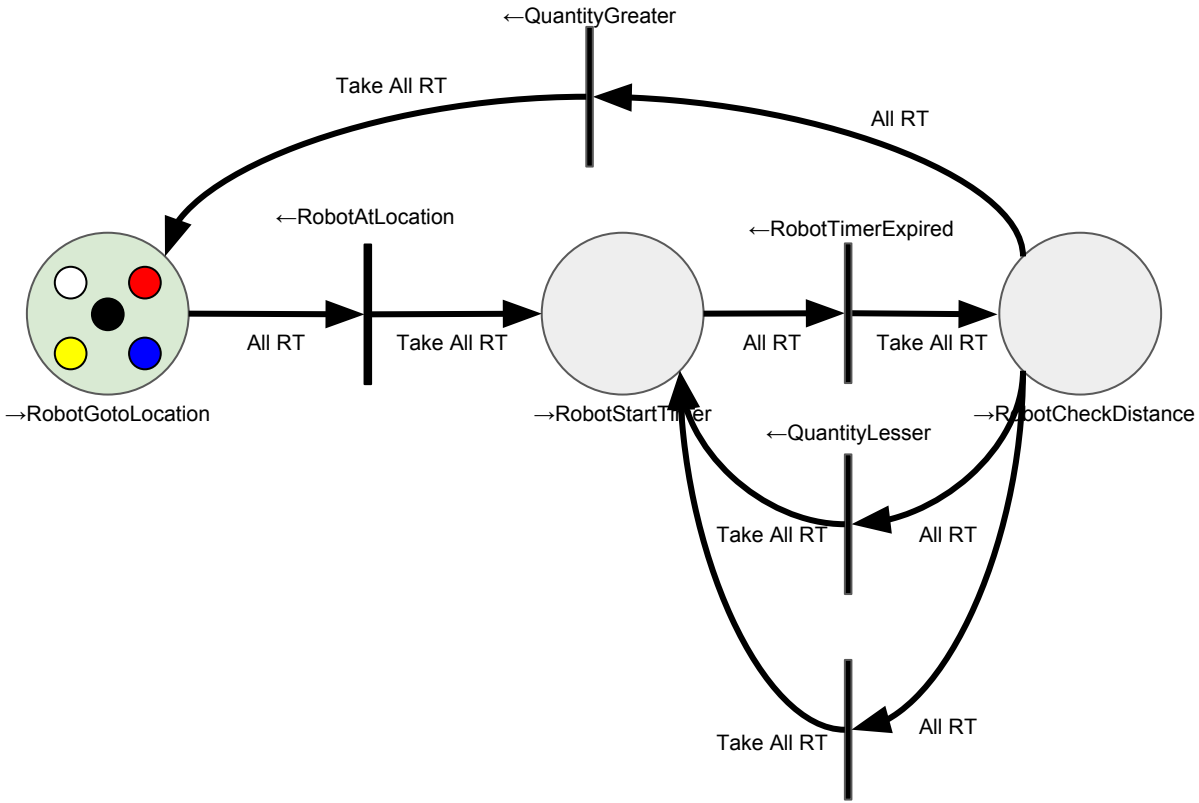
The start place is colored green.



Let’s make the place with the “→RobotGotoLocation” the start place.



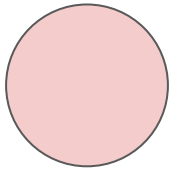
Now if we start the plan with White, Red, Yellow, and Blue robots connected, the following tokens are added to the start place.



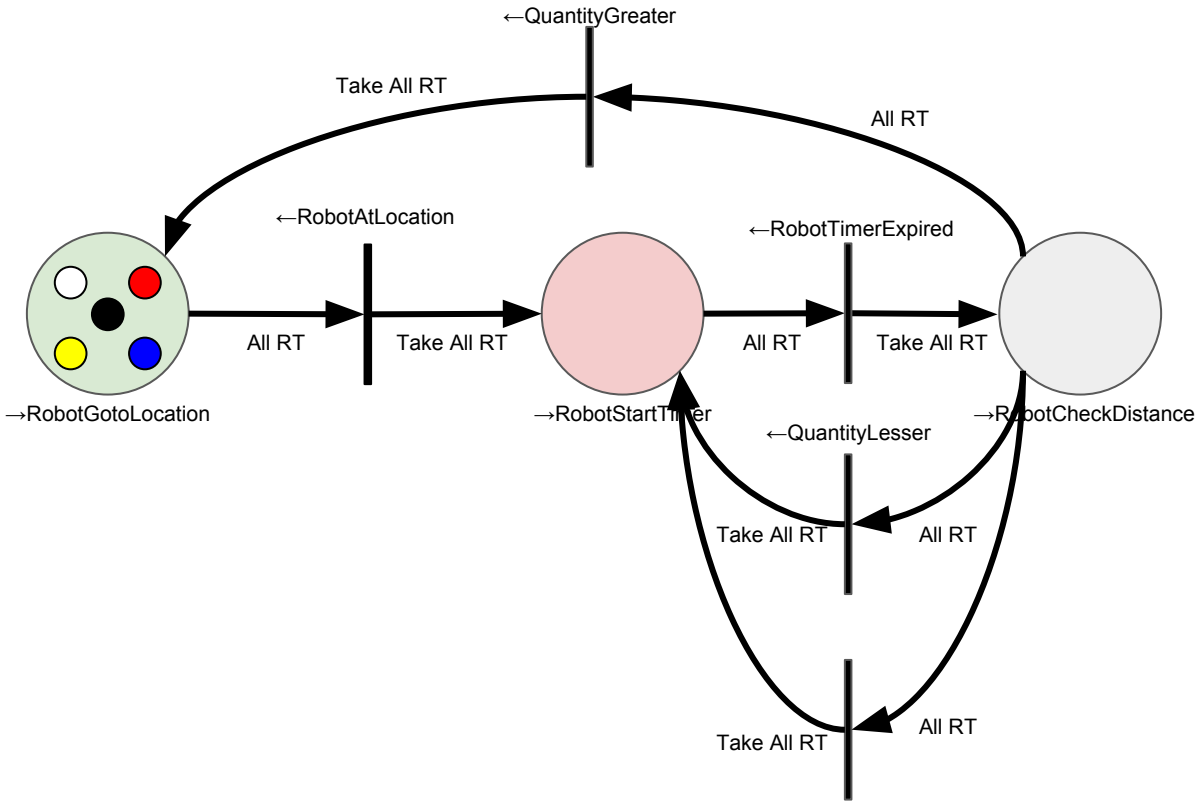
End Place

Each SPN has any number of end places which terminate all plan activity when a token enters it, such as stopping any robot movement initiated by that SPN.

End places are colored red.



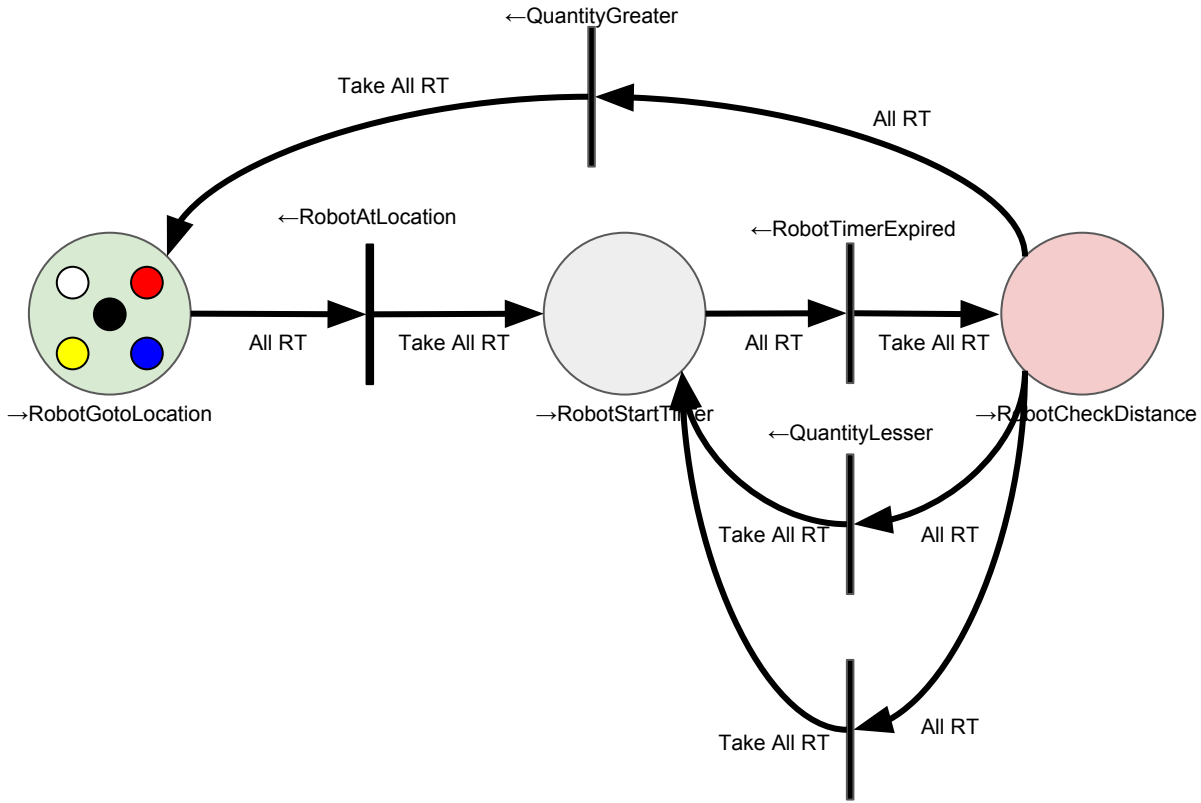
QUIZ 6-10: What will happen if we make →RobotStartTimer an End place?



Quiz 6-10 Solution

The plan will end as soon as the first robot reaches its destination.

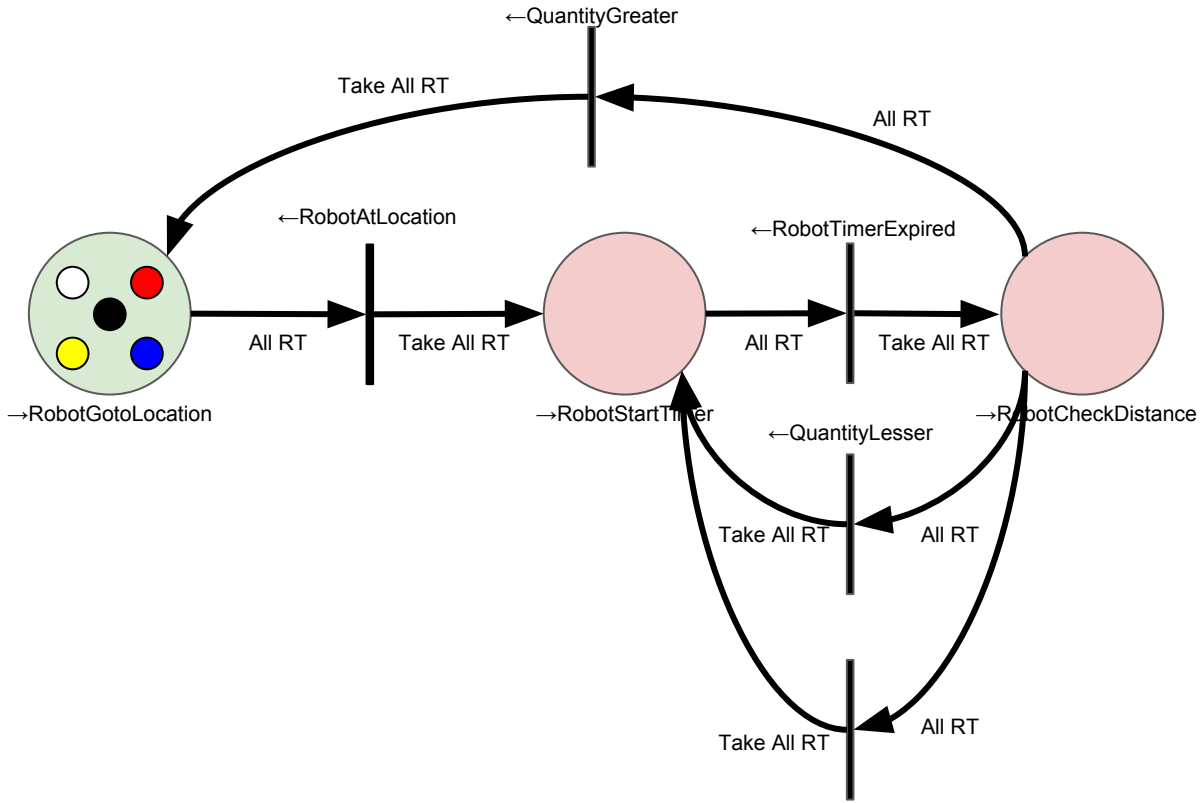
QUIZ 6-11: What will happen if we make →RobotCheckDistance an End place?



Quiz 6-11 Solution

The plan will end as soon as the first robot timer expires.

QUIZ 6-12: What will happen if we make →RobotStartTimer and →RobotCheckDistance End places?



Quiz 6-12 Solution

The plan will end as soon as the first robot reaches its destination.

Currently there is no place which makes a good end place. Next we will talk about interacting with the operator for information, such as when to end a SPN.