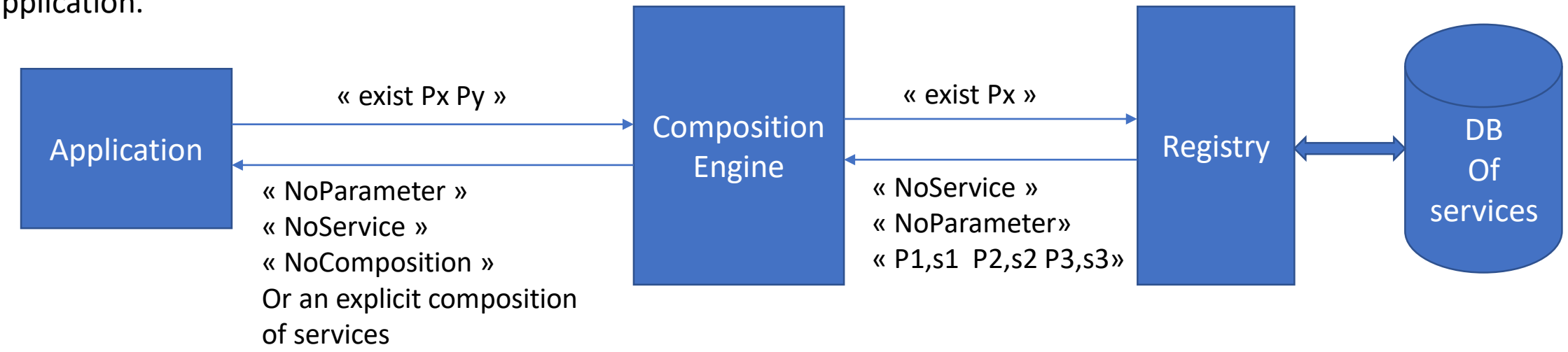
Description of the system

The compositon engine get three types of responses from the registry:

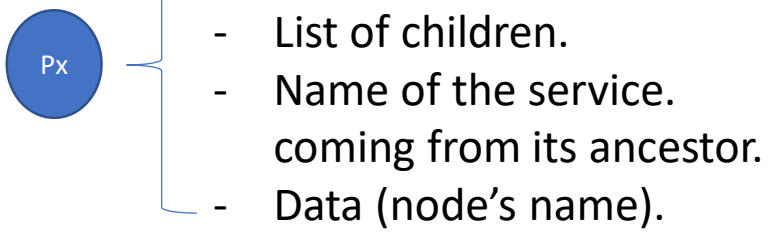
→NoParameter : if Px doesn't exist in the registry, so RETURN NoParameter to the application.
→NoService : if Px exist but there is no services that take Px as entry, so RETURN NoService to the application.
→Otherwise, the engine get the services that have Px as entry and the corresponding output, then start the Tree algorithm. The form of the response is « Output,servicename Output,servicename … ». If the composition engine find a composition of services, it returns the explicit composition of services to the application, otherwise it will return « NoComposition ».

The algorithm ran by the composition engine will create a tree starting from Px as a root and search for a path (composition on arcs that represent services) to Py. If it doesn't find a path it will RETURN NoComposition to the application.



Description of the Tree Algorithm

Each node of the tree represents a parameter, and the arcs represent the service from a parameter to another. The name of a service going from Px to Px' is stocked in the node Px'.

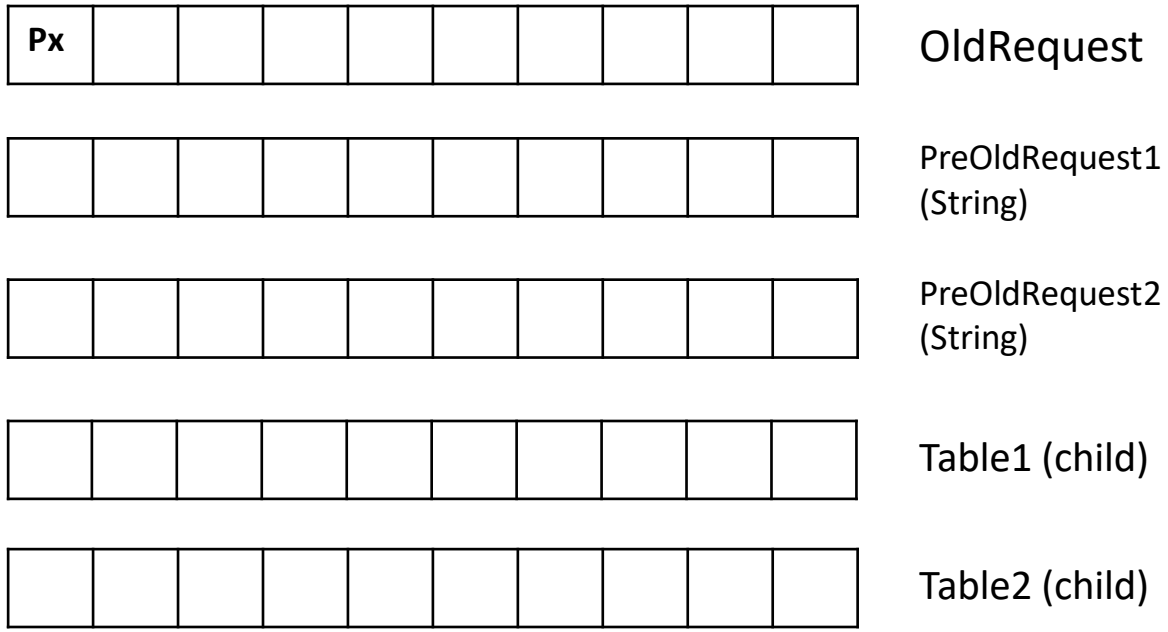


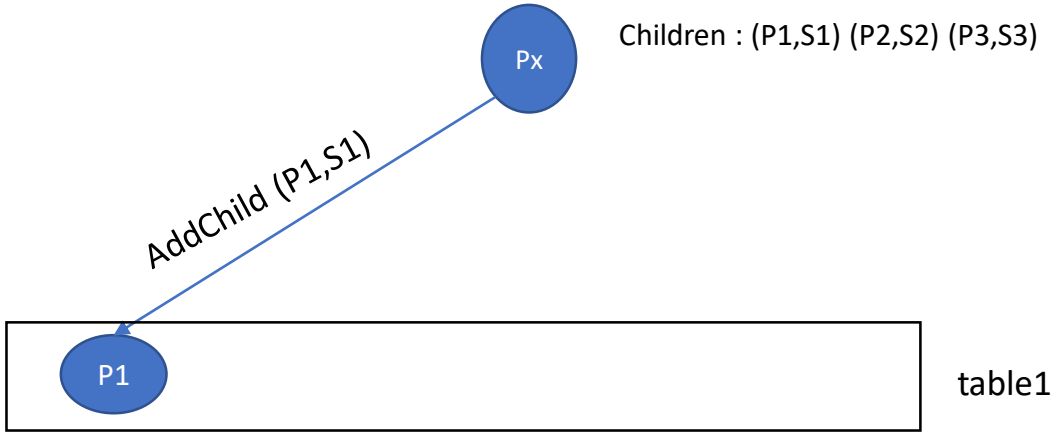
We define also 5 tables :

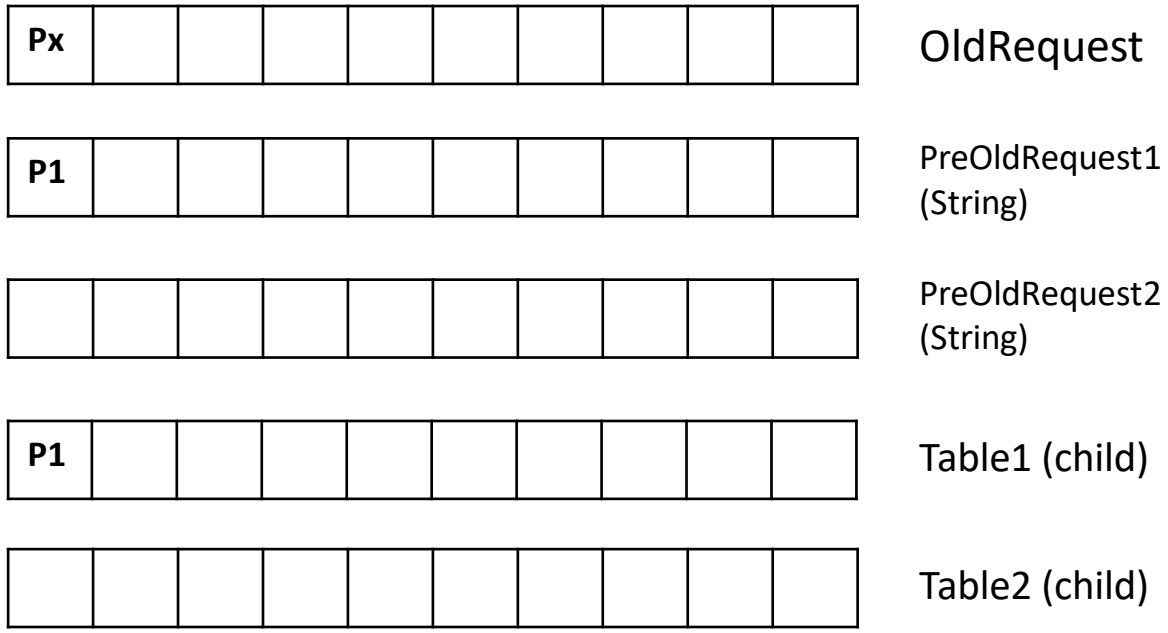
→OldRequest : allow us to verify if the algorithm already searched for children of the node.
→PreOldRequest1(String) and table1(child) / PreOldRequest2(String) and table2(child) : in each round, one table contains nodes, that we search for its children and stock them in the other table, when this process finished, we add all parents to OldRequest and we reset the table.

Children : (P1,S1) (P2,S2) (P3,S3)

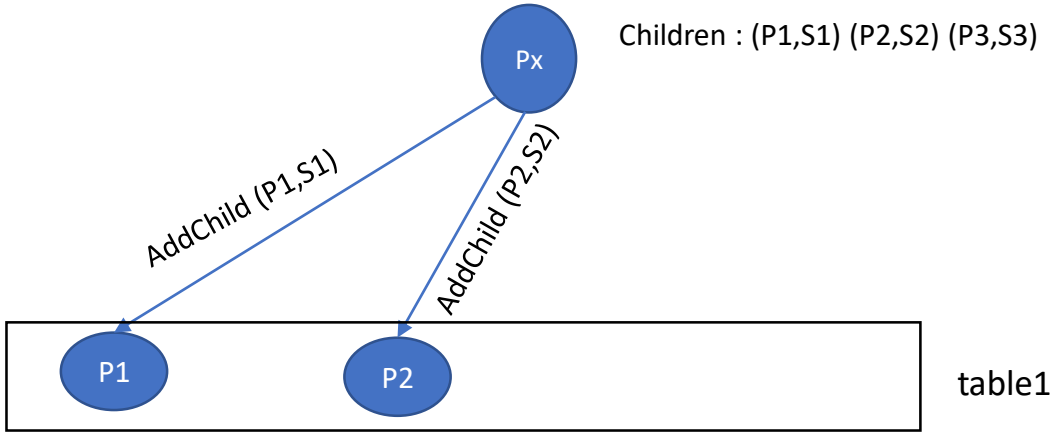
Create the root, and add its name to OldRequest, then start creating its children.

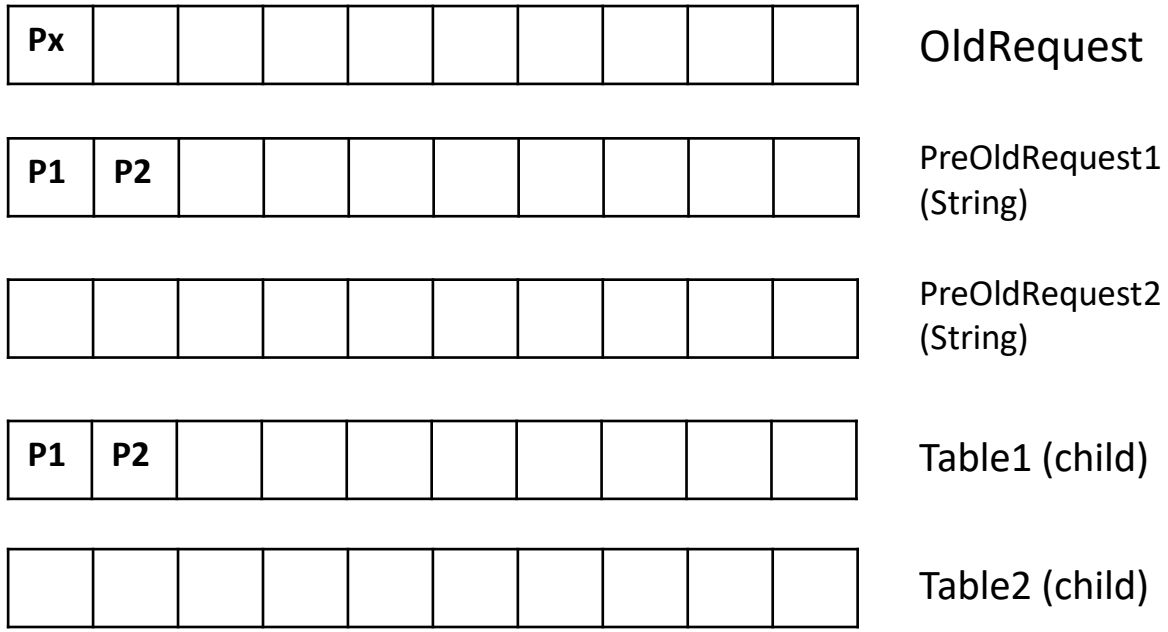




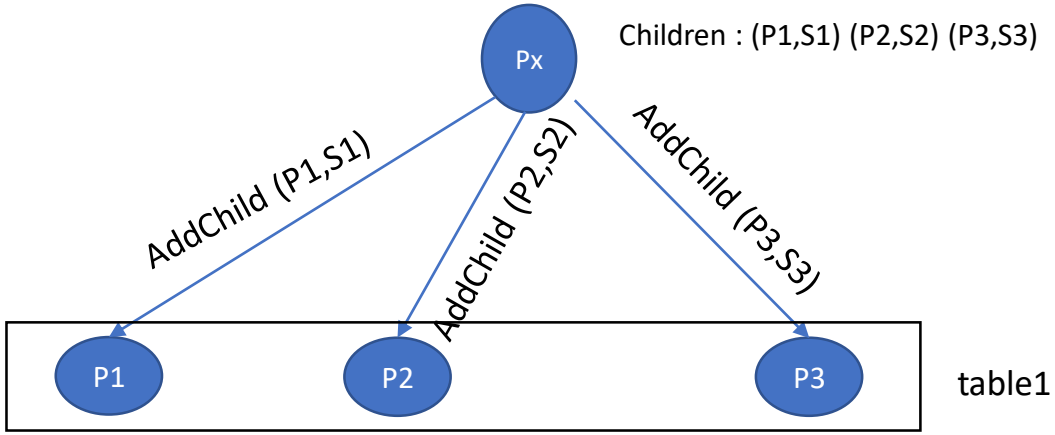


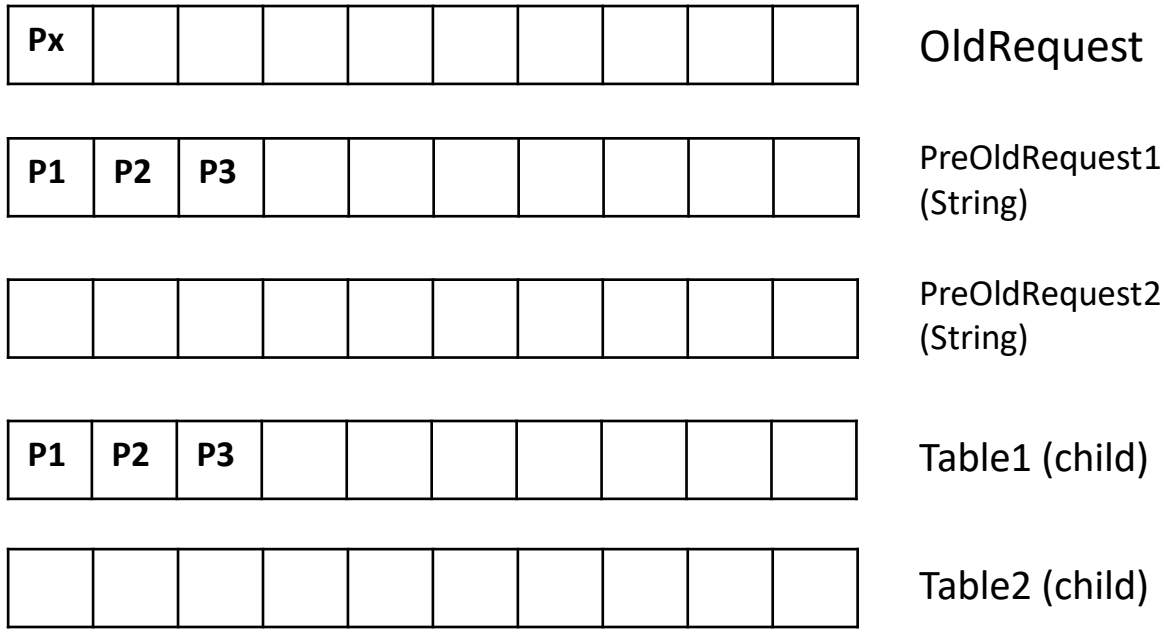
In each child, we verify if it is equal to the destination parameter wanted by the application.
If yes, found = true, and add the child to the table, then exit the loop of adding children of the node, then stop the algorithm. (this will prevent concluding that there is no composition in the case of finding the destination in the first row of the table[by verifying if the table is empty or not]).



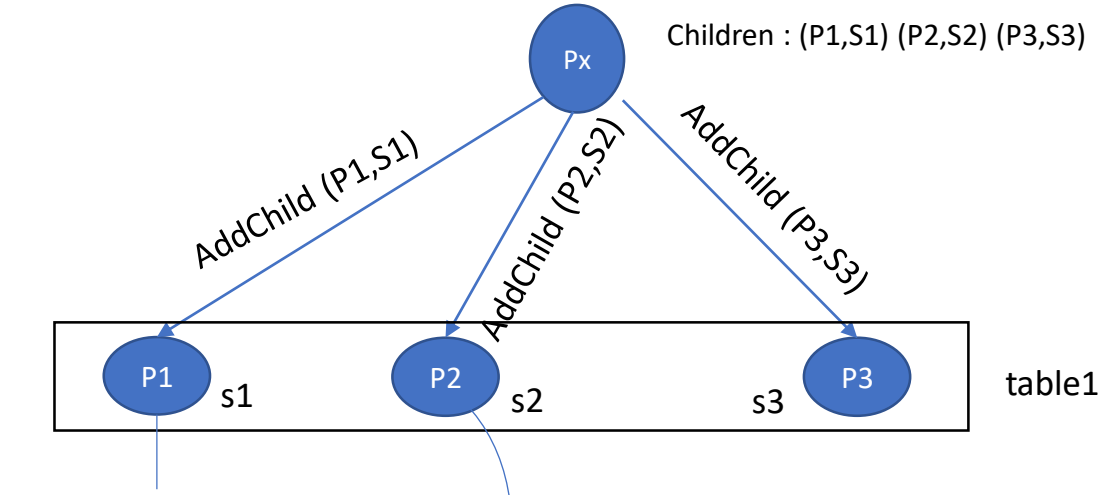


In each child, we verify if it is equal to the destination parameter wanted by the application.
If yes, found = true, and add the child to the table, then exit the loop of adding children of the node, then stop the algorithm. (this will prevent concluding that there is no composition in the case of finding the destination in the first row of the table[by verifying if the table is empty or not]).





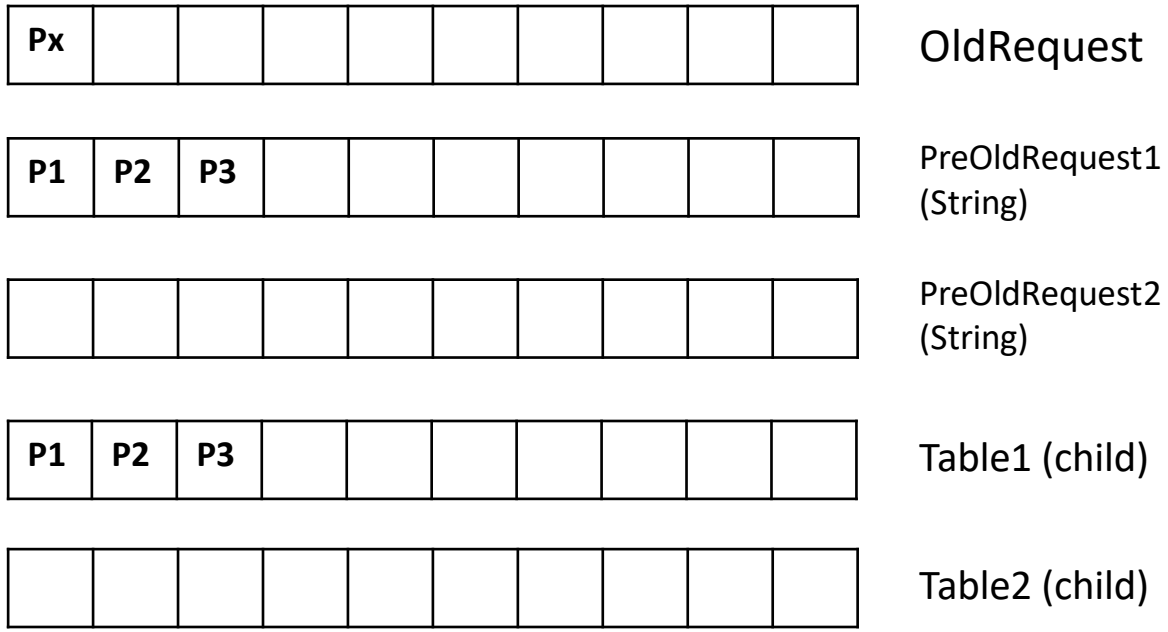
In each child, we verify if it is equal to the destination parameter wanted by the application.
If yes, found = true, and add the child to the table, then exit the loop of adding children of the node, then stop the algorithm. (this will prevent concluding that there is no composition in the case of finding the destination in the first row of the table[by verifying if the table is empty or not]).

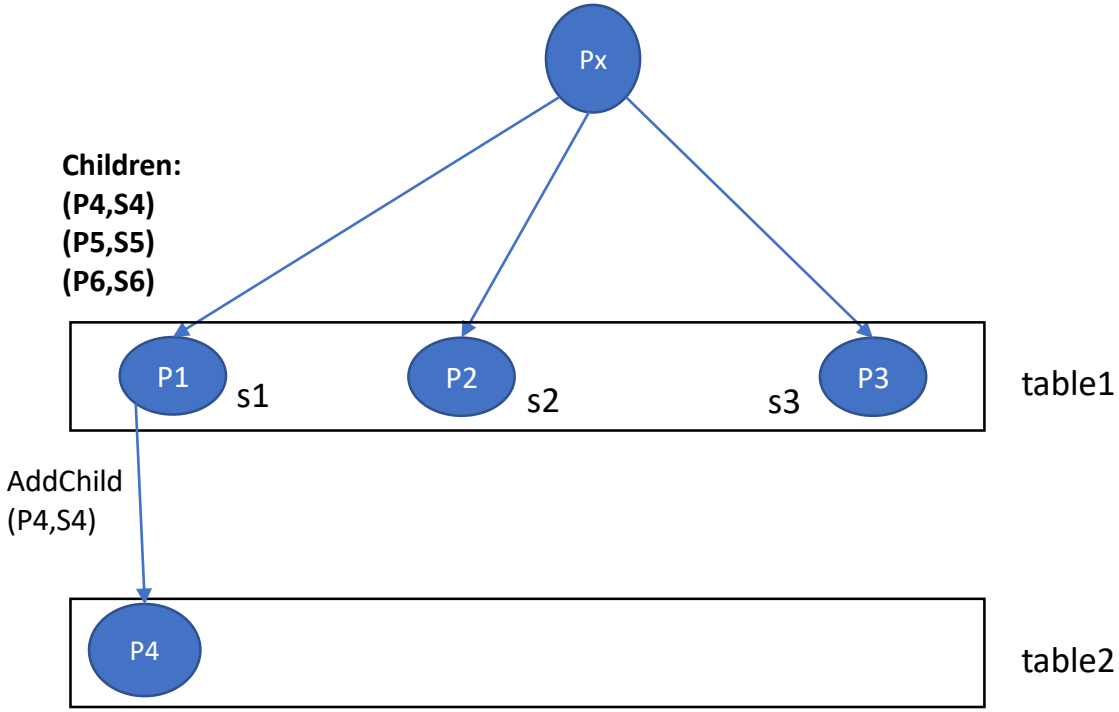


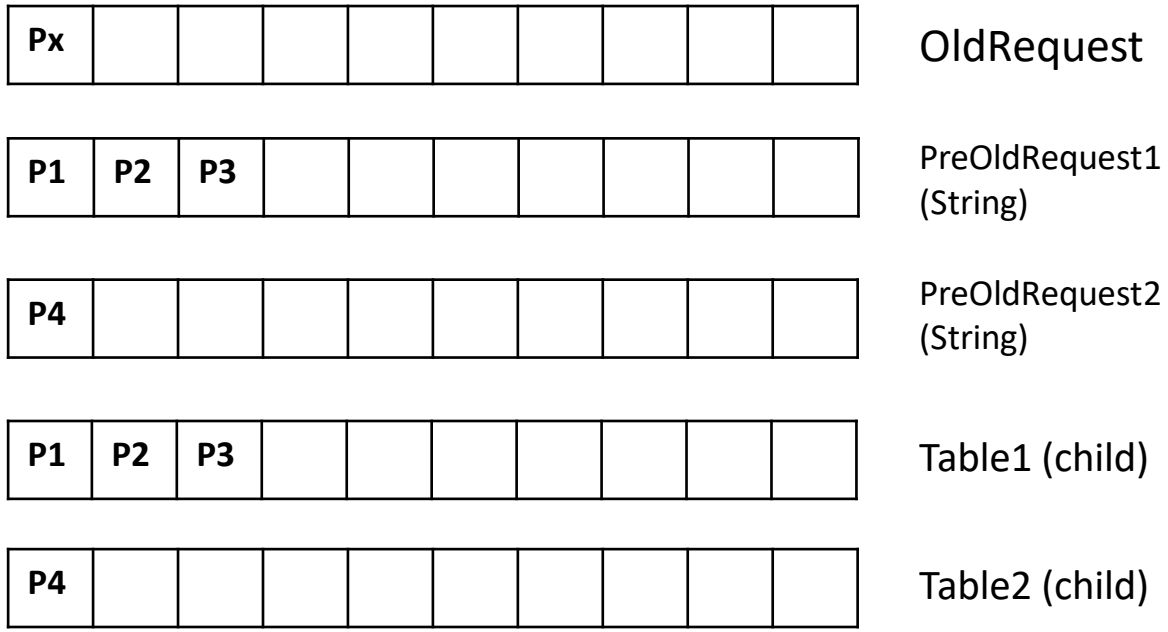
If P1 doesn't exist in **OldRequest**,then send message to the registry to get its children.

-> Children : (P4,S4) (P5,S5) (P6,S6)

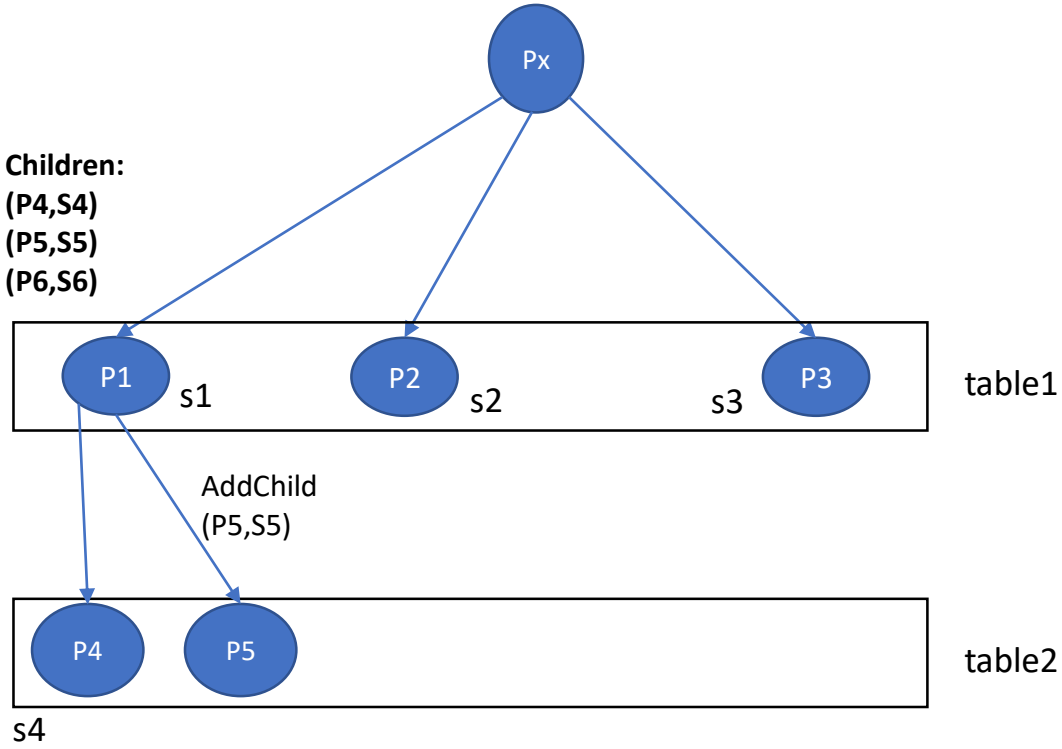
-> If « NoService» or P1 exist in OldRequest, then continue to P2

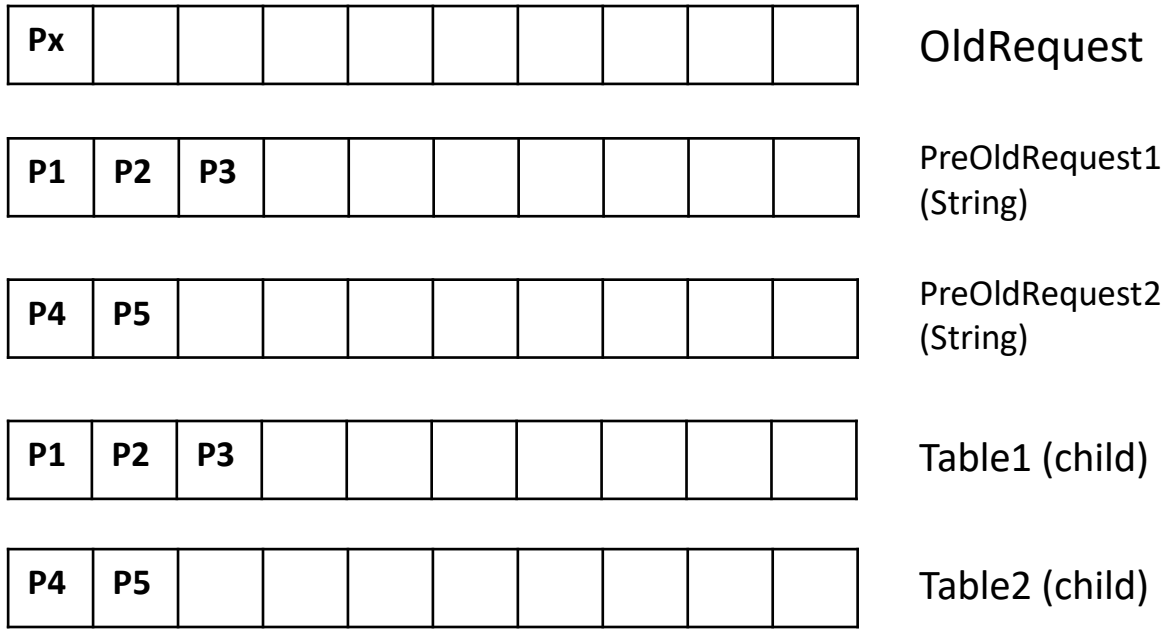




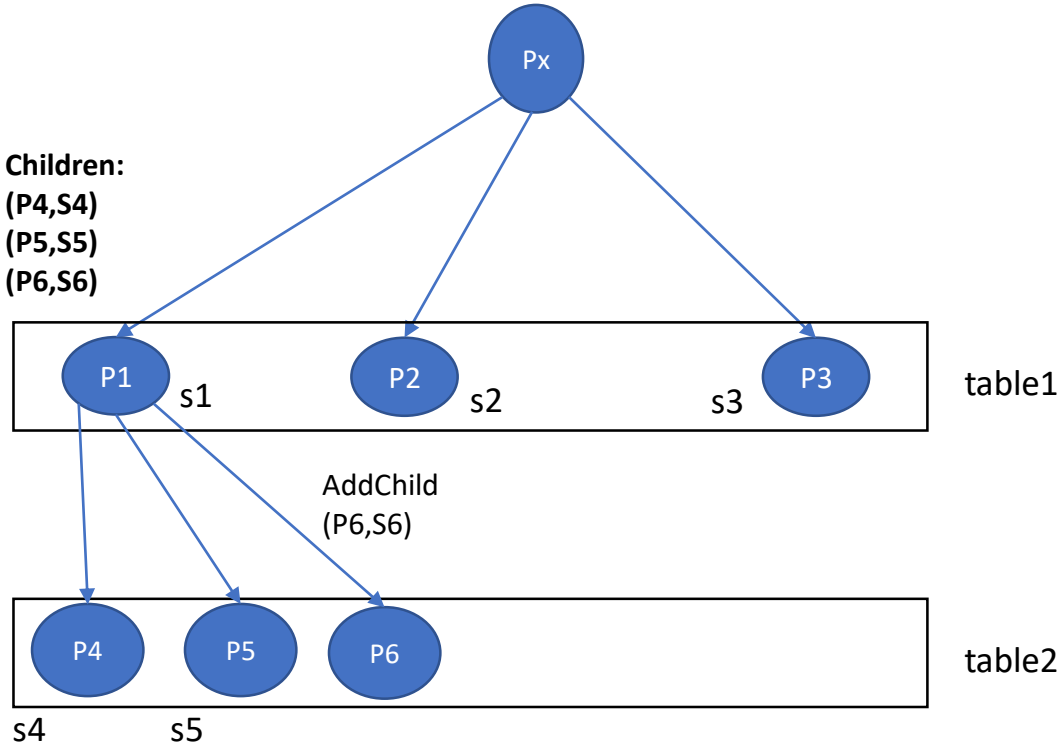


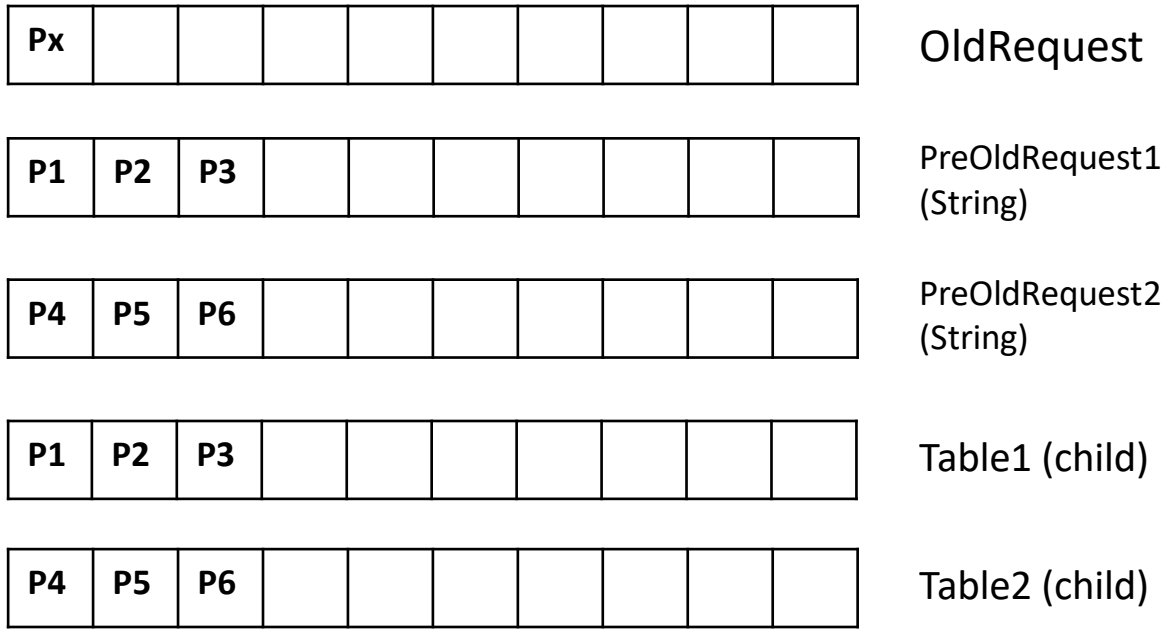
In each child, we verify if it is equal to the destination parameter wanted by the application.
If yes, found = true, and add the child to the table, then exit the loop of adding children of the node, then stop the algorithm. (this will prevent concluding that there is no composition in the case of finding the destination in the first row of the table[by verifying if the table is empty or not]).



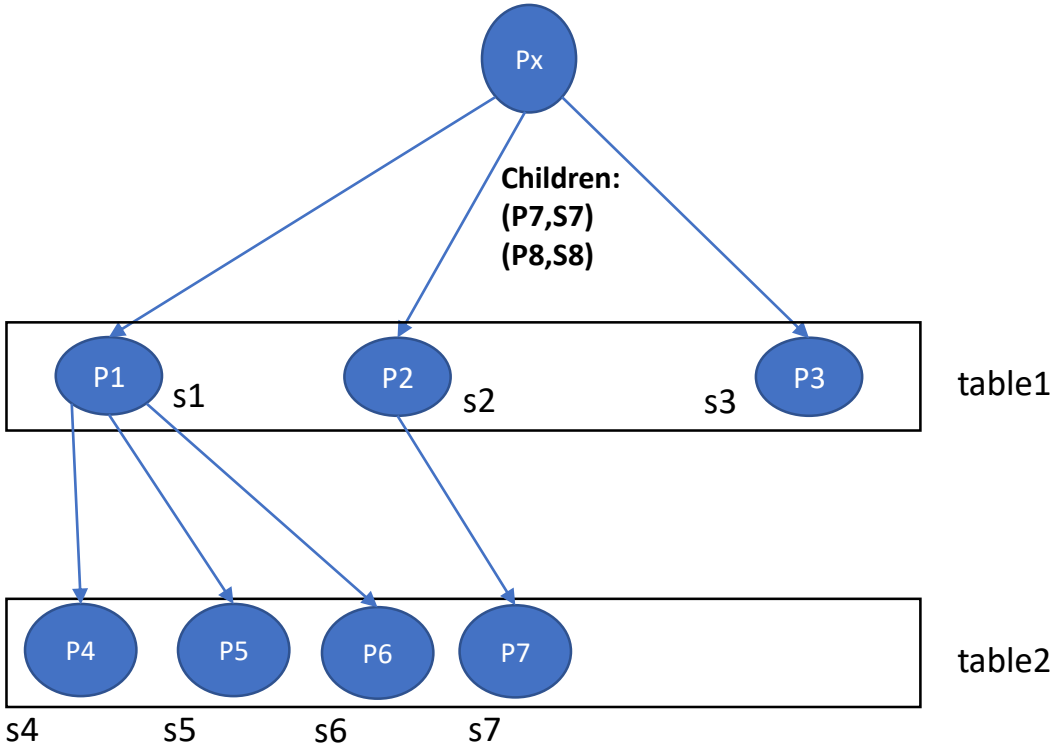


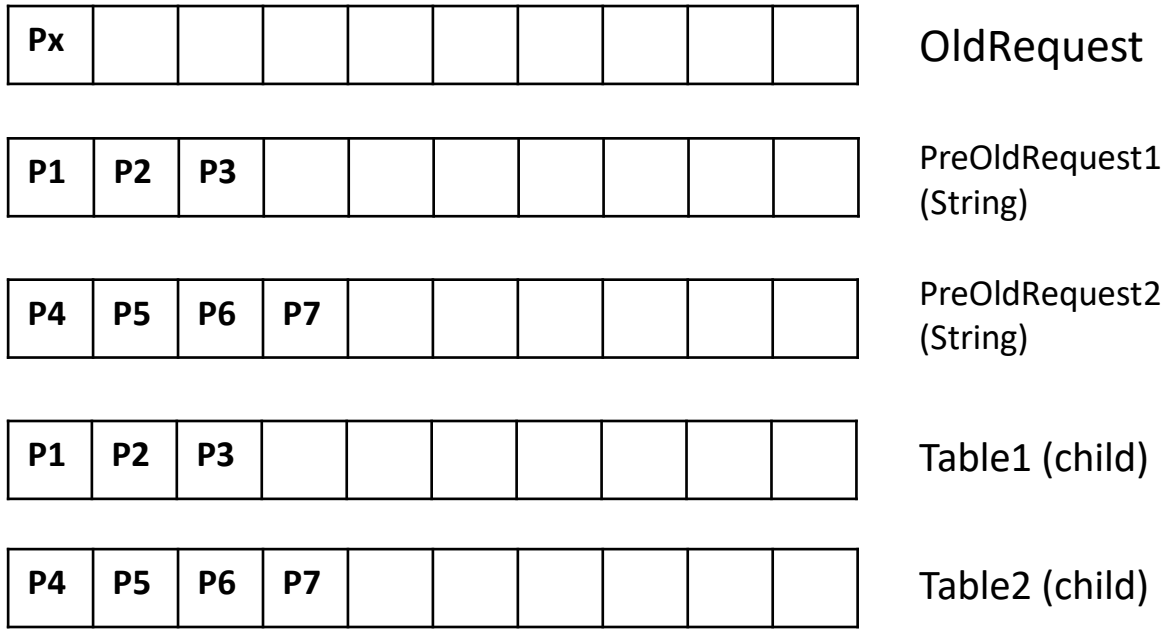
In each child, we verify if it is equal to the destination parameter wanted by the application.
If yes, found = true, and add the child to the table, then exit the loop of adding children of the node, then stop the algorithm. (this will prevent concluding that there is no composition in the case of finding the destination in the first row of the table[by verifying if the table is empty or not]).



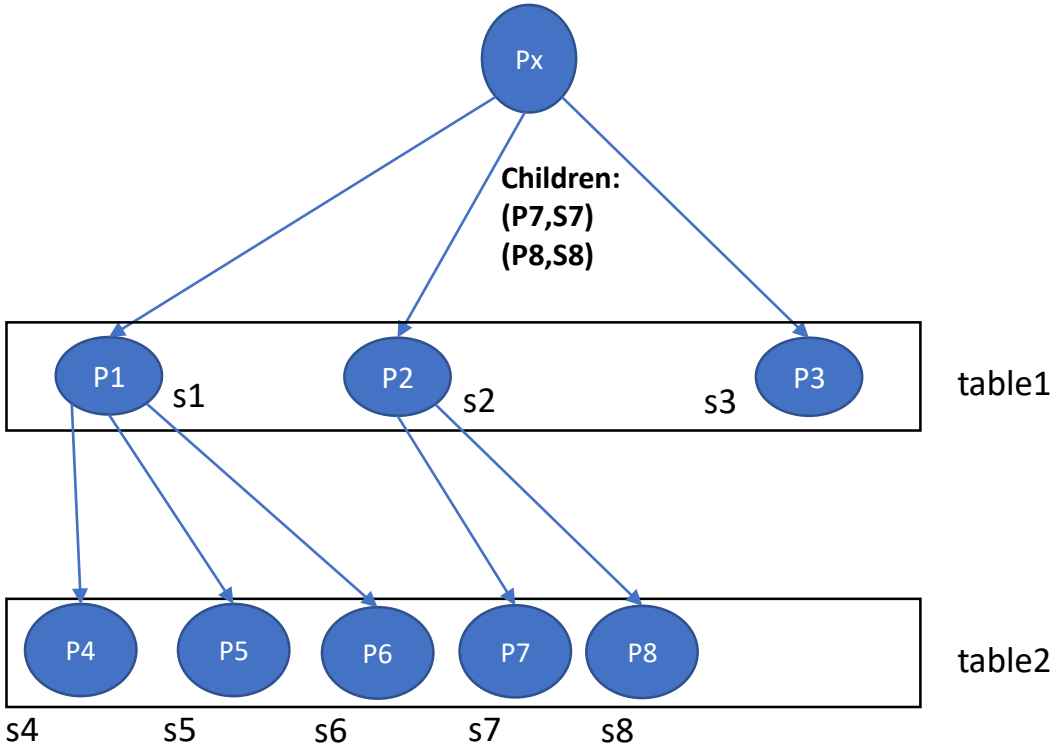


In each child, we verify if it is equal to the destination parameter wanted by the application.
If yes, found = true, and add the child to the table, then exit the loop of adding children of the node, then stop the algorithm. (this will prevent concluding that there is no composition in the case of finding the destination in the first row of the table[by verifying if the table is empty or not]).



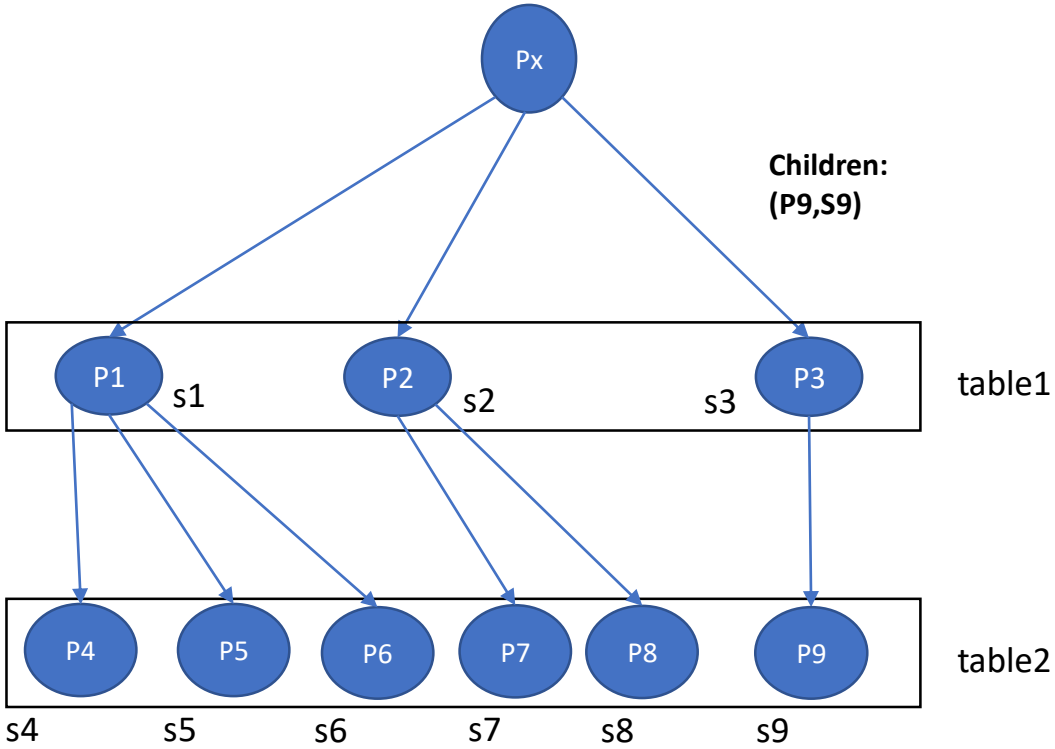


In each child, we verify if it is equal to the destination parameter wanted by the application.
If yes, found = true, and add the child to the table, then exit the loop of adding children of the node, then stop the algorithm. (this will prevent concluding that there is no composition in the case of finding the destination in the first row of the table[by verifying if the table is empty or not]).



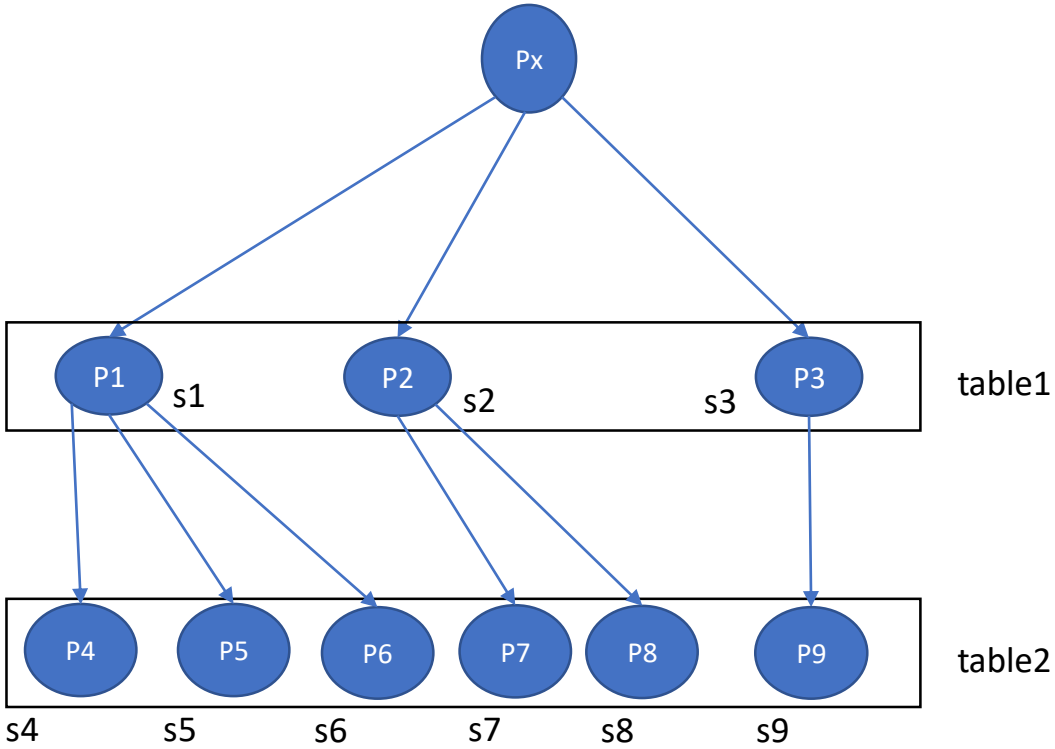
Px										OldRequest
P1	**P2**	**P3**								PreOldRequest1 (String)
P4	**P5**	**P6**	**P7**	**P8**						PreOldRequest2 (String)
P1	**P2**	**P3**								Table1 (child)
P4	**P5**	**P6**	**P7**	**P8**						Table2 (child)

In each child, we verify if it is equal to the destination parameter wanted by the application.
If yes, found = true, and add the child to the table, then exit the loop of adding children of the node, then stop the algorithm. (this will prevent concluding that there is no composition in the case of finding the destination in the first row of the table[by verifying if the table is empty or not]).



Px										OldRequest
P1	**P2**	**P3**								PreOldRequest1 (String)
P4	**P5**	**P6**	**P7**	**P8**	**P9**					PreOldRequest2 (String)
P1	**P2**	**P3**								Table1 (child)
P4	**P5**	**P6**	**P7**	**P8**	**P9**					Table2 (child)

In each child, we verify if it is equal to the destination parameter wanted by the application.
If yes, found = true, and add the child to the table, then exit the loop of adding children of the node, then stop the algorithm. (this will prevent concluding that there is no composition in the case of finding the destination in the first row of the table[by verifying if the table is empty or not]).



Px									

OldRequest

P1	**P2**	**P3**							

PreOldRequest1 (String)

P4	**P5**	**P6**	**P7**	**P8**	**P9**				

PreOldRequest2 (String)

P1	**P2**	**P3**							

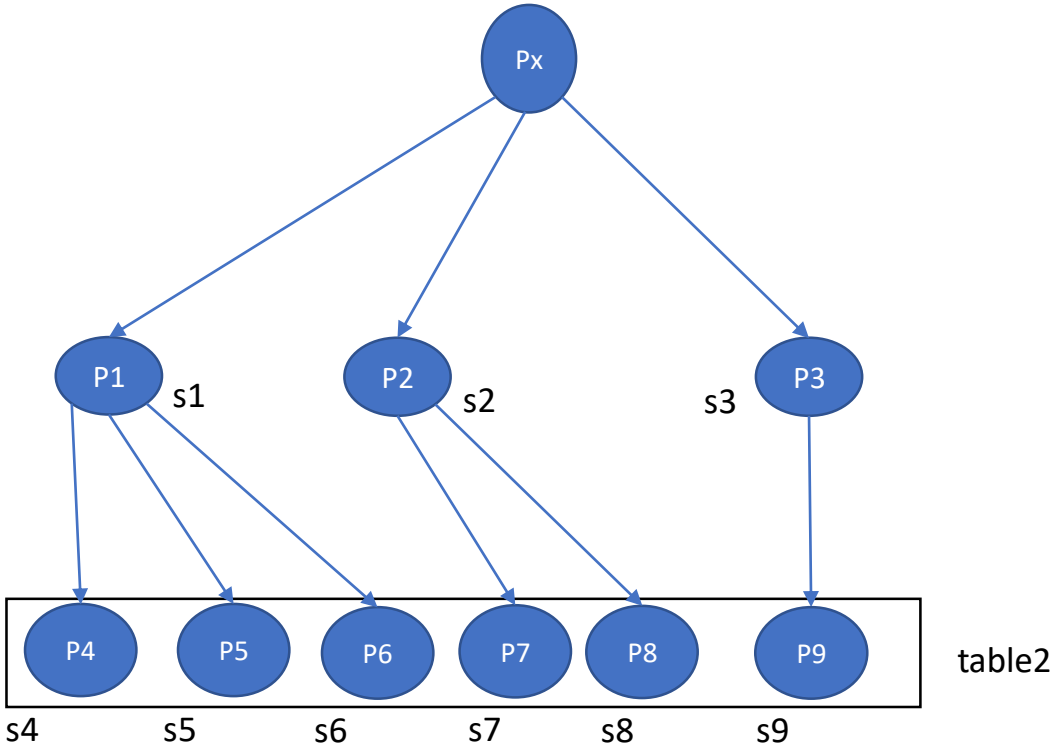
Table1 (child)

P4	**P5**	**P6**	**P7**	**P8**	**P9**				

Table2 (child)

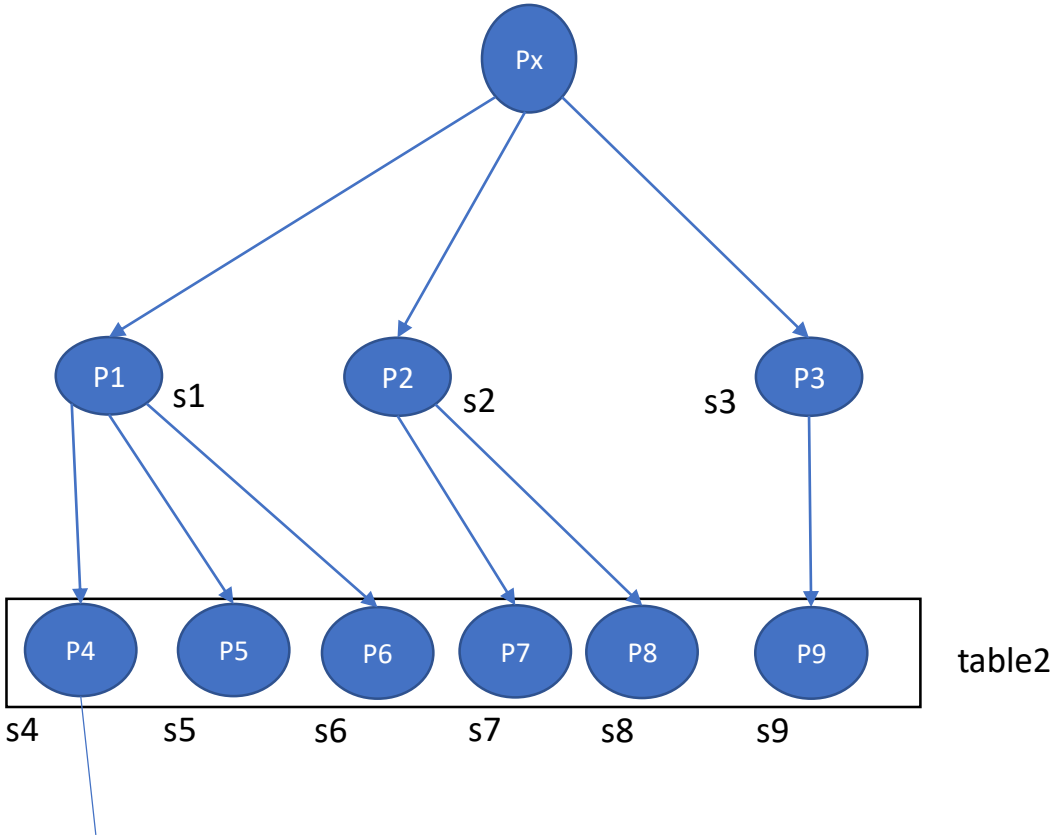
Verify if table 2 is empty. If yes, means that we searched in the whole database and we didn't find any parameter that correspond to the destination parameter wanted by the client.

→ STOP THE ALGORITHM AND RETURN : NoComposition



Px	**P1**	**P2**	**P3**							OldRequest
										PreOldRequest1 (String)
P4	**P5**	**P6**	**P7**	**P8**	**P9**					PreOldRequest2 (String)
										Table1 (child)
P4	**P5**	**P6**	**P7**	**P8**	**P9**					Table2 (child)

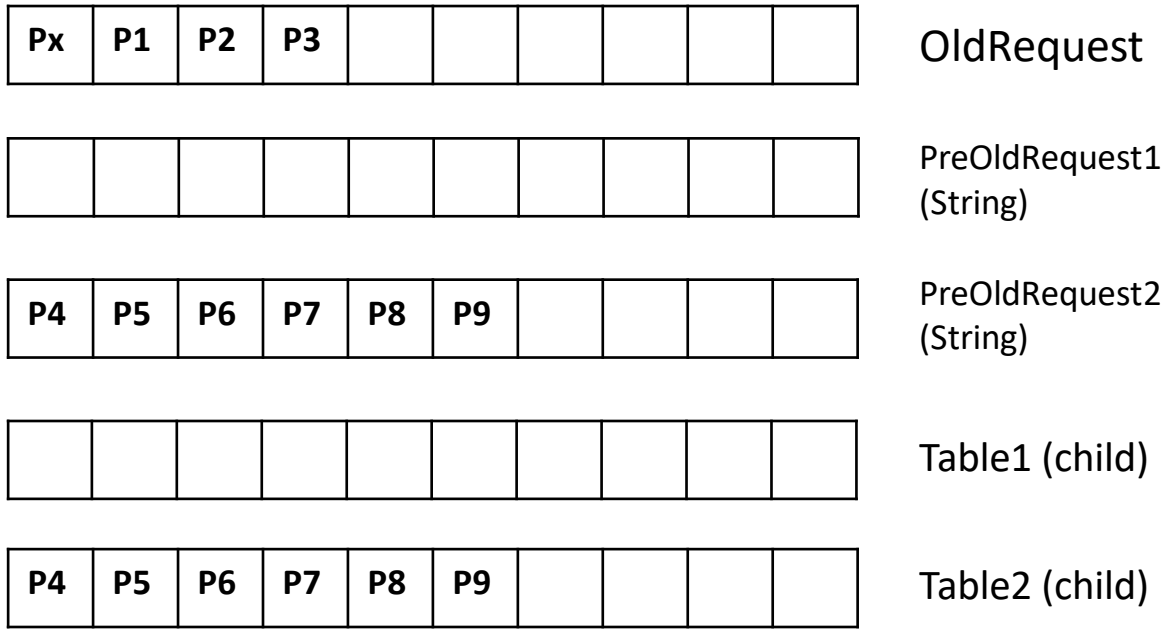
->Add PreOldRequest1 to OldRequest
->Empty PreOldRequest1
->Empty table1

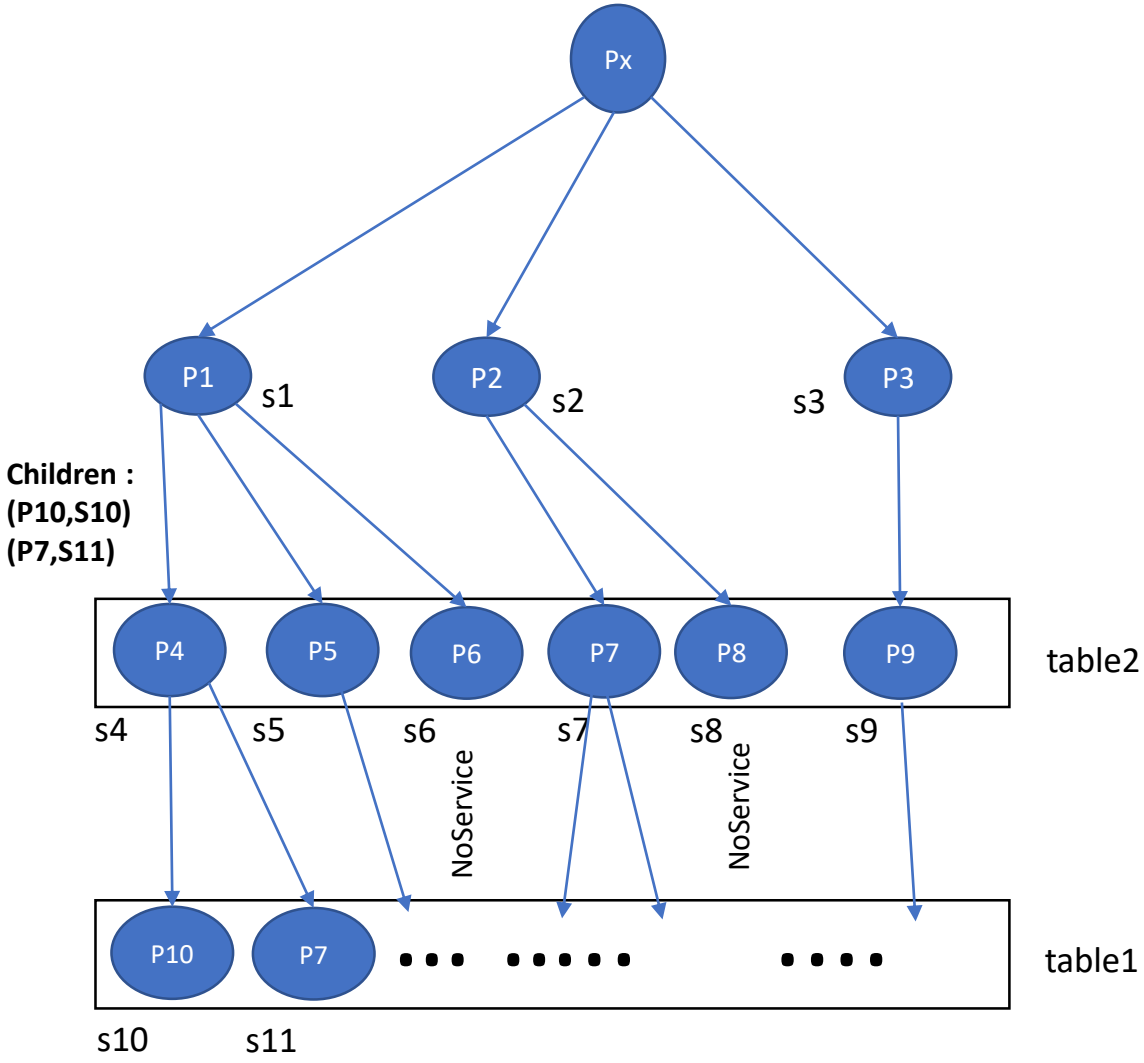


If P4 doesn't exist in **OldRequest**,then send message to the registry to get its children.

-> Children : (P10,S10) (P7,S11)

-> If « NoService» **or** P4 exist in OldRequest, then continue to P5





Px	P1	P2	P3						

OldRequest

P10	P7	.	.	.	.				

PreOldRequest1 (String)

P4	P5	P6	P7	P8	P9				

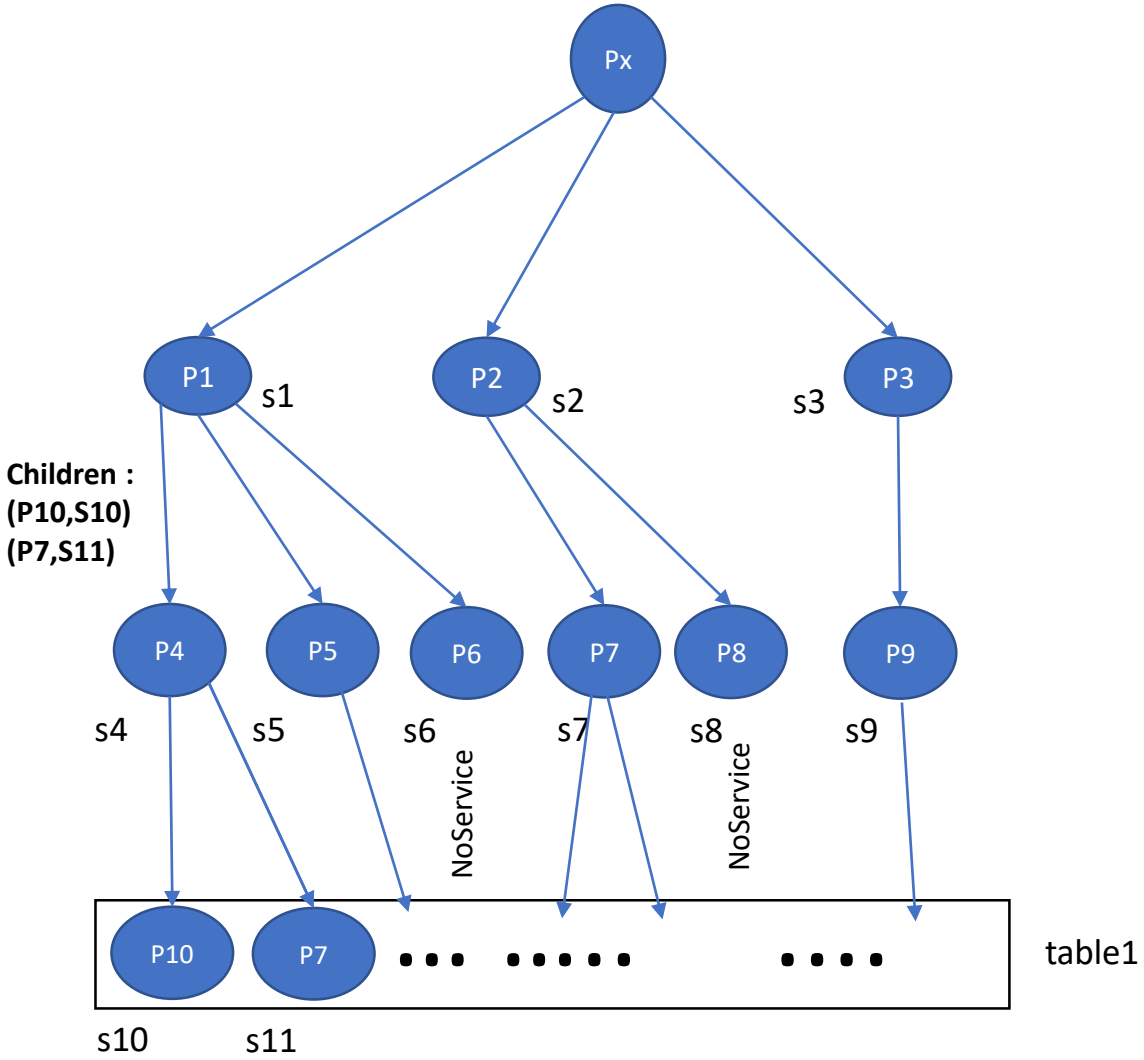
PreOldRequest2 (String)

P10	P7	.	.	.	.				

Table1 (child)

P4	P5	P6	P7	P8	P9				

Table2 (child)



Px	P1	P2	P3	P4	P5	P6	P7	P8	P9

OldRequest

P10	P7	.	.	.	.				

PreOldRequest1
(String)

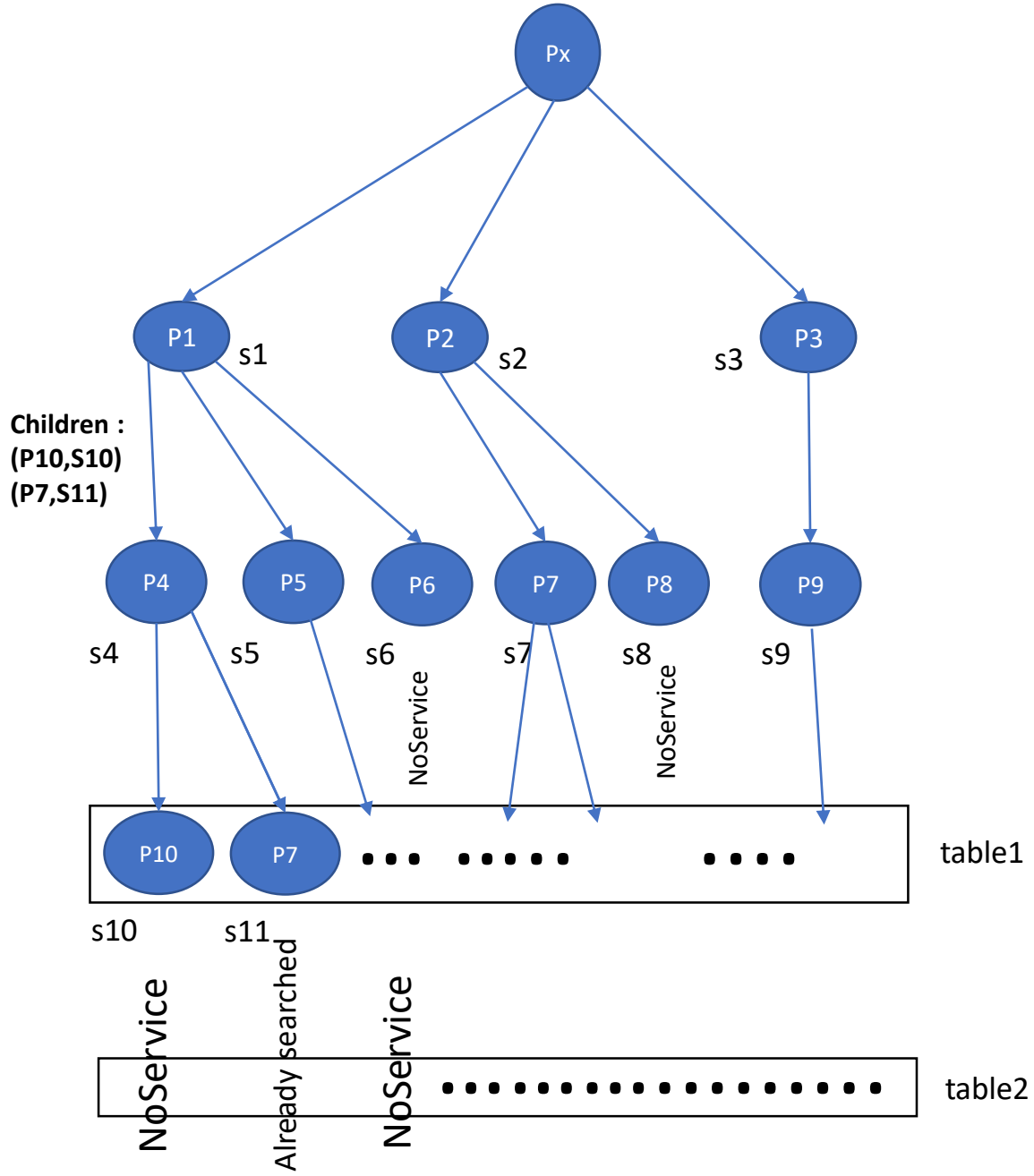
PreOldRequest2
(String)

P10	P7	.	.	.	.				

Table1 (child)

Table2 (child)

->Add PreOldRequest2 to OldRequest
->Empty PreOldRequest2
->Empty table2



Px	P1	P2	P3	P4	P5	P6	P7	P8	P9

OldRequest

P10	P7	.	.	.	.				

PreOldRequest1 (String)

PreOldRequest2 (String)

P10	P7	.	.	.	.				

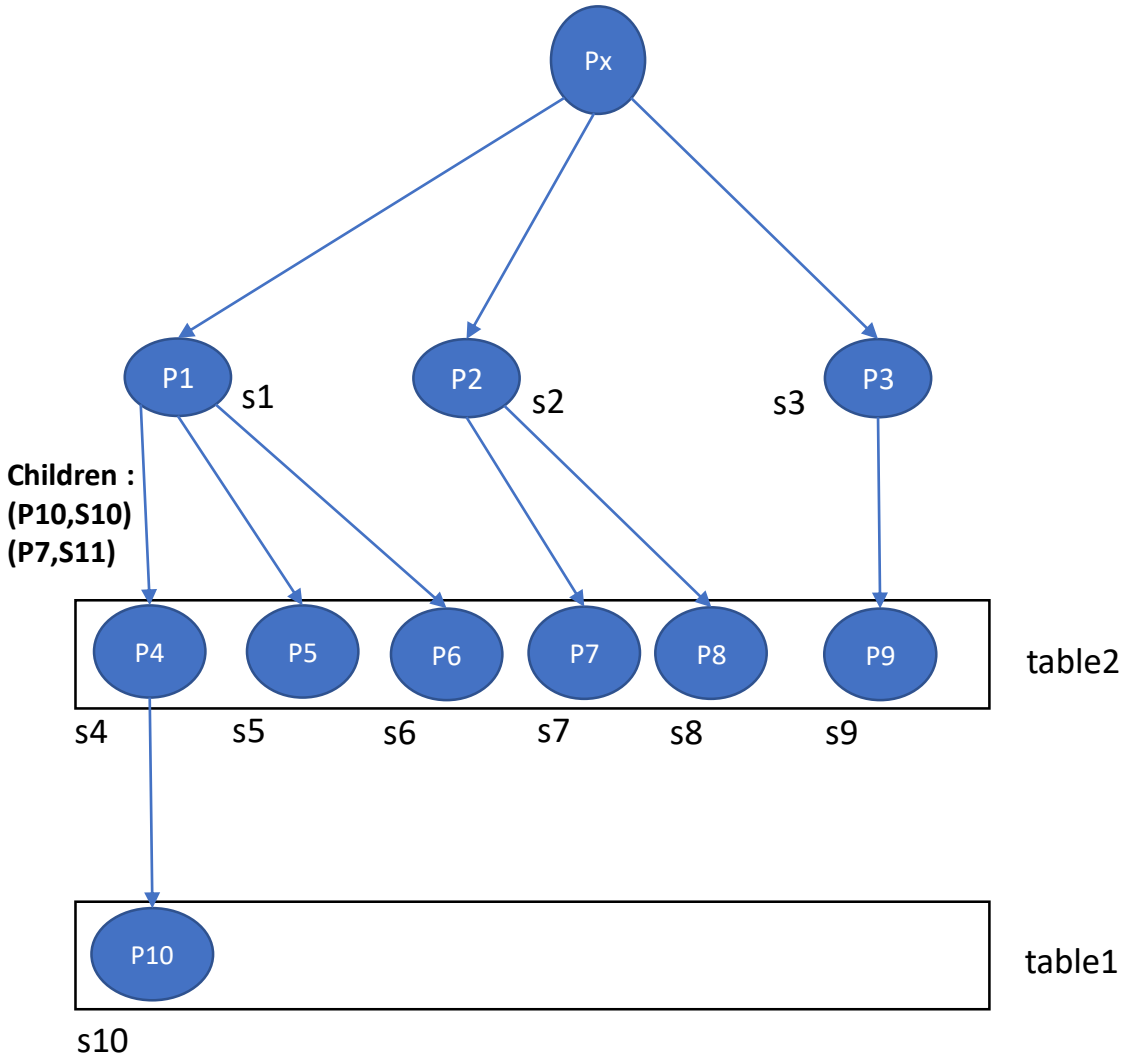
Table1 (child)

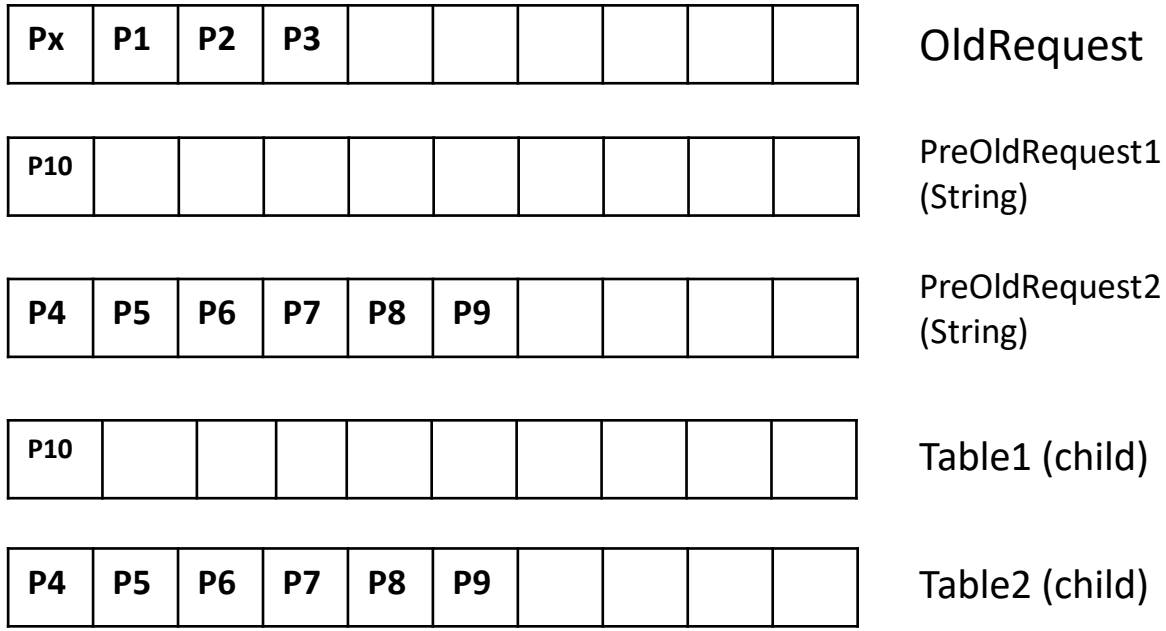
Table2 (child)

→ table2 is empty, means that we created the whole Tree without finding the destination parameter wanted.

-> RETURN NoComposition.

Case of finding the destination parameter wanted





When we find the parameter wanted (P10) :
→Get the name of the service from the current node
LOOP:
→Retrieve the node of the ancestor
→Get the name of the service in the ancestor's node
STOP THE ALGORITHM.