Specifying qlock lockfreeness and fairness

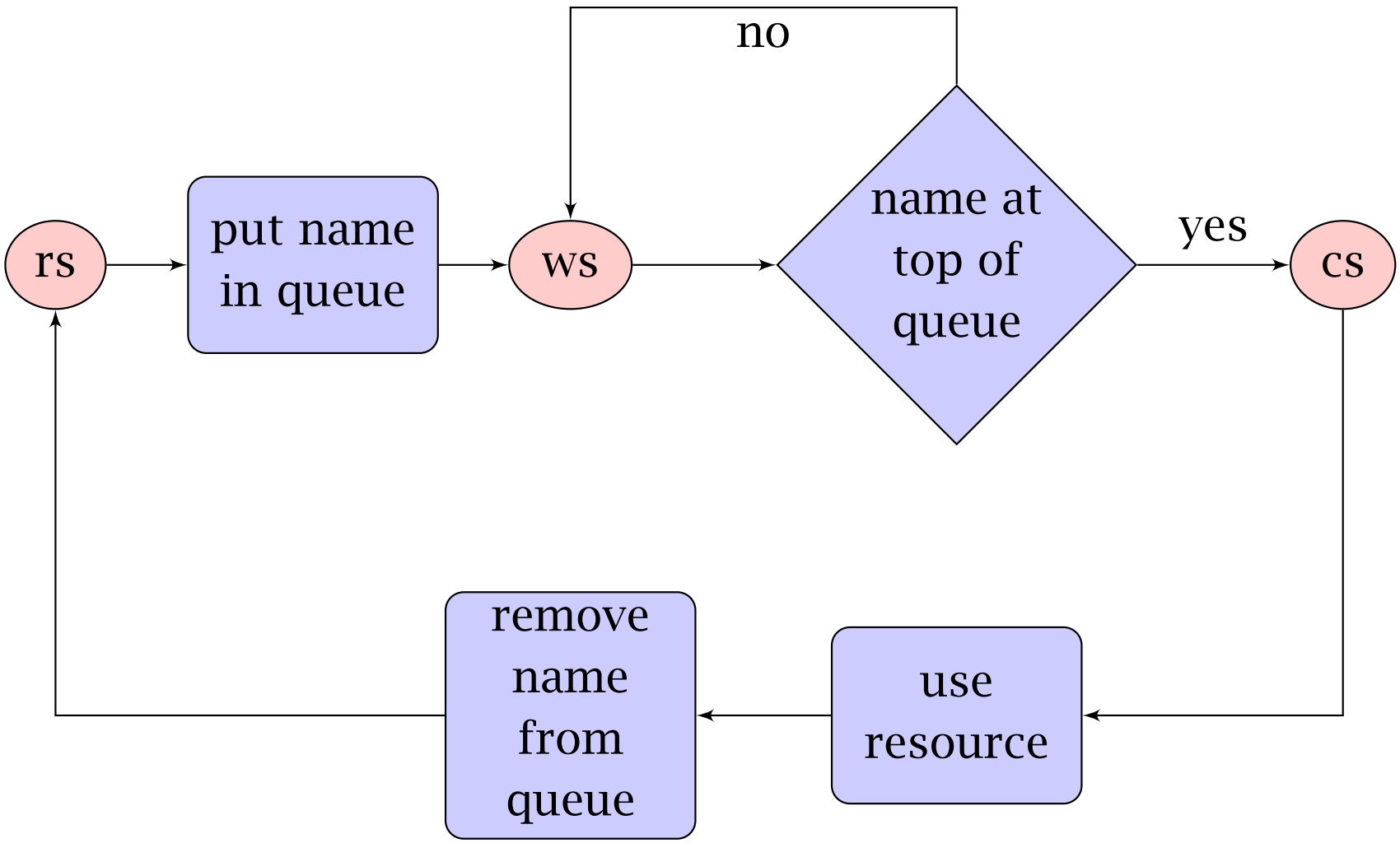
Norbert Preining

Lecture Note 14
Formal Method (i613)

Review: QLOCK

- arbitrary number of agents compete for a resource (e.g., keyboard)
- regulation mechanism to ensure exclusive usage of that resource
- operational description:
  - agents wanting to use the resource put their name into a queue (if it is not already in the queue)
  - if agent name is in the queue but not on top of the queue, wait
  - if agent name is at the top of the queue use the resource
  - if resource usage is finished remove name from the top of the queue

QLOCK FLOW

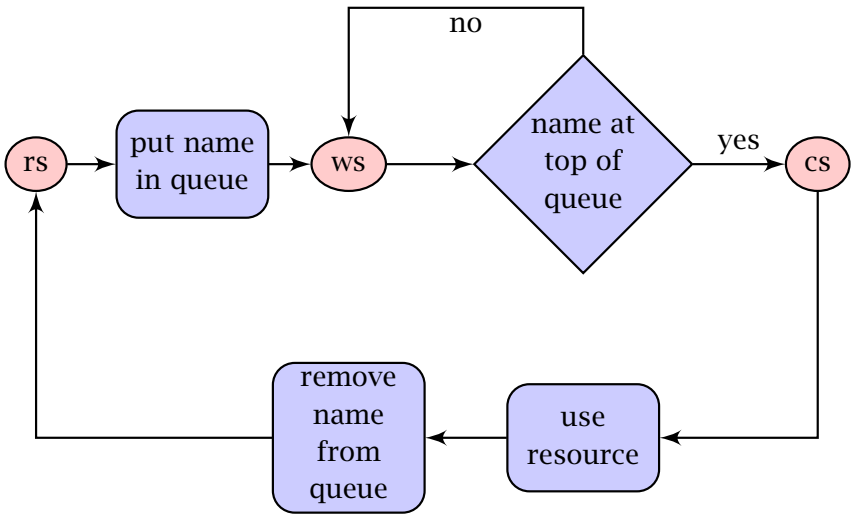

no
rs
put name in queue
ws
name at top of queue
yes
cs
remove name from queue
use resource

Qlock Example

Queue

Section

a	rs
b	rs
c	rs



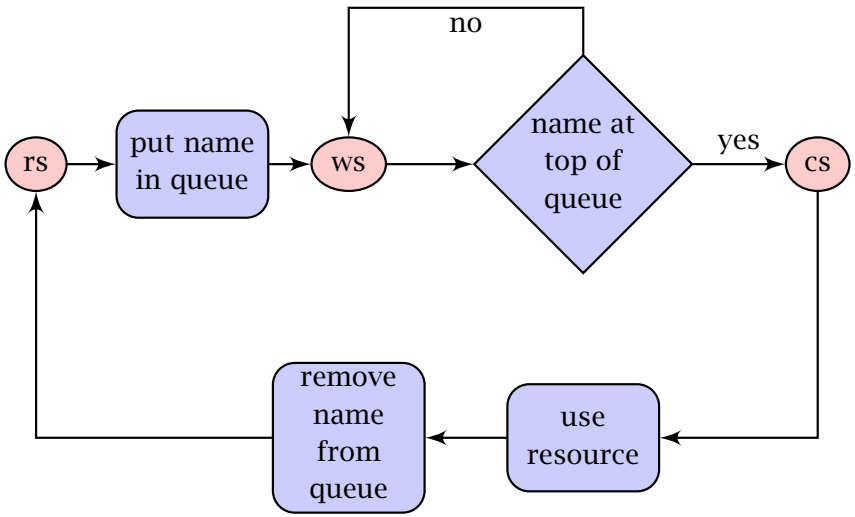
Qlock Example

Trans Queue Section

Enter

a

a	ws
b	rs
c	rs

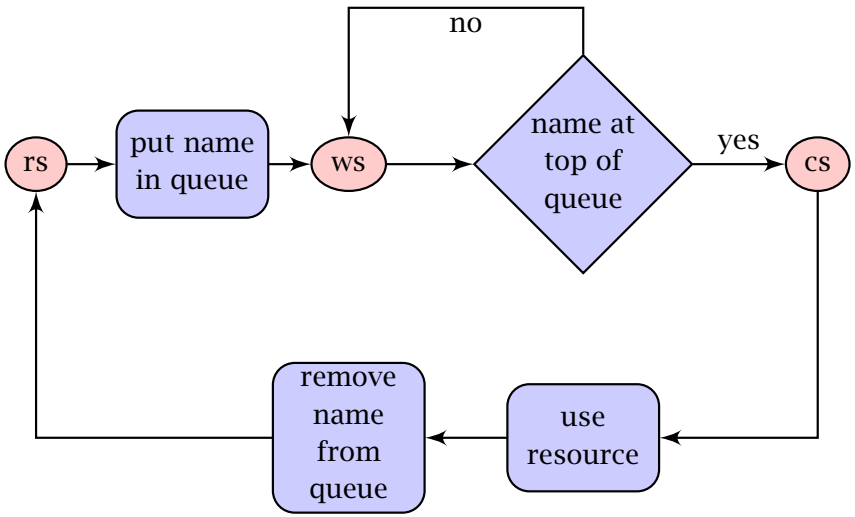


TransSeq

a

Qlock Example

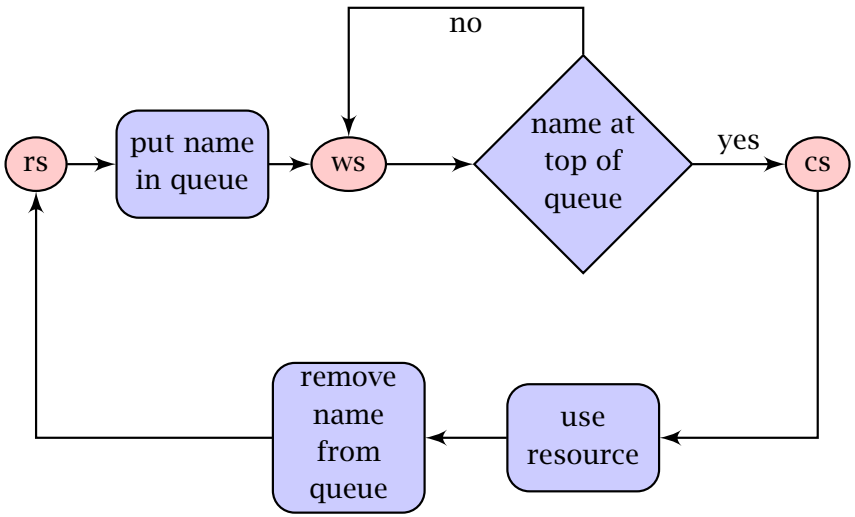
Trans	Queue	Section	
Enter	a	a	ws
	b	b	ws
		c	rs



TransSeq	a	b

Qlock Example

Trans	Queue	Section	
Enter	a	a	ws
	b	b	ws
	c	c	ws



TransSeq			
	a	b	c

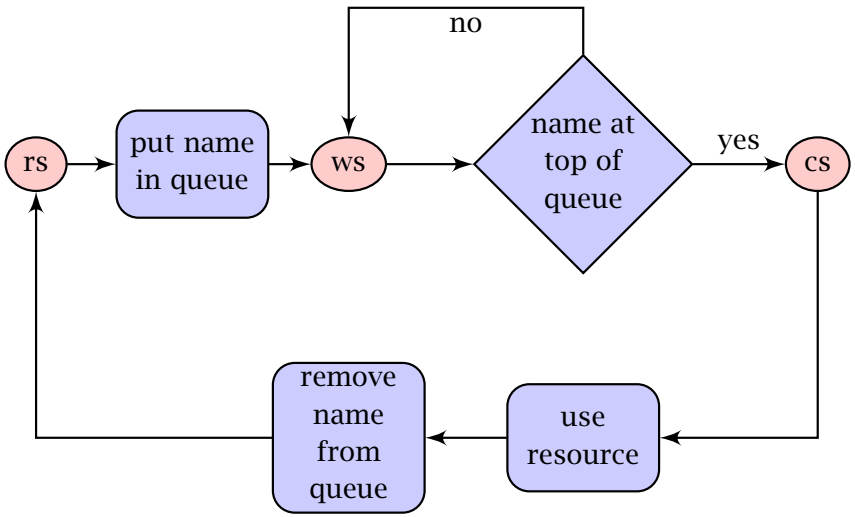
Qlock Example

Trans Queue Section

Try

a
b
c

a	cs
b	ws
c	ws



TransSeq

a	b	c	a

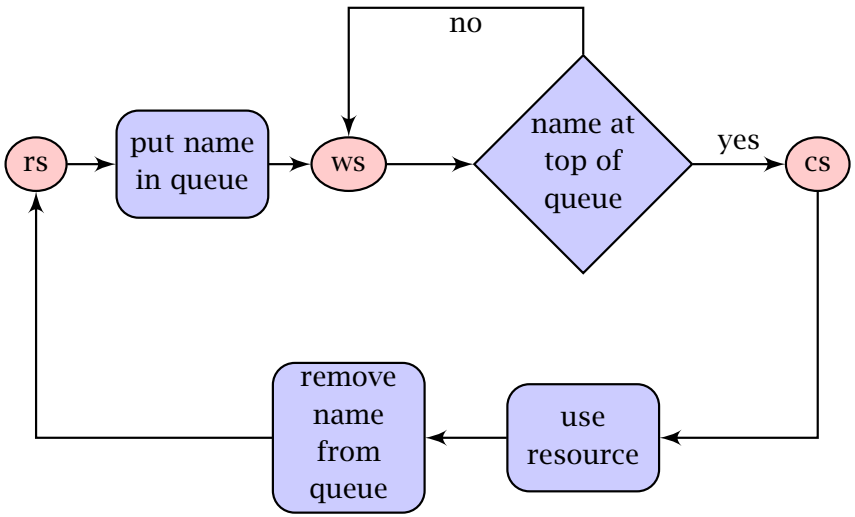
Qlock Example

Trans Queue Section

Try

a
b
c

a	cs
b	ws
c	ws



TransSeq

a	b	c	a	c

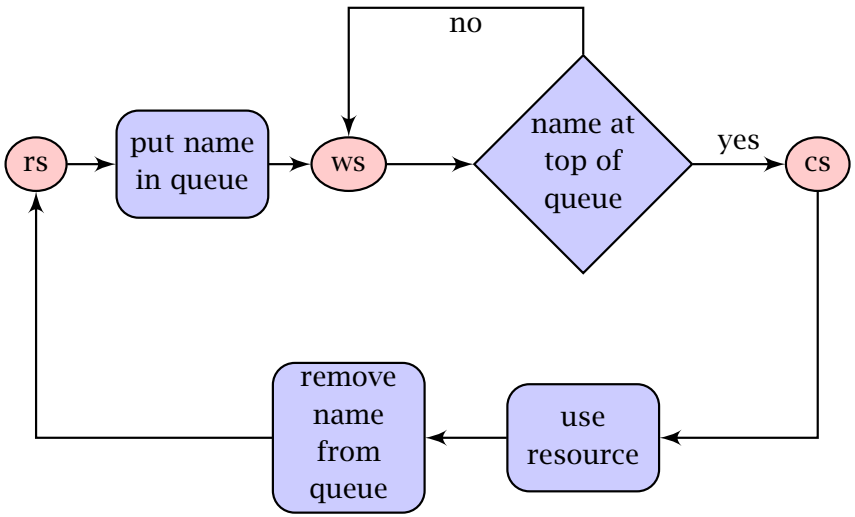
Qlock Example

Trans Queue Section

Try

a
b
c

a	cs
b	ws
c	ws



TransSeq

a	b	c	a	c	b

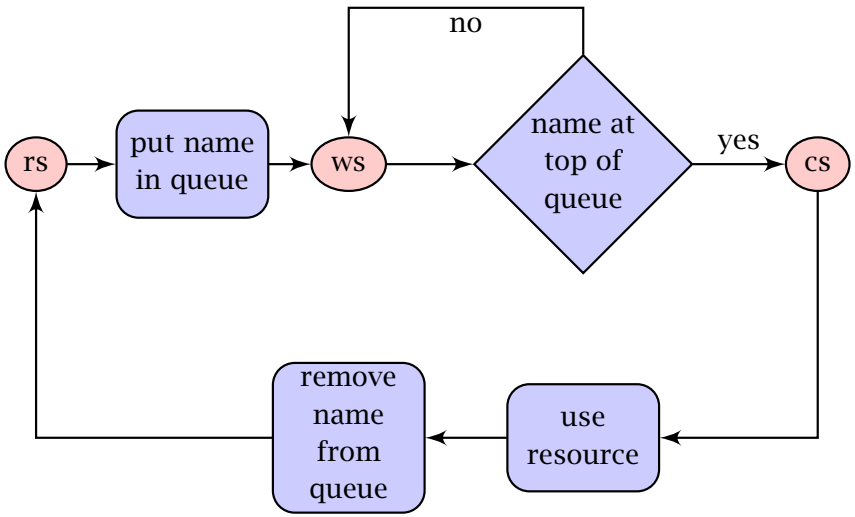
Qlock Example

Trans Queue Section

Exit

b
c

a	rs
b	ws
c	ws

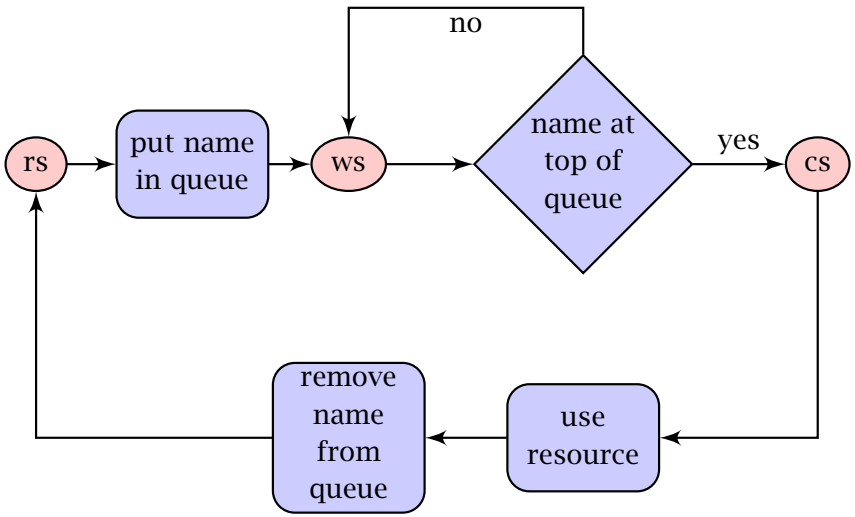


TransSeq

a	b	c	a	c	b	a

Qlock Example

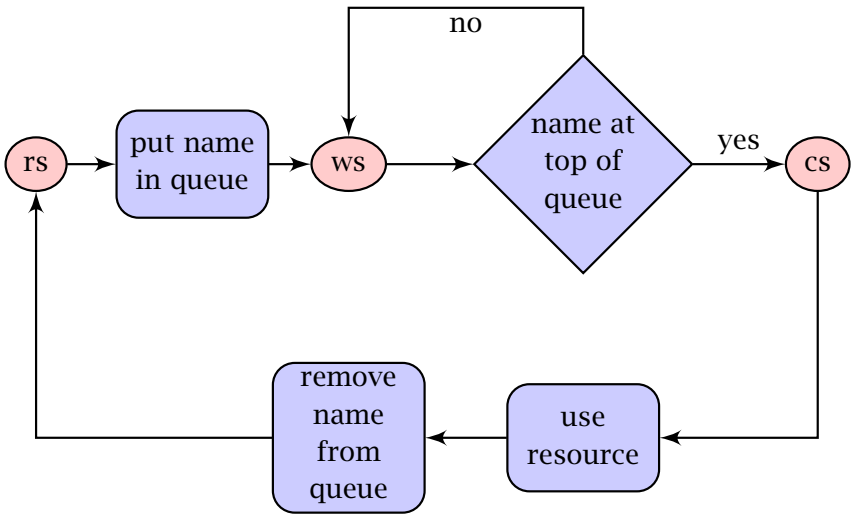
Trans	Queue	Section	
Enter	b	a	ws
	c	b	ws
	a	c	ws



TransSeq								
	a	b	c	a	c	b	a	a

Qlock Example

Trans	Queue	Section
Try	b c a	a ws b cs c ws



TransSeq									
	a	b	c	a	c	b	a	a	b

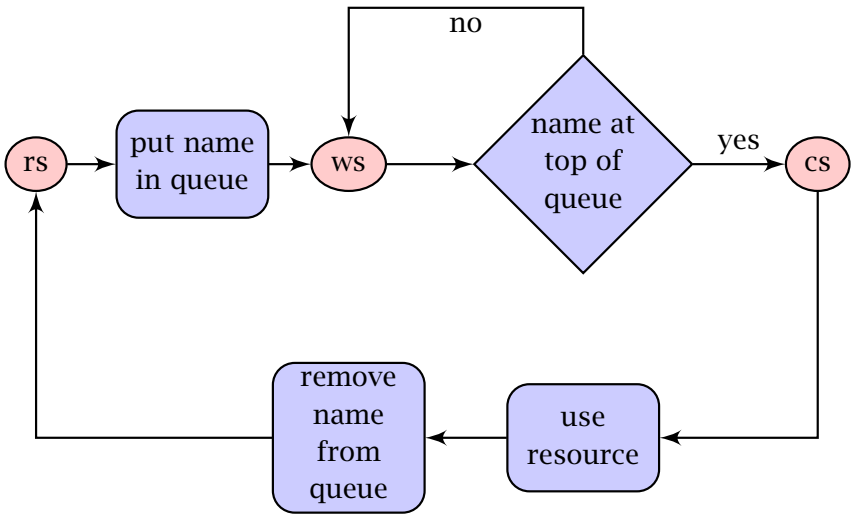
Qlock Example

Trans Queue Section

Try

b
c
a

a	ws
b	cs
c	ws



TransSeq

a	b	c	a	c	b	a	a	b	c

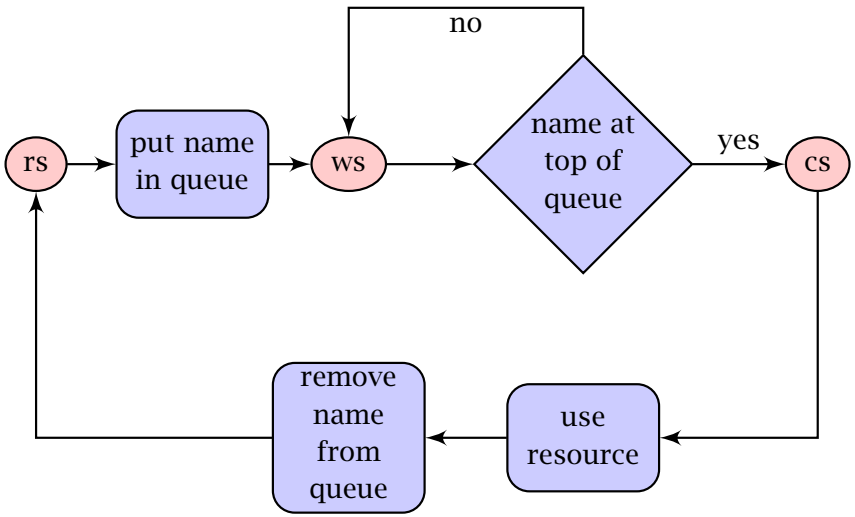
Qlock Example

Trans Queue Section

Exit

Queue
c
a

Trans	Section
a	ws
b	rs
c	ws



TransSeq

a	b	c	a	c	b	a	a	b	c	b

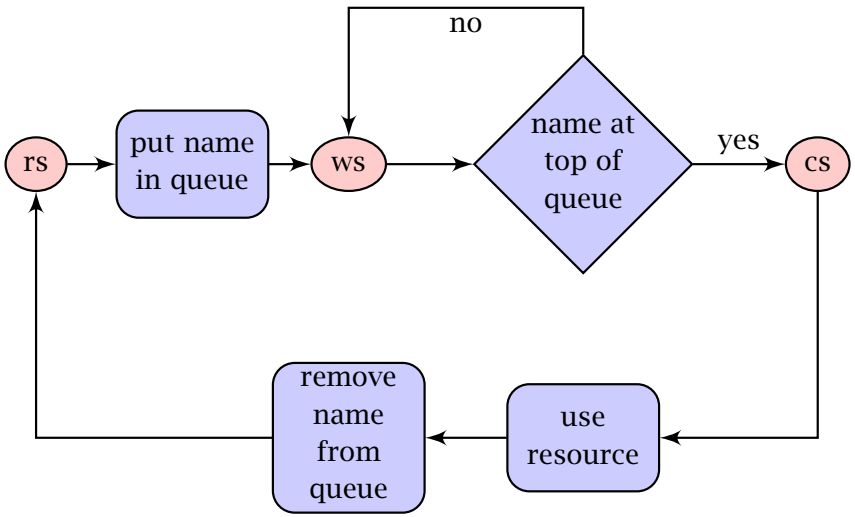
Qlock Example

Trans Queue Section

Try

c
a

a	ws
b	rs
c	cs



TransSeq

a	b	c	a	c	b	a	a	b	c	b	c

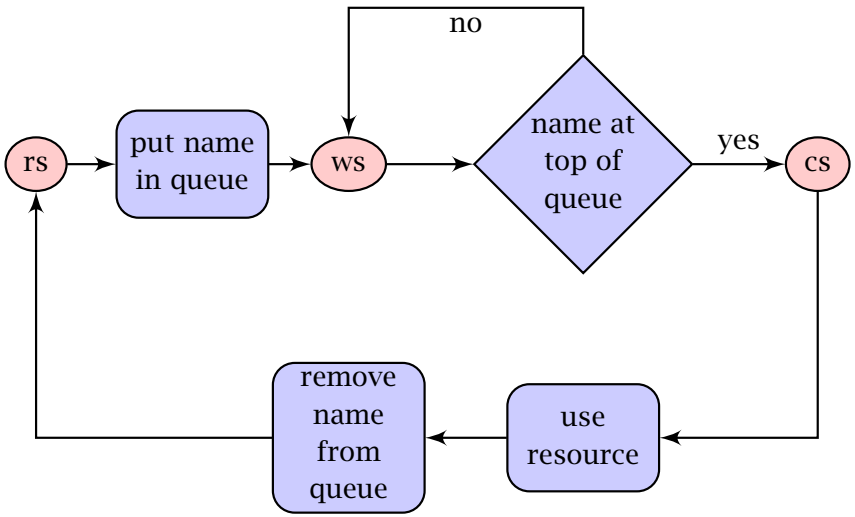
Qlock Example

Trans Queue Section

Exit

a

a	ws
b	rs
c	rs



TransSeq

a	b	c	a	c	b	a	a	b	c	b	c	c

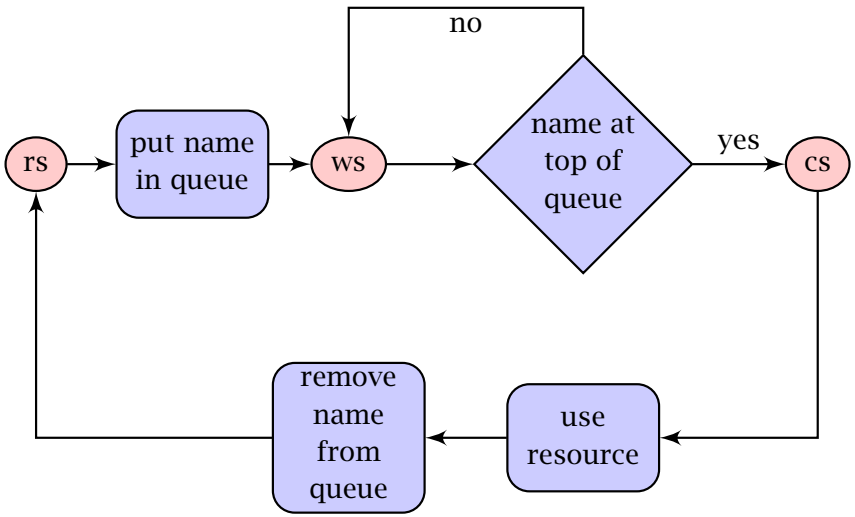
Qlock Example

Trans Queue Section

Try

Queue
a

Trans	Section
a	cs
b	rs
c	rs



TransSeq

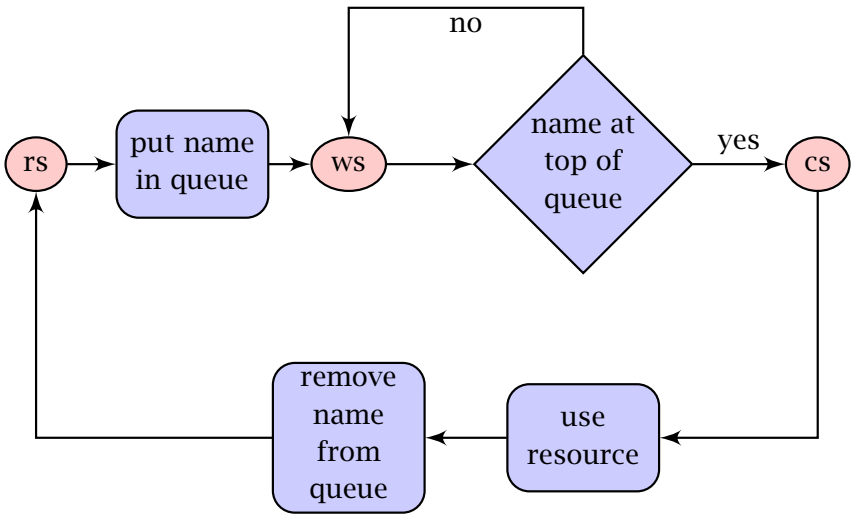
a	b	c	a	c	b	a	a	b	c	b	c	c	a

Qlock Example

Trans Queue Section

Exit

a	rs
b	rs
c	rs



TransSeq

a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

Review II: properties shown

exclusion property only one agent is in critical state (Lecture 8)

lockfree I if agent enters queue, it reaches critical section (Lecture 9)

Review II: properties shown

exclusion property only one agent is in critical state (Lecture 8)

lockfree I if agent enters queue, it reaches critical section (Lecture 9)

What is missing?

Review II: properties shown

exclusion property only one agent is in critical state (Lecture 8)

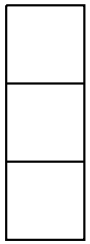
lockfree I if agent enters queue, it reaches critical section (Lecture 9)

What is missing?

Assume Norbert (a) going to the ice cream parlor (resource) ...

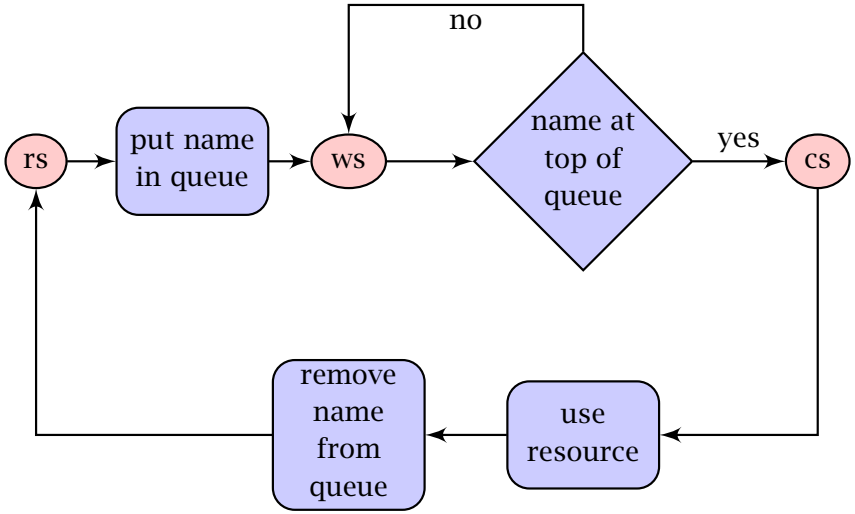
Another lockfree condition

Queue



Section

a	rs
b	rs
c	rs



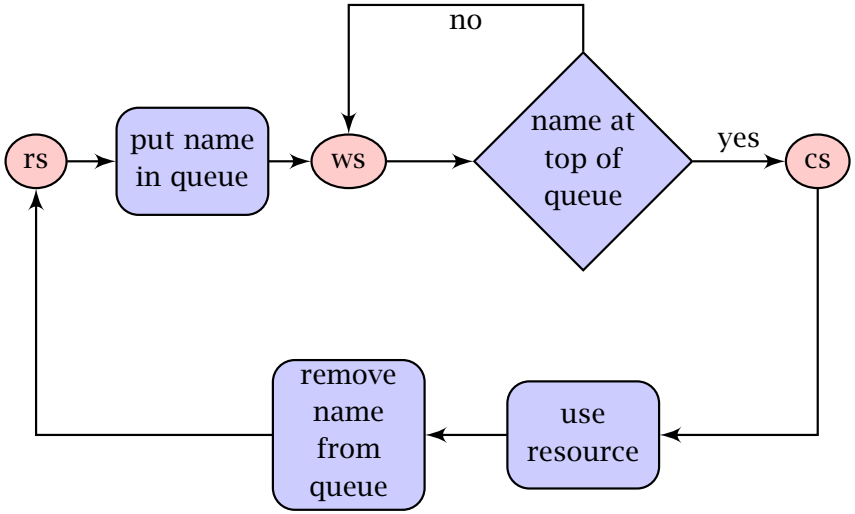
Another lockfree condition

Trans Queue Section

Enter

a

a	ws
b	rs
c	rs



TransSeq

a

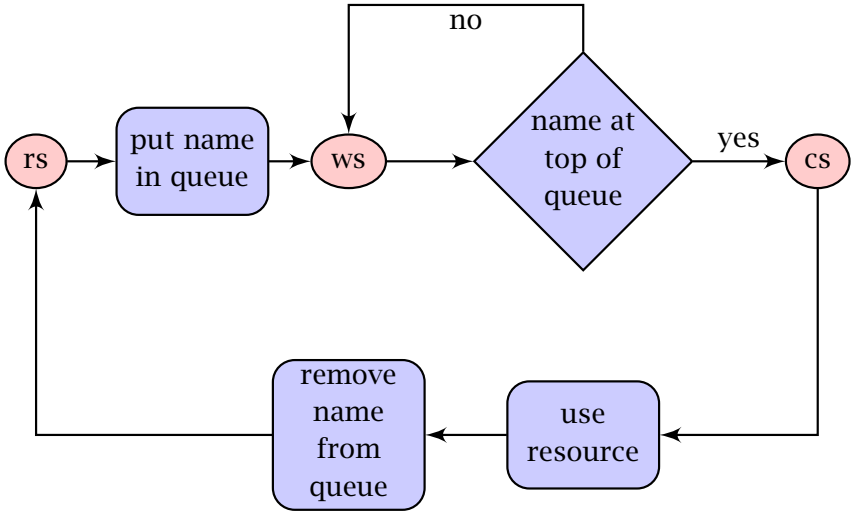
Another lockfree condition

Trans Queue Section

Try

a

a	cs
b	rs
c	rs



TransSeq

a	a

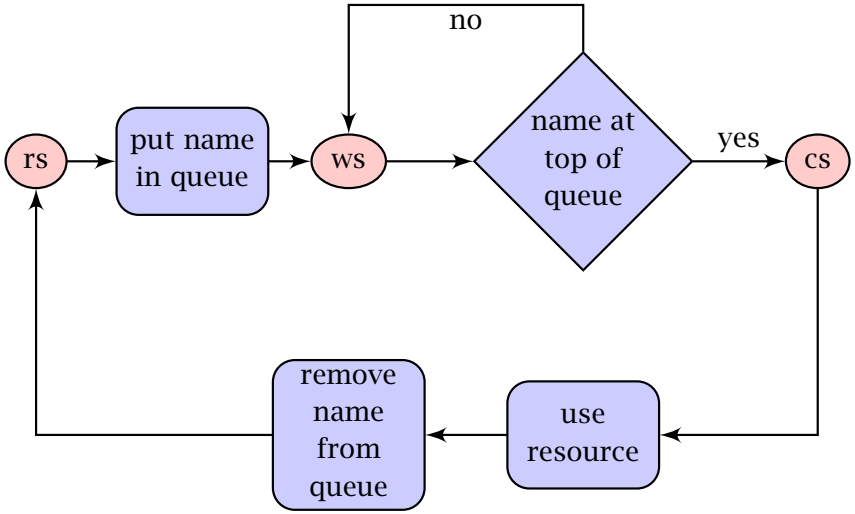
Another lockfree condition

Trans Queue Section

Exit

a

a	rs
b	rs
c	rs



TransSeq

a	a	a

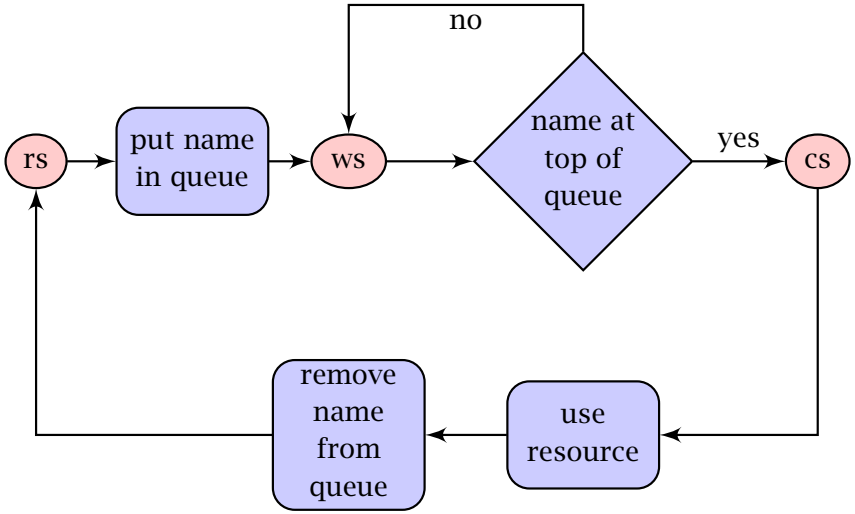
Another lockfree condition

Trans Queue Section

Enter

a

a	ws
b	rs
c	rs



TransSeq

a	a	a	a

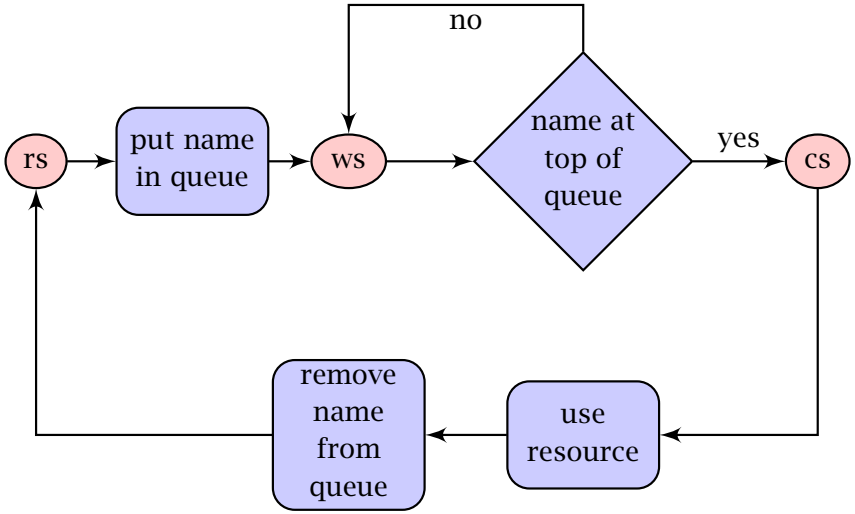
Another lockfree condition

Trans Queue Section

Try

a

a	cs
b	rs
c	rs



TransSeq

a	a	a	a	a

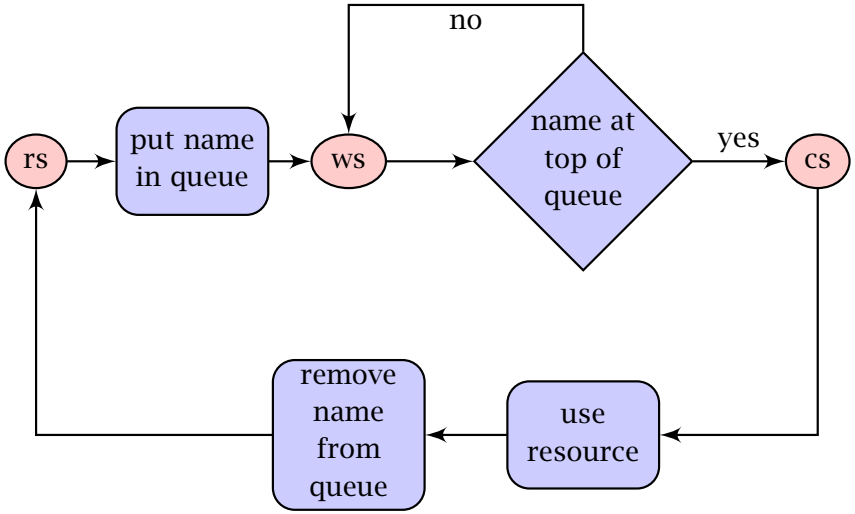
Another lockfree condition

Trans Queue Section

Exit

a

a	rs
b	rs
c	rs



TransSeq

a	a	a	a	a	a

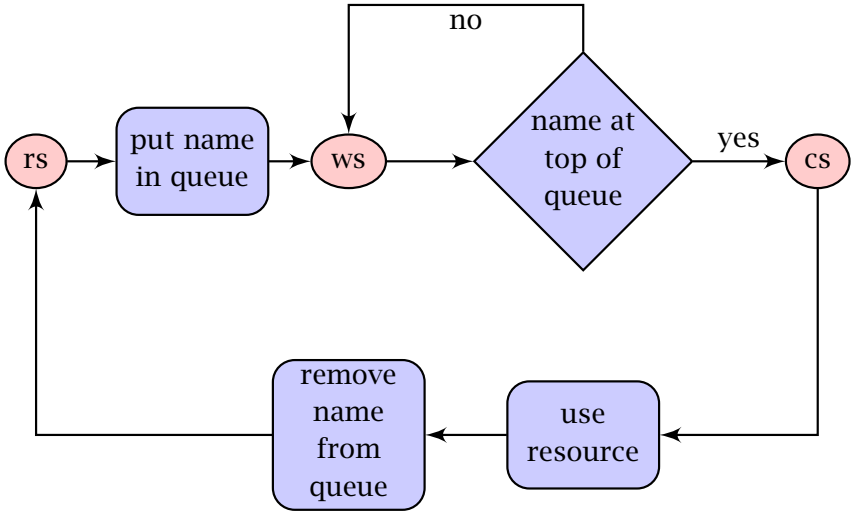
Another lockfree condition

Trans Queue Section

Enter

a

a	ws
b	rs
c	rs



TransSeq

a	a	a	a	a	a	a

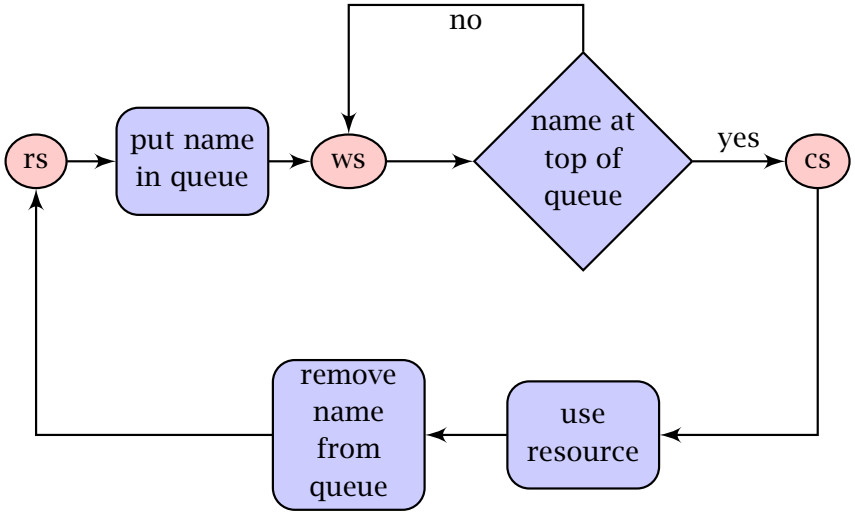
Another lockfree condition

Trans Queue Section

Try

a

a	cs
b	rs
c	rs



TransSeq

a	a	a	a	a	a	a	a

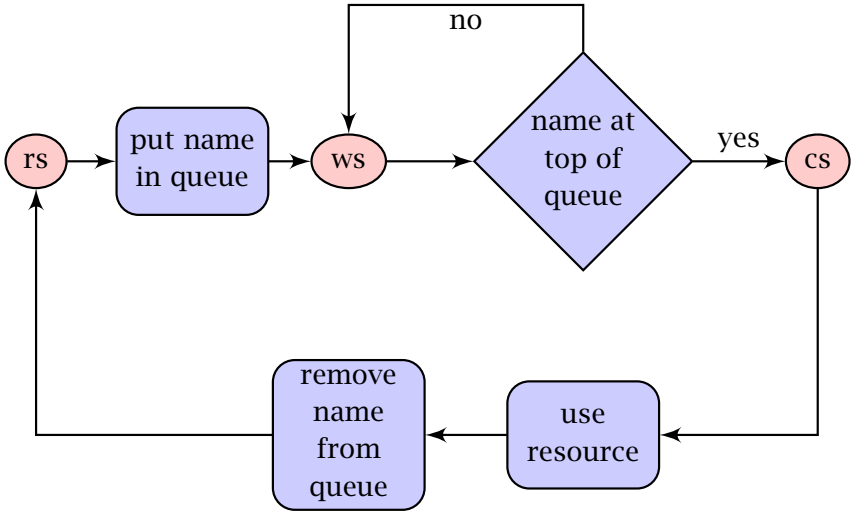
Another lockfree condition

Trans Queue Section

Exit

a

a	rs
b	rs
c	rs



TransSeq

a	a	a	a	a	a	a	a	a

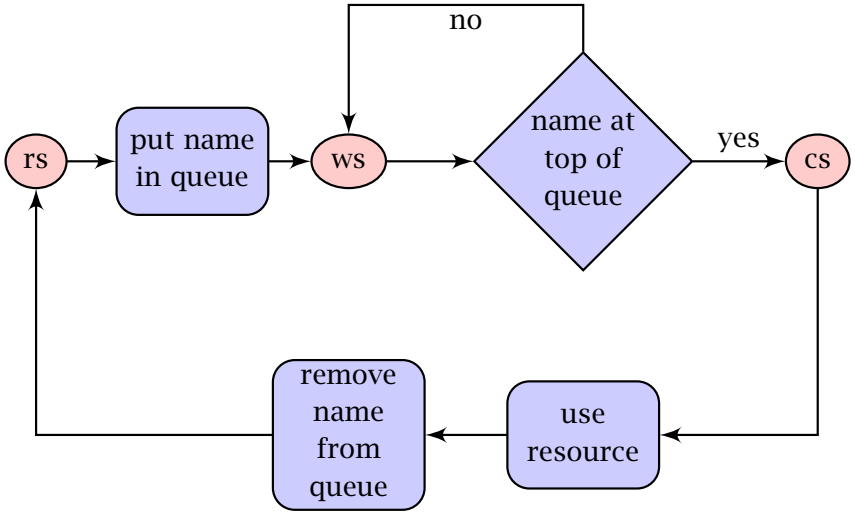
Another lockfree condition

Trans Queue Section

Enter

a

a	ws
b	rs
c	rs



TransSeq

a	a	a	a	a	a	a	a	a	a

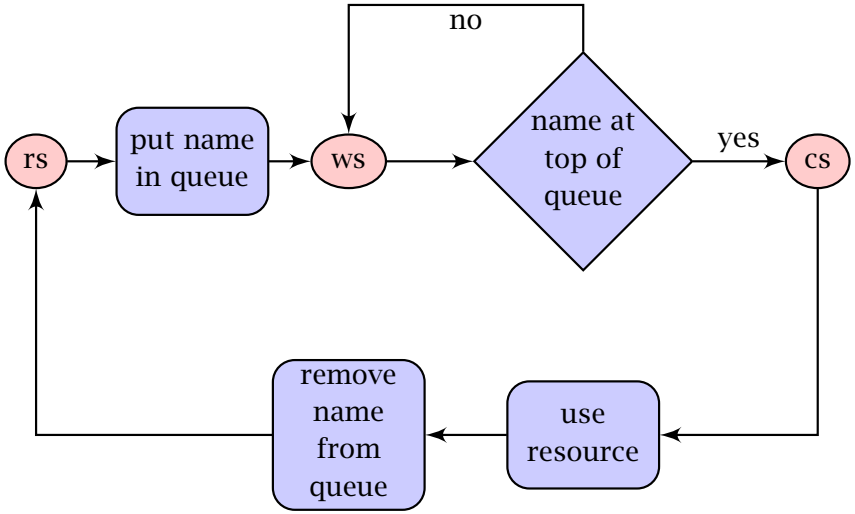
Another lockfree condition

Trans Queue Section

Try

a

a	cs
b	rs
c	rs



TransSeq

a	a	a	a	a	a	a	a	a	a	a

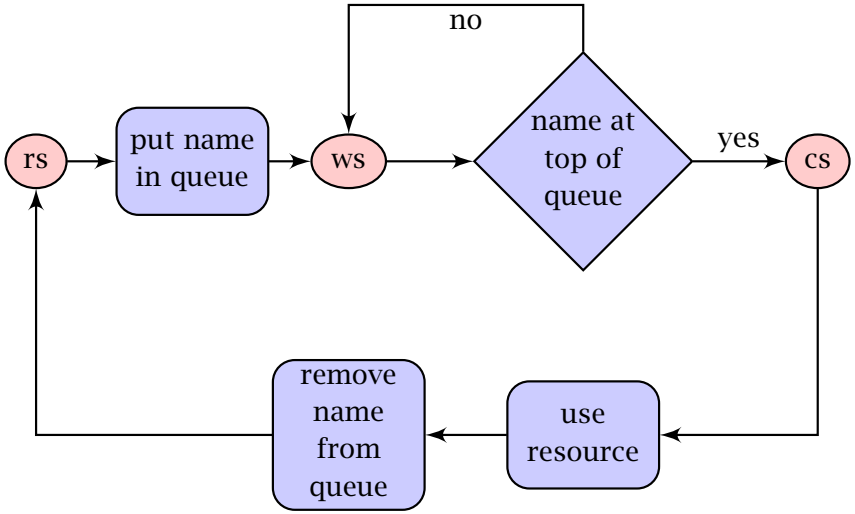
Another lockfree condition

Trans Queue Section

Exit

a

a	rs
b	rs
c	rs

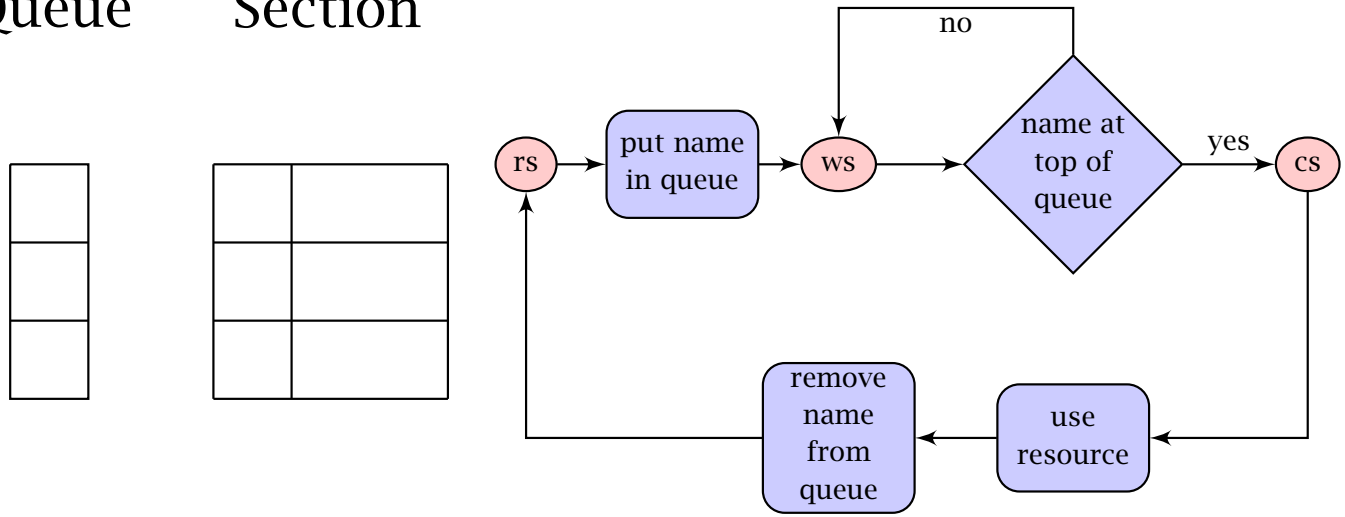


TransSeq

a	a	a	a	a	a	a	a	a	a	a	a

Another lockfree condition

Trans Queue Section



TransSeq

a	a	a	a	a	a	a	a	a	a	a	a

I think you got the idea …

Fairness

a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

$\cdots$

Fairness condition

Every agent id appears infinitely many times in the transition sequence.

Fairness

a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

$\cdots$

Fairness condition

Every agent id appears infinitely many times in the transition sequence.

Equivalent

Every arbitrary given sequence of agent ids is a subsequence of the transition sequence.

(proof straight-forward)

Proving fairness

Central idea

Proving fairness requires a metalevel argument – fairness of transition sequence.

Include the transition sequence in the modelling.

Model a *meta-state* by combining the normal state with a transition sequence

Proving fairness

Central idea

Proving fairness requires a metalevel argument – fairness of transition sequence.

Include the transition sequence in the modelling.

Model a *meta-state* by combining the normal state with a transition sequence

Deeper understanding . . .

We need to actually know when a section change happens, i.e., be able to compute the waiting time for each agent till the next section change.

Another look onto qlock

Queue

Section

a	rs
b	rs
c	rs

TransSeq

a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

Another look onto qlock

Trans	Queue	Section
Enter	a	a ws
		b rs
		c rs

↓

TransSeq														
a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

Another look onto qlock

Trans	Queue	Section	
Enter	a	a	ws
	b	b	ws
		c	rs

TransSeq

~~a~~	b	c	a	c	b	a	a	b	c	b	c	c	a	a

Another look onto qlock

Trans	Queue	Section	
Enter	a	a	ws
	b	b	ws
	c	c	ws

TransSeq														
	~~a~~	~~b~~	↓ c	a	c	b	a	a	b	c	b	c	c	a	a

Another look onto qlock

Trans	Queue	Section
Try	a b c	a cs b ws c ws

TransSeq

~~a~~	~~b~~	~~c~~	a	c	b	a	a	b	c	b	c	c	a	a

Another look onto qlock

Trans	Queue	Section
Try	a b c	a cs b ws c ws

↓

TransSeq														
	~~a~~	~~b~~	~~c~~	~~a~~	c	b	a	a	b	c	b	c	c	a	a

Another look onto qlock

Trans	Queue	Section
Try	a b c	a cs b ws c ws

↓

TransSeq														
	~~a~~	~~b~~	~~c~~	~~a~~	~~c~~	b	a	a	b	c	b	c	c	a	a

Another look onto qlock

Trans	Queue	Section	
Exit	b	a	rs
	c	b	ws
		c	ws

↓

TransSeq														
	~~a~~	~~b~~	~~c~~	~~a~~	~~c~~	~~b~~	a	a	b	c	b	c	c	a	a

Another look onto qlock

Trans	Queue	Section	
Enter	b	a	ws
	c	b	ws
	a	c	ws

TransSeq

~~a~~	~~b~~	~~c~~	~~a~~	~~c~~	~~b~~	~~a~~	a	b	c	b	c	c	a	a

Another look onto qlock

Trans	Queue	Section	
Try	b	a	ws
	c	b	cs
	a	c	ws

TransSeq

~~a~~	~~b~~	~~c~~	~~a~~	~~c~~	~~b~~	~~a~~	~~a~~	b	c	b	c	c	a	a

Another look onto qlock

Trans	Queue	Section	
Try	b	a	ws
	c	b	cs
	a	c	ws

TransSeq

~~a~~	~~b~~	~~c~~	~~a~~	~~c~~	~~b~~	~~a~~	~~a~~	~~b~~	c	b	c	c	a	a

Another look onto qlock

Trans	Queue	Section	
Exit	c	a	ws
	a	b	rs
		c	ws

TransSeq

~~a~~	~~b~~	~~c~~	~~a~~	~~c~~	~~b~~	~~a~~	~~a~~	~~b~~	~~c~~	b	c	c	a	a

Another look onto qlock

Trans	Queue	Section	
Try	c	a	ws
	a	b	rs
		c	cs

TransSeq

~~a~~	~~b~~	~~c~~	~~a~~	~~c~~	~~b~~	~~a~~	~~a~~	~~b~~	~~c~~	~~b~~	c	c	a	a

Another look onto qlock

Trans	Queue	Section	
Exit	a	a	ws
		b	rs
		c	rs

TransSeq

a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

Another look onto qlock

Trans	Queue	Section	
Try	a	a	cs
		b	rs
		c	rs

TransSeq

a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

Another look onto qlock

Trans	Queue	Section	
Exit		a	rs
		b	rs
		c	rs

TransSeq

a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

Waiting times

How many steps in the transition sequence an agent has to wait until it can transition into a different section?

rs-ws no further condition, thus next occurrence of the agent id in the TransSeq

Waiting times

How many steps in the transition sequence an agent has to wait until it can transition into a different section?

- rs-ws: no further condition, thus next occurrence of the agent id in the TransSeq
- ws-cs: see later

Waiting times

How many steps in the transition sequence an agent has to wait until it can transition into a different section?

- rs-ws: no further condition, thus next occurrence of the agent id in the TransSeq
- ws-cs: see later
- cs-rs: no further condition, thus next occurrence of the agent id in the TransSeq

Waiting time in waiting section ws

Queue

Section

a	rs
b	rs
c	rs

TransSeq

a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

Waiting time in waiting section ws

Trans	Queue	Section	
Enter	a	a	ws
		b	rs
		c	rs

↓

TransSeq														
a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

Waiting time in waiting section ws

Trans	Queue	Section	
Enter	a	a	ws
	b	b	ws
		c	rs

↓

TransSeq														
	~~a~~	b	c	a	c	b	a	a	b	c	b	c	c	a	a

Waiting time in waiting section ws

Trans	Queue	Section	
Enter	a	a	ws
	b	b	ws
	c	c	ws

↓

TransSeq														
~~a~~	~~b~~	c	a	c	b	a	a	b	c	b	c	c	a	a

When can c enter into the critical section?

Waiting time in waiting section ws

Trans	Queue	Section	
Enter	a	a	ws
	b	b	ws
	c	c	ws

TransSeq														
~~a~~	~~b~~	c	a	c	b	a	a	b	c	b	c	c	a	a

When can c enter into the critical section?
First a has to go into cs

Waiting time in waiting section ws

Trans	Queue	Section	
Enter	a	a	ws
	b	b	ws
	c	c	ws

TransSeq														
~~a~~	~~b~~	c	a	c	b	a	a	b	c	b	c	c	a	a

When can c enter into the critical section?
Then a has to go into rs

Waiting time in waiting section ws

Trans	Queue	Section	
Enter	a	a	ws
	b	b	ws
	c	c	ws

TransSeq														
~~a~~	~~b~~	c	a	c	b	a	a	b	c	b	c	c	a	a

When can c enter into the critical section?
Then b has to go into cs

Waiting time in waiting section ws

Trans	Queue	Section	
Enter	a	a	ws
	b	b	ws
	c	c	ws

TransSeq														
~~a~~	~~b~~	c	a	c	b	a	a	b	c	b	c	c	a	a

When can c enter into the critical section?
Then b has to go into rs

Waiting time in waiting section ws

Trans	Queue	Section	
Enter	a	a	ws
	b	b	ws
	c	c	ws

TransSeq														
~~a~~	~~b~~	c	a	c	b	a	a	b	c	b	c	c	a	a
			●			●		●		●	●			

When can c enter into the critical section?

Then on the next c can go into cs — subsequence: a a b b c

Waiting time in waiting section ws

Trans	Queue	Section	
Try	a	a	cs
	b	b	ws
	c	c	ws

TransSeq

~~a~~	~~b~~	~~c~~	a	c	b	a	a	b	c	b	c	c	a	a

When can c enter into the critical section?

Then on the next c can go into cs — subsequence: a a b b c

Waiting time in waiting section ws

Trans	Queue	Section	
Try	a	a	cs
	b	b	ws
	c	c	ws

TransSeq

~~a~~	~~b~~	~~c~~	~~a~~	c	b	a	a	b	c	b	c	c	a	a

When can c enter into the critical section?

Then on the next c can go into cs — subsequence: a a b b c

Waiting time in waiting section ws

Trans	Queue	Section	
Try	a	a	cs
	b	b	ws
	c	c	ws

TransSeq

~~a~~	~~b~~	~~c~~	~~a~~	~~c~~	b	a	a	b	c	b	c	c	a	a

When can c enter into the critical section?

Then on the next c can go into cs — subsequence: a a b b c

Waiting time in waiting section ws

Trans	Queue	Section	
Exit	b	a	rs
	c	b	ws
		c	ws

TransSeq

~~a~~	~~b~~	~~c~~	~~a~~	~~c~~	~~b~~	a	a	b	c	b	c	c	a	a

When can c enter into the critical section?

Then on the next c can go into cs — subsequence: a a b b c

Waiting time in waiting section ws

Trans	Queue	Section	
Enter	b	a	ws
	c	b	ws
	a	c	ws

TransSeq

a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

When can c enter into the critical section?

Then on the next c can go into cs — subsequence: a a b b c

Waiting time in waiting section ws

Trans	Queue	Section	
Try	b	a	ws
	c	b	cs
	a	c	ws

TransSeq

a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

When can c enter into the critical section?

Then on the next c can go into cs — subsequence: a a b b c

Waiting time in waiting section ws

Trans	Queue	Section	
Try	b	a	ws
	c	b	cs
	a	c	ws

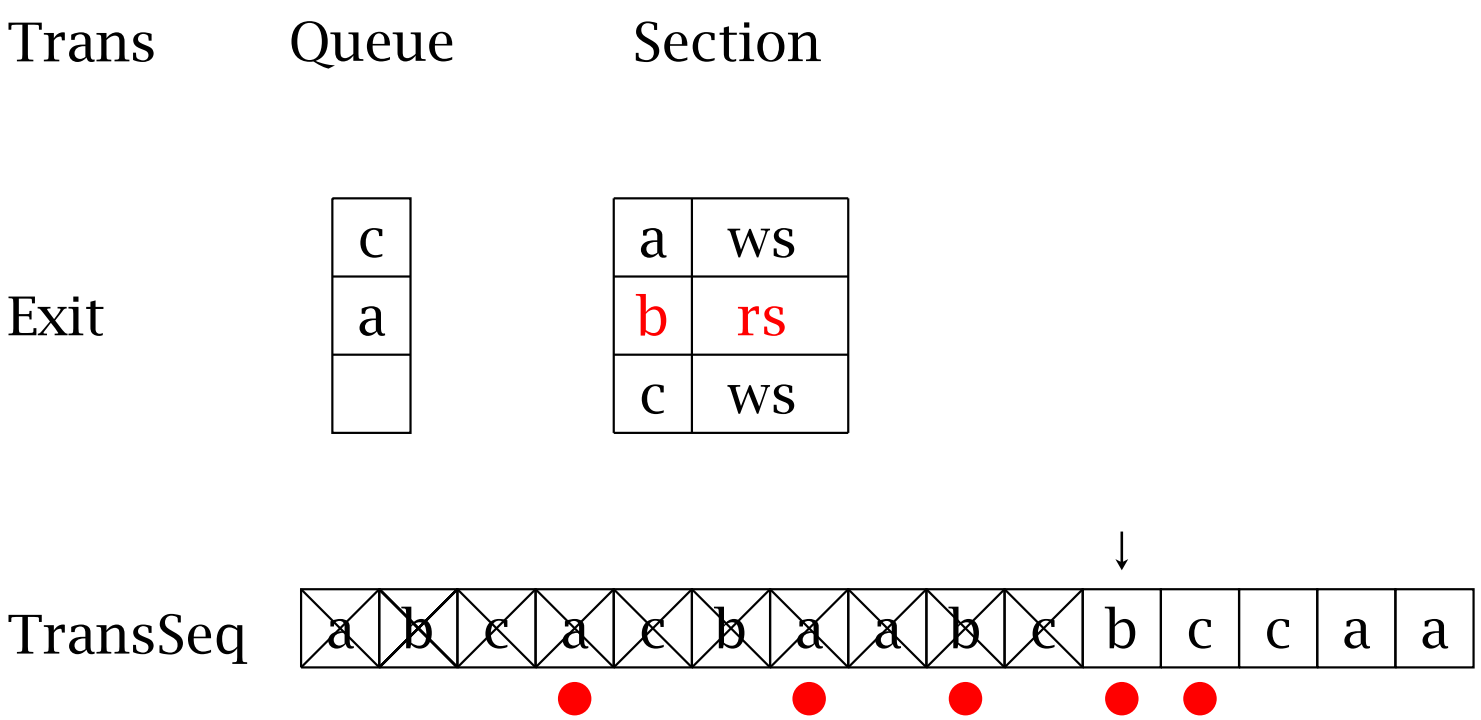
TransSeq

a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

When can c enter into the critical section?

Then on the next c can go into cs — subsequence: a a b b c

Waiting time in waiting section ws



When can c enter into the critical section?

Then on the next c can go into cs — subsequence: a a b b c

Waiting time in waiting section ws

Trans	Queue	Section	
Try	c	a	ws
	a	b	rs
		c	cs

TransSeq

a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

When can c enter into the critical section?

Then on the next c can go into cs — subsequence: a a b b c

Waiting time in waiting section ws

Trans Queue Section

Exit

a

a	ws
b	rs
c	rs

TransSeq

a	b	c	a	c	b	a	a	b	c	b	c	c	a	a

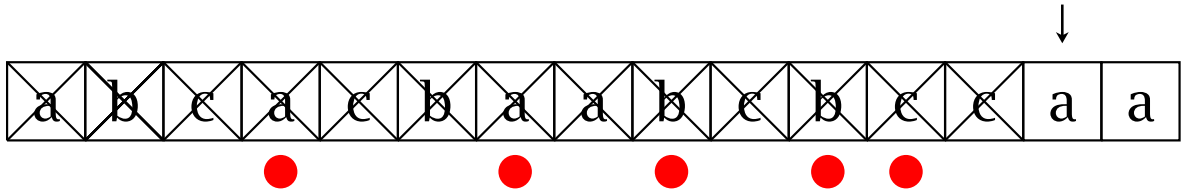
When can c enter into the critical section?

Then on the next c can go into cs — subsequence: a a b b c

Waiting time in waiting section ws

Trans	Queue	Section	
Try	a	a	cs
		b	rs
		c	rs

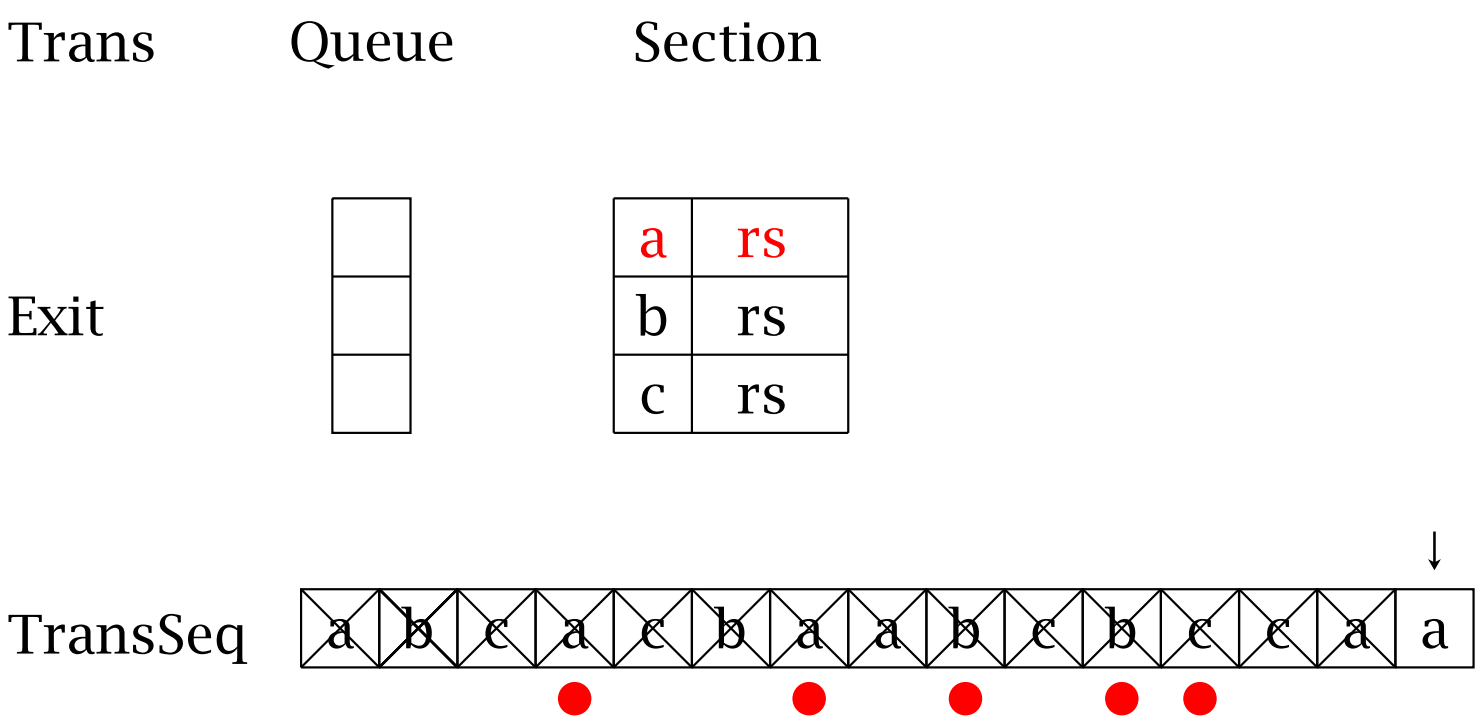
TransSeq



When can c enter into the critical section?

Then on the next c can go into cs — subsequence: a a b b c

Waiting time in waiting section ws



When can c enter into the critical section?

Then on the next c can go into cs — subsequence: a a b b c

Waiting time for ws-cs

Consequence for waiting time to transition form ws to cs:

- for all the agents higher up in the queue then the current agent, this agent has to appear two times in the transition sequence:
  - once for entering the critical section
  - once for leaving the critical section
- after all these agents has happened, the current agent can transition into the critical section

Implementation

Definition of fairness via definedness

- extended AID-QUEUE with an error type
- add a find operator that searches a finite sequence within another sequence and returns the smallest prefix sequence containing the searched sub-sequence, or error type

Definition of fairness via definedness

- extended AID-QUEUE with an error type
- add a find operator that searches a finite sequence within another sequence and returns the smallest prefix sequence containing the searched sub-sequence, or error type
- assuming that this function is always defined is equivalent to the assumption of fairness

Implementation ideas

- keep original code `qlock-sysProp.cafe`, `qlock-invPS.cafe`
- reuse most modules, functions, invariants
- extended some modules to carry error handling
- define a ‘meta-state’ consisting

Extension of PNAT

```
mod! PNATERR {
  pr(PNAT)
  [ Nat Err < Nat&Err ]
  op err : -> Err {constr} .
  op s_ : Nat&Err -> Nat&Err {constr} .
  -- disjointness
  eq (N:Nat = err) = false .
  eq s(err) = err .
  op _<_ : Nat&Err Nat&Err -> Bool
  eq X:Nat&Err < err = false .
  eq err < X:Nat&Err = false .
}
```

Extension of AID-QUEUE

```
mod! AIDQERR {
  pr(AID-QUEUE)
  [ Aid ErrQ Qu < Qu&Err ]
  op errQ : -> ErrQ {constr} .
  eq (Q:Qu = errQ) = false .
  op (_&_) : Qu&Err Qu&Err -> Qu&Err {constr assoc↲
       ↳ id: empQ} .
  eq (Q:Qu&Err & errQ) = errQ .
  eq (errQ & Q:Qu&Err) = errQ .
}
```

METASTATES

Models the state of the qlock-system including the future execution order as another queue of agent ids.

```
mod! METASTATE {
  pr(PNATERR)
  pr(STATE)
  pr(AID-QUEUE)
  pr(AIDQERR)
  [MetaState]
  op __^__ : State Qu -> MetaState {constr} .
  ...
}
```

Operators to compute waittime I: find

```
op find : Qu Qu -> Qu&Err .
eq find(empQ, Q:Qu) = empQ .
eq find(A:Aid & Q:Qu, empQ) = errQ .
eq find(A:Aid & Q:Qu, A & L:Qu) = (A & find(Q, L))
    ↳ .
ceq find(A:Aid & Q:Qu, B:Aid & L:Qu) = (B & find(A
    ↳ & Q, L))
  if not (A == B) .
```

Operators to compute waittime I: find

```
op find : Qu Qu -> Qu&Err .
eq find(empQ, Q:Qu) = empQ .
eq find(A:Aid & Q:Qu, empQ) = errQ .
eq find(A:Aid & Q:Qu, A & L:Qu) = (A & find(Q, L))
    .
ceq find(A:Aid & Q:Qu, B:Aid & L:Qu) = (B & find(A
    & Q, L))
  if not (A == B) .
```

Example for find

```
find ( A & B , C & D & A & C & B & D & D ) = C &
    D & A & C & B
```

Operators to compute waittime II: droplast

Drops the last element of a queue

```
op droplast : Qu&Err -> Qu&Err .
eq droplast(empQ) = errQ .
eq droplast(errQ) = errQ .
eq droplast(A:Aid) = empQ .
eq droplast(A:Aid & B:Aid & Q:Qu) = (A & droplast(
    (B & Q)) .
```

Operators to compute waittime II: droplast

Drops the last element of a queue

```
op droplast : Qu&Err -> Qu&Err .
eq droplast(empQ) = errQ .
eq droplast(errQ) = errQ .
eq droplast(A:Aid) = empQ .
eq droplast(A:Aid & B:Aid & Q:Qu) = (A & droplast(
    (B & Q)) .
```

Example for droplast

```
droplast( A & B & C ) = A & B
```

Operators to compute waittime III: length

Length of a queue

```
op length : Qu&Err -> Nat&Err .
eq length(empQ) = 0 .
eq length(errQ) = err .
eq length(A:Aid & Q:Qu) = s(length(Q)) .
```

Operators to compute waittime IV: double

Duplicates each element in a queue

```
op double : Qu&Err -> Qu&Err .
eq double(empQ) = empQ .
eq double(errQ) = errQ .
eq double(A:Aid & Q:Qu) = A & A & double(Q) .
```

Operators to compute waittime IV: double

Duplicates each element in a queue

```
op double : Qu&Err -> Qu&Err .
eq double(empQ) = empQ .
eq double(errQ) = errQ .
eq double(A:Aid & Q:Qu) = A & A & double(Q) .
```

Example for double

```
double( A & B ) = A & A & B & B
```

Computation of waiting times

Assume that `Q:Qu`, `A:Aid`, `T:TransSeq`
Part in front of `A` in the Queue: `droplast(find(A,Q))`

rs-ws `length(find(A, T))`

Computation of waiting times

Assume that `Q:Qu`, `A:Aid`, `T:TransSeq`
Part in front of `A` in the Queue: `droplast(find(A,Q))`

rs-ws `length(find(A, T))`

ws-cs `length(find(double(droplast(find(A, Q)))& A , T))`

Computation of waiting times

Assume that `Q:Qu`, `A:Aid`, `T:TransSeq`
Part in front of `A` in the Queue: `droplast(find(A,Q))`

rs-ws `length(find(A, T))`

ws-cs `length(find(double(droplast(find(A, Q)))& A , T))`

cs-rs `find(A, T)`

Compute waittime

```
op waittime : Aid MetaState -> Nat&Err .
eq waittime(A:Aid, ((Q:Qu $ ((lb[A]: rs) AS:Aobs)) ^ (↴
    ↳T:Qu))) =
    length(find(A, T)) .
eq waittime(A:Aid, ((A & Q:Qu) $ ((lb[A]: ws) AS:↴
    ↳Aobs)) ^ (T:Qu)) =
    length(find(A, T)) .
ceq waittime(A:Aid, ((B:Aid & Q:Qu) $ ((lb[A]: ws) ↴
    ↳AS:Aobs)) ^ (T:Qu)) =
    length(find(double(droplast(find(A, Q))) & A, T)↴
        ↳)
    if not (A == B) .
eq waittime(A:Aid, (Q:Qu $ ((lb[A]: cs) AS:Aobs)) ^ (↴
    ↳T:Qu)) =
    length(find(A, T)) .
```

Meta-transitions

The meta-transitions are completely parallel to the normal transitions, just adding the transition sequence and only operating on the first element of the transition sequence.

```
mod! MWT {pr(METASTATE)
trans[mwt]:
    ( (Q:Qu $ ((lb[A:Aid]: rs) AS:Aobs)) ^ (A:Aid & T:
        Qu))
=> ( (Q & A) $ ((lb[A ]: ws) AS) ^ T ) .
}
```

Try transition

```
mod! MTY {pr(METASTATE)
trans[ty]:
          (((B:Aid & Q:Qu) $ ((lb[A:Aid]: ws) AS:Aobs))
               ) ^ (A & T:Qu))
     => if (A = B) then
          (((A & Q) $ ((lb[A]: cs) AS)) ^ T )
        else
          (((B & Q) $ ((lb[A]: ws) AS)) ^ T )
        fi .
}
```

EXIT TRANSITION

```
mod! MEX {pr(METASTATE)
trans[ex]:
          (((A:Aid & Q:Qu) $ ((lb[A]: cs) AS:Aobs)) ^ (↻
               ↺A & T:Qu))
       => ((Q $ ((lb[A]: rs) AS)) ^ T ) .
}
```

EXIT TRANSITION

```
mod! MEX {pr(METASTATE)
trans[ex]:
          (((A:Aid & Q:Qu) $ ((lb[A]: cs) AS:Aobs)) ^ (↻
               ↺A & T:Qu))
       => ((Q $ ((lb[A]: rs) AS)) ^ T ) .
}
```

```
mod! METAQLOCKsys{ pr(MWT + MTY + MEX) }
```

Verification of the original properties

Re-Verification of original properties

Idea: Reuse the same functions

```
mod! METASTATEpred-init {
  pr(STATEpred-init)
  pr(METAPNAMEcj)
  eq[aoa]: cj(aoa, ( S:State ^ T:Qu )) = cj(aoa, S) .
  ...
}
mod! METASTATEpred-inv {
  pr(METASTATE)
  pr(STATEpred-inv)
  pr(METAPNAMEcj)
  eq[mx]: cj(mx, ( S:State ^ T:Qu )) = cj(mx, S) .
  ...
}
```

Verification

With a few more necessary meta-modules the verification of the original properties goes through, even with the more extended state space (the covering set needs to cover also the transition sequence)

Verification of the remaining property

new invariant property

for each agent:

- at a transition the section does not change, then `waittime(A, S)> waittime(A, SS)`
  i.e., the waittime is reduced when no section change occurs
- if the waittime is 0 then a section change occurs

Verification of the remaining property

new invariant property

for each agent:

- at a transition the section does not change, then `waittime(A, S)> waittime(A, SS)`
  i.e., the waittime is reduced when no section change occurs
- if the waittime is 0 then a section change occurs

Current state

- initial set of invariants is defined
- case study of failing cases is in progress – need to add additional invariant properties

Conclusion

We have seen …

- new technique to reflect back parts of the meta-level into the language
- reusing and extending of existing code
- challenging open problems …

Conclusion

We have seen …

- new technique to reflect back parts of the meta-level into the language
- reusing and extending of existing code
- challenging open problems …

We still don't know if this is the emphcorrect or most optimal way to approach this problem – this is research!