DATAMANAGEMENT

PROJECT

By Andrea Nardocci - 1709323

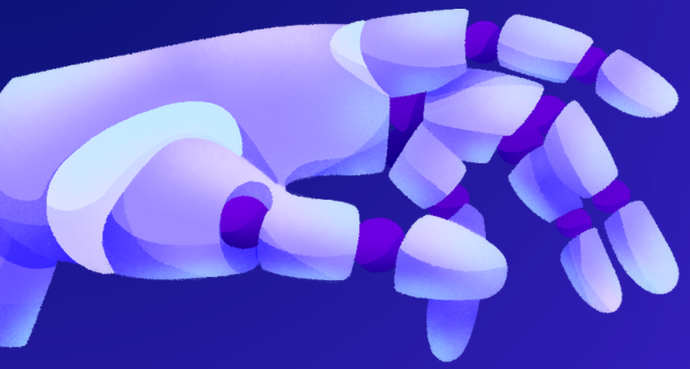
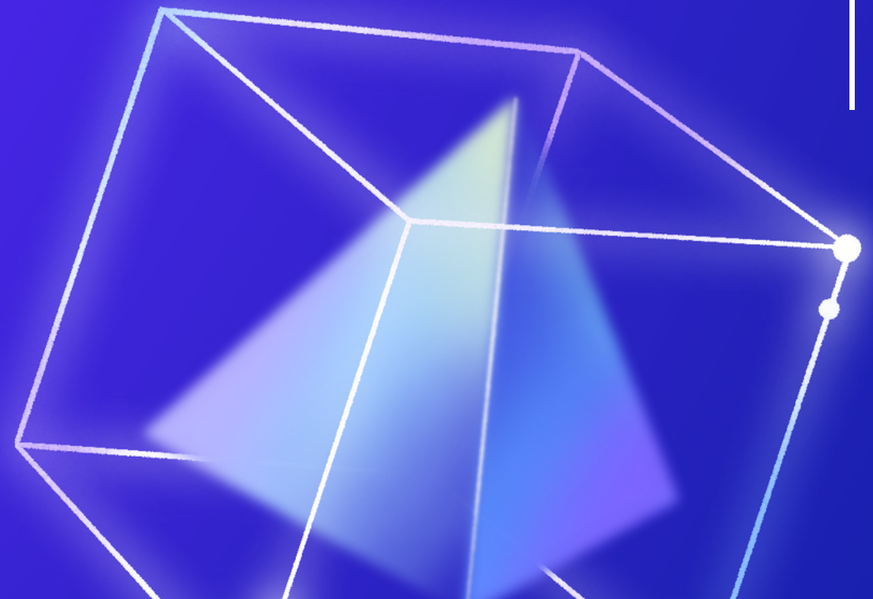


TABLE OF CONTENTS



INTRODUCTION

Software that emulates a scheduler based on a concurrency control via timestamp

Requirements

ALGORITHM IMPLEMENTATION

INIT VARIABLE

PROCESS THE INPUT

IMPLEMENTATION OF THE STRATEGY

DEADLOCK MANAGEMENT

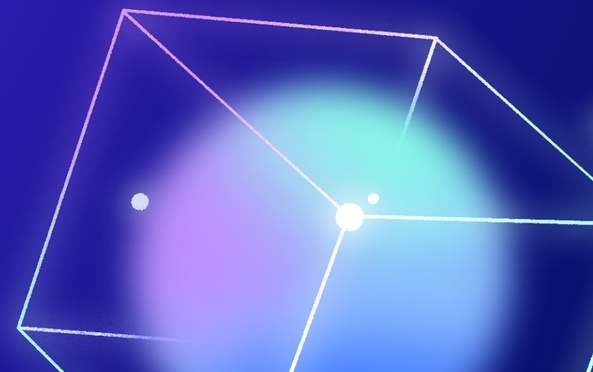
python™

CONCURRENCY CONTROL VIA TIMESTAMP

PROPERTIES : DATA FOR RESOURCES

Associated variable for each resource

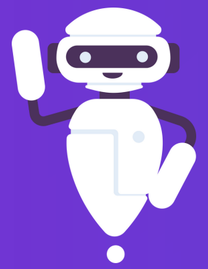
rts(X)	wts(X)
wts-c(X)	cb(X) (commit-bit)



CONCURRENCY CONTROL VIA TIMESTAMP

PROPERTIES : RULES

READ EXECUTION



READ EXEC

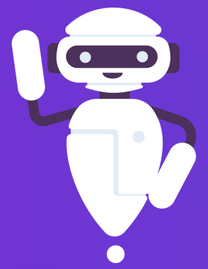
The read action changes only. a single parameter , rts(X)

```
if action_type == "read":
    if (transaction_ts >= resource_info[resource_index].wts):
        if (resource_info[resource_index].cb == True) or (transaction_ts  ==
        ↪  resource_info[resource_index].wts):
            ...
            resource_info[resource_index].rts = max(transaction_ts,
            ↪  resource_info[resource_index].rts)
        else:
            ...
            #add the action that generate waiting into deadlock_list_detector
            deadlock_detector.append(( transaction , action_type, resource ))
            transaction_info[transaction_index].waiting = True
            ignored_actions.append(elem)
            self.check_deadlock(( transaction , action_type, resource ))

    else:
        ...
        #add the transaction that need to be rollbacked
        rollback_transaction.append(transaction)
        transaction_info[transaction_index].rollback = True
        self.rollback(transaction_ts) #execute rollback
```

READ ERROR

It generate waiting or rollback

WRITE EXECUTION



WRITE EXEC

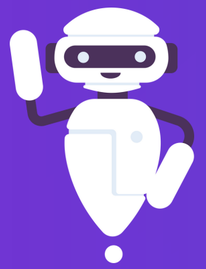
The write action changes cb(X) and wts.

```
if action_type == "write":
    if (transaction_ts >= resource_info[resource_index].rts) and (transaction_ts
    ↪  >= resource_info[resource_index].wts):
        if resource_info[resource_index].cb == True:
            resource_info[resource_index].wts = transaction_ts
            resource_info[resource_index].cb = False
            ...

        else:
            ...

            #add the action that generate waiting into deadlock_list_detector
            deadlock_detector.append(( transaction , action_type, resource ))
            transaction_info[transaction_index].waiting = True
            ignored_actions.append(( transaction , action_type, resource ))
            self.check_deadlock(( transaction , action_type, resource ))

    elif (transaction_ts >= resource_info[resource_index].rts) and (transaction_ts
    ↪  < resource_info[resource_index].wts):
        if resource_info[resource_index].cb == True:
            # Ignore the action
            ...
        else:
            ...
            #add the action that generate waiting into deadlock_list_detector
            deadlock_detector.append(( transaction , action_type, resource ))
            transaction_info[transaction_index].waiting = True
            ignored_actions.append(( transaction , action_type, resource ))
            self.check_deadlock(( transaction , action_type, resource ))
    else:
        ...
        #add the transaction that need to be rollbacked
        rollback_transaction.append(transaction)
        transaction_info[transaction_index].rollback = True
        self.rollback(transaction_ts) # execute rollback
```

WRITE ERROR

It generate waiting or rollback, and also we need to check whether we enter in the Thomas Rule

COMMIT EXECUTION



COMMIT EXEC

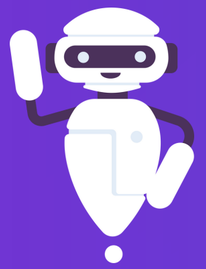
The commit action changes the commit bit cb(X) and also the wts_c(X)

```
    if action_type == "commit":
        ...
        # We can check in the array of the resourceinfo objects, if there's resources
        ↪  where the last write have the ts = to the ts of the commit, it means is the
        ↪  last transaction that wrote in this element,
        # so we need to set cb to true
        resource_to_check = None
        for index, elem in enumerate(resource_info):
            if elem.wts == transaction_ts :
                elem.cb = True
                elem.wts_c = transaction_ts
                resource_to_check = elem.name
        # Now we need to check if some other actions are in waiting for this commit.
        self.check_waiting(resource_to_check)
```

COMMIT RESULT

We need to check if there’re other transaction that are in waiting for the changes of the commit-bit to True.

ROLLBACK EXECUTION



ROLLBACK EXEC

Rollback action, puts wts(X) = wts_c(X) , and change the commit-bit. if the last transaction that wrote X is the one being rolledback.

```
def rollback(self,transaction_ts):
    resource_to_check = None
    for index, elem in enumerate(resource_info):
        if elem.wts == transaction_ts :
            elem.wts = elem.wts_c
            elem.cb = True
            resource_to_check = elem.name
            # Now we need to check if some other transaction is in waiting , each time
            ↪  we put to True a new variable
            self.check_waiting(resource_to_check)
```

ROLLBACK RESULTS

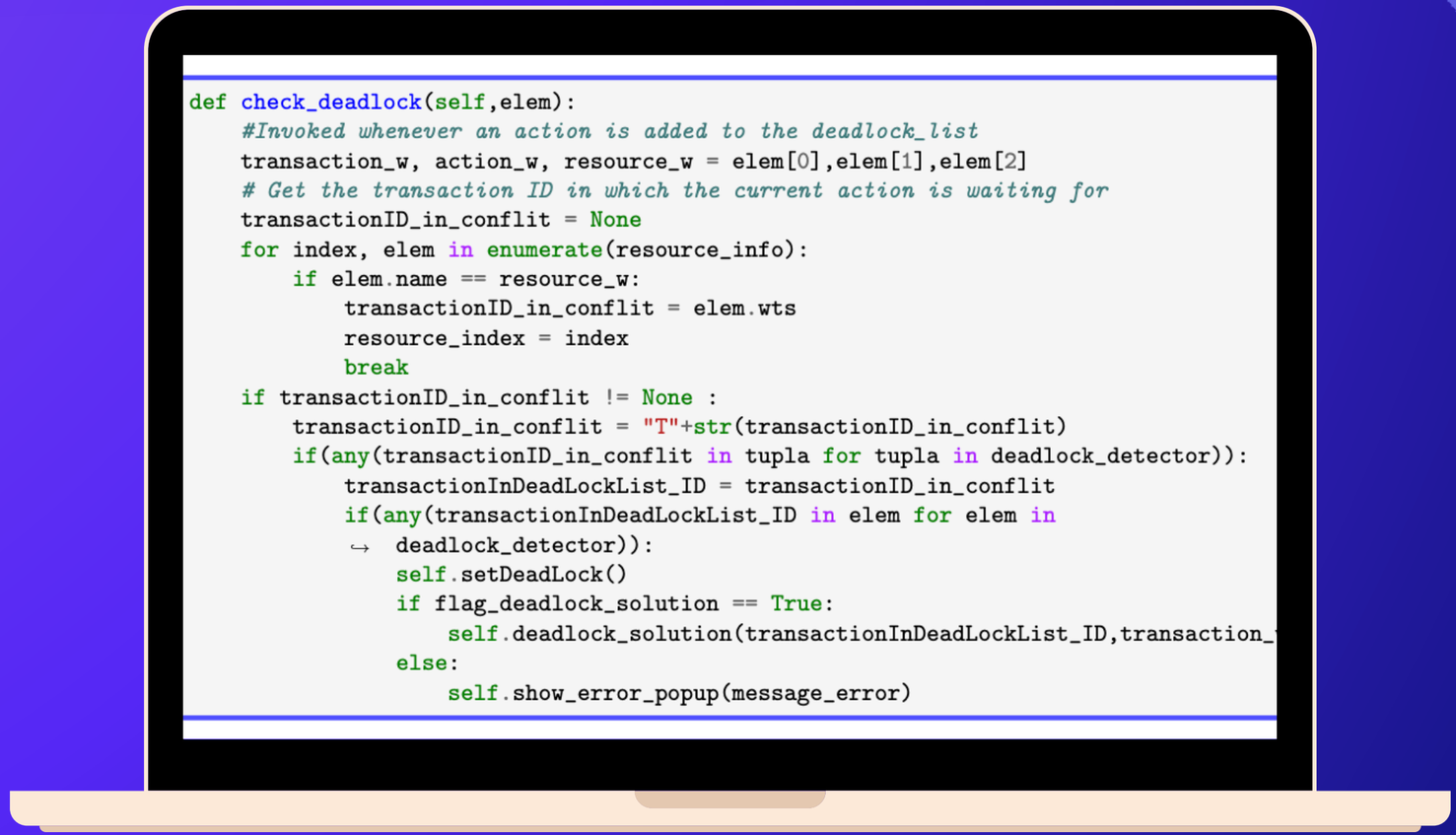
We need to check if there're other transaction that are in waiting for the changes of the commit-bit to True.

DEADLOCK MANAGEMENT

Deadlock Event

DeadLock Recognition

Deadlock Solution



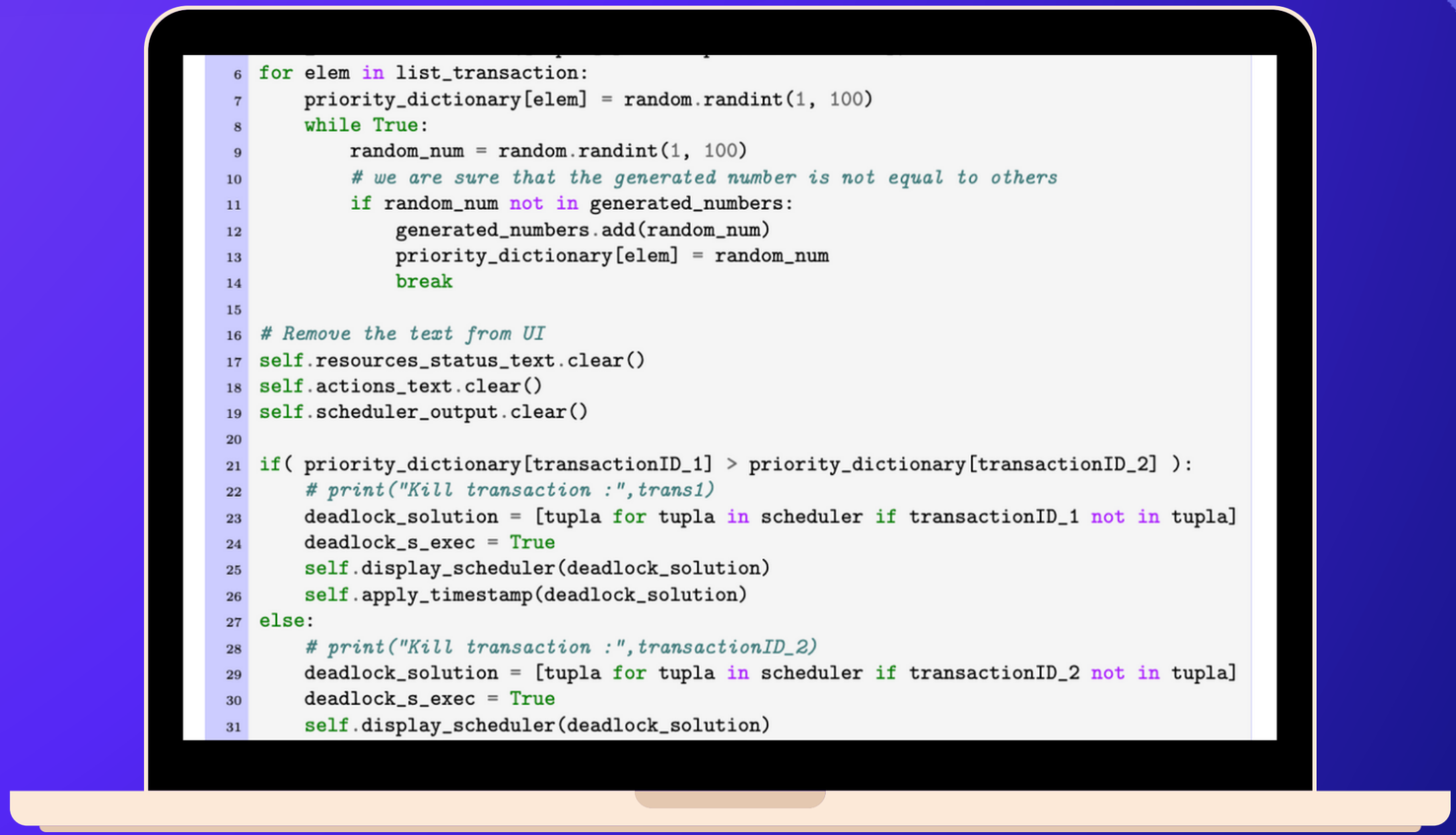
```
def check_deadlock(self,elem):
    #Invoked whenever an action is added to the deadlock_list
    transaction_w, action_w, resource_w = elem[0],elem[1],elem[2]
    # Get the transaction ID in which the current action is waiting for
    transactionID_in_conflit = None
    for index, elem in enumerate(resource_info):
        if elem.name == resource_w:
            transactionID_in_conflit = elem.wts
            resource_index = index
            break
    if transactionID_in_conflit != None :
        transactionID_in_conflit = "T"+str(transactionID_in_conflit)
        if(any(transactionID_in_conflit in tupla for tupla in deadlock_detector)):
            transactionInDeadLockList_ID = transactionID_in_conflit
            if(any(transactionInDeadLockList_ID in elem for elem in
            ↪  deadlock_detector)):
                self.setDeadLock()
                if flag_deadlock_solution == True:
                    self.deadlock_solution(transactionInDeadLockList_ID,transaction_
                else:
                    self.show_error_popup(message_error)
```

DEADLOCK MANAGEMENT

Deadlock Event

DeadLock Recognition

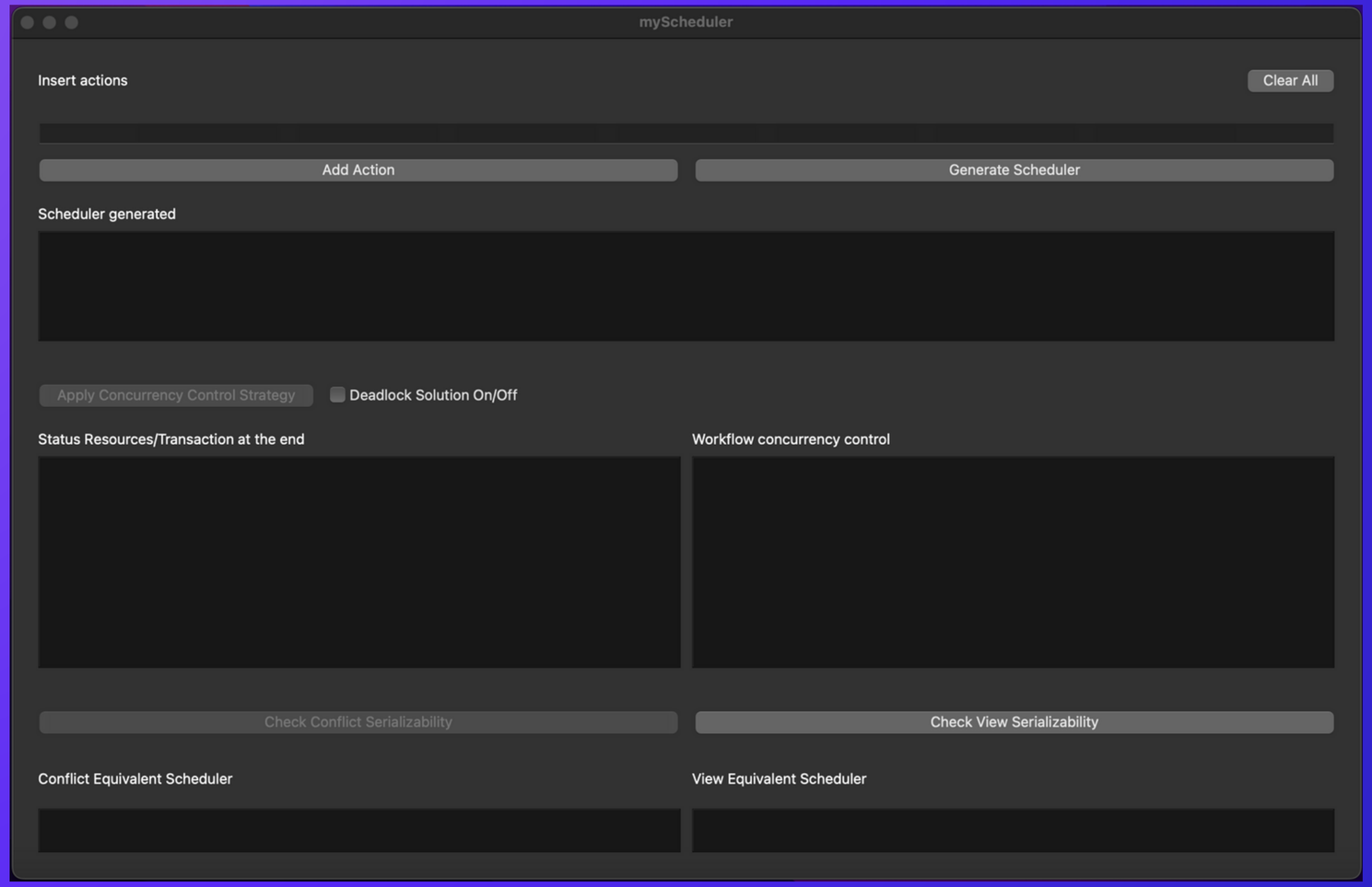
Deadlock Solution



```
for elem in list_transaction:
    priority_dictionary[elem] = random.randint(1, 100)
    while True:
        random_num = random.randint(1, 100)
        # we are sure that the generated number is not equal to others
        if random_num not in generated_numbers:
            generated_numbers.add(random_num)
            priority_dictionary[elem] = random_num
            break

# Remove the text from UI
self.resources_status_text.clear()
self.actions_text.clear()
self.scheduler_output.clear()

if( priority_dictionary[transactionID_1] > priority_dictionary[transactionID_2] ):
    # print("Kill transaction :",trans1)
    deadlock_solution = [tupla for tupla in scheduler if transactionID_1 not in tupla]
    deadlock_s_exec = True
    self.display_scheduler(deadlock_solution)
    self.apply_timestamp(deadlock_solution)
else:
    # print("Kill transaction :",transactionID_2)
    deadlock_solution = [tupla for tupla in scheduler if transactionID_2 not in tupla]
    deadlock_s_exec = True
    self.display_scheduler(deadlock_solution)
```

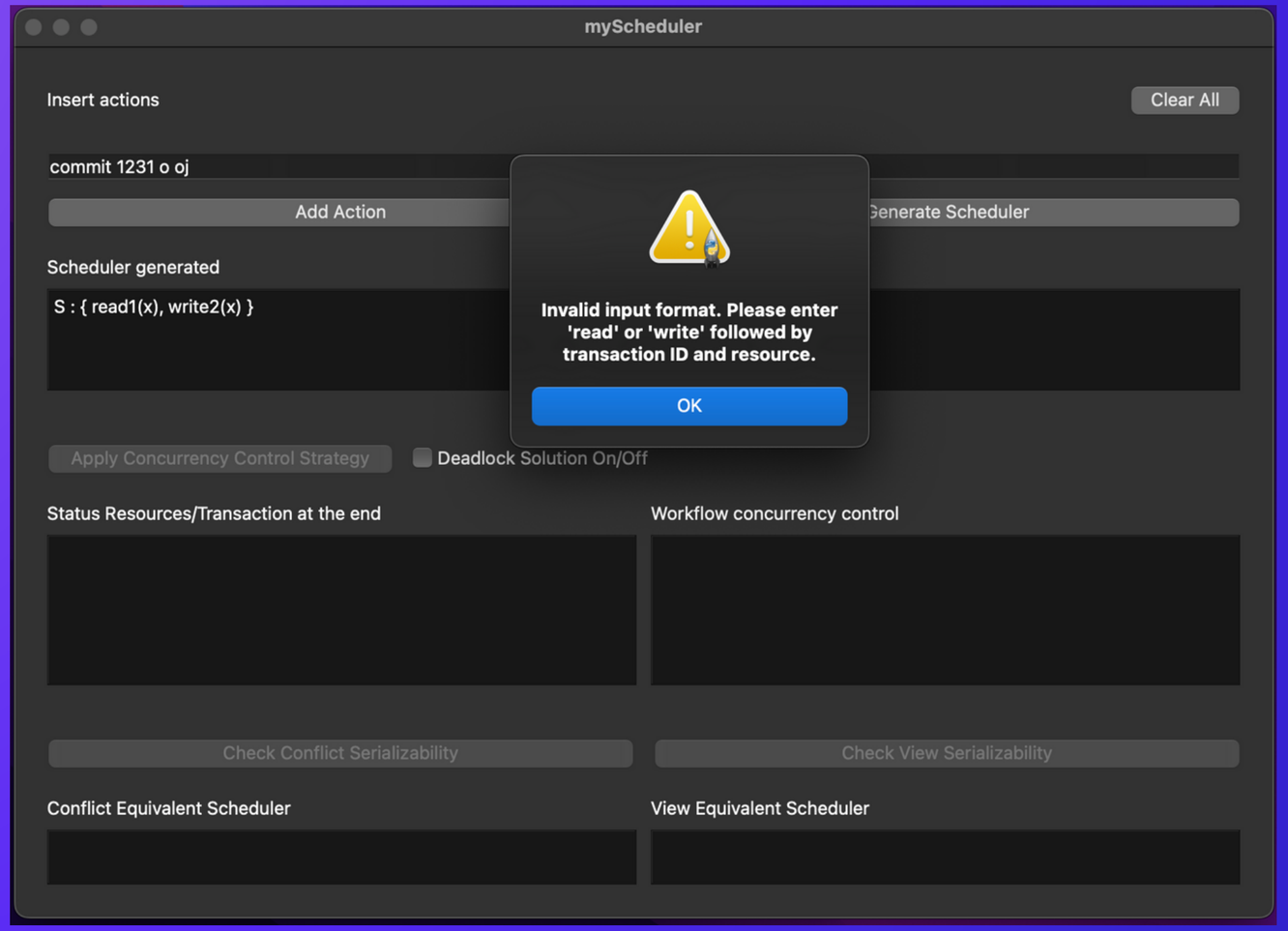
GUI

PyQt6


myScheduler
Insert actions
Clear All
Add Action
Generate Scheduler
Scheduler generated
Apply Concurrency Control Strategy
Deadlock Solution On/Off
Status Resources/Transaction at the end
Workflow concurrency control
Check Conflict Serializability
Check View Serializability
Conflict Equivalent Scheduler
View Equivalent Scheduler

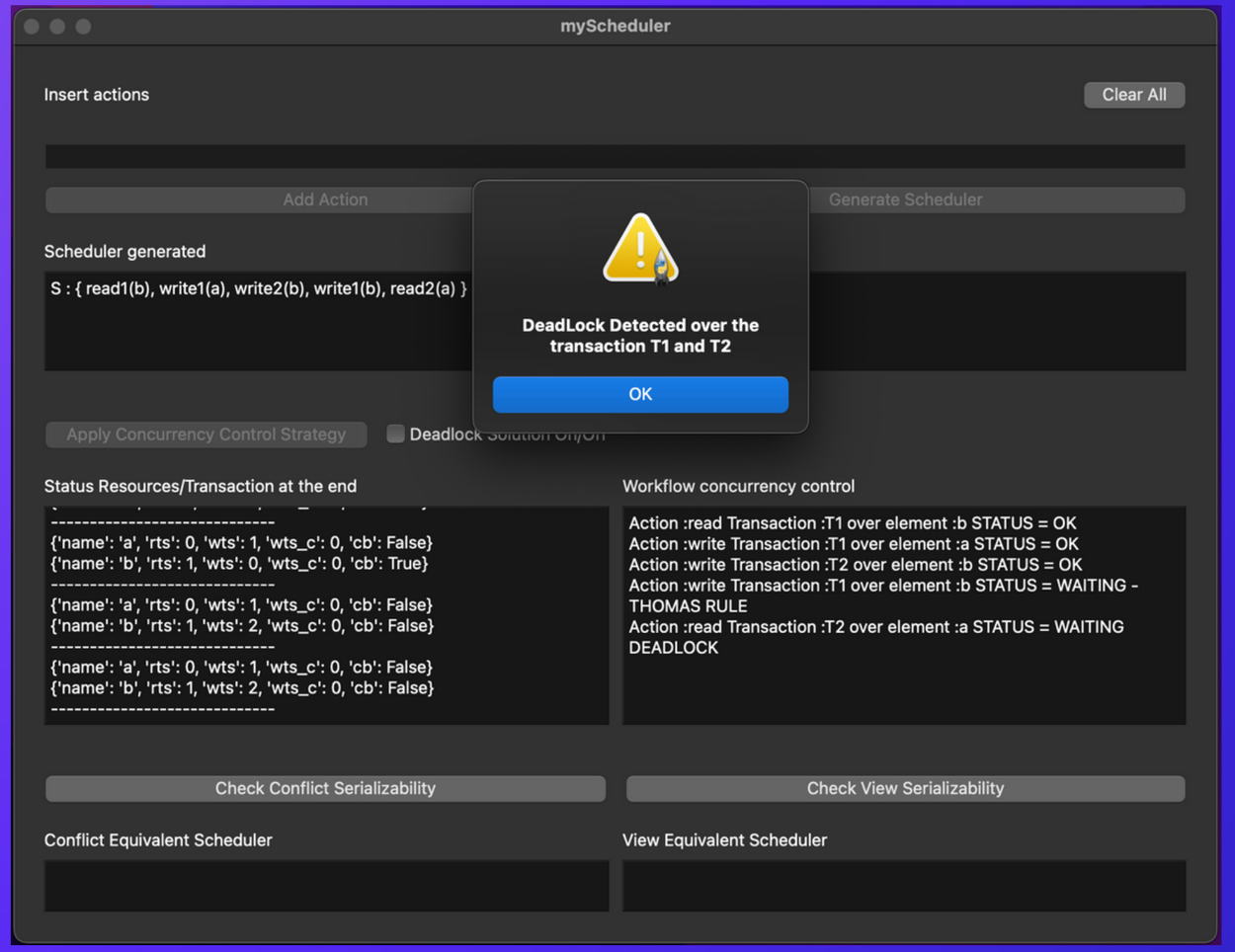
GUI

Input Error Message



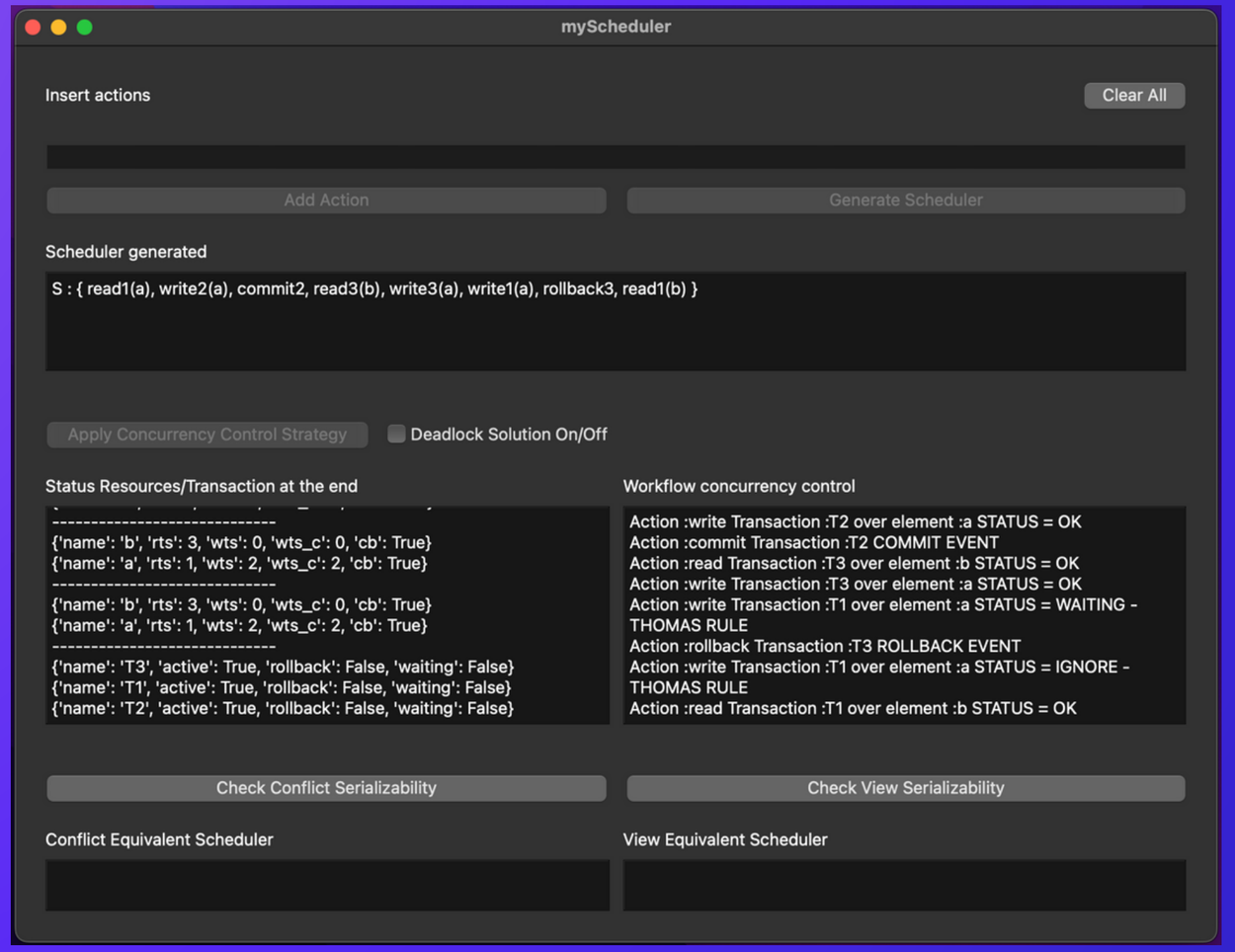
GUI

Deadlock Recognition



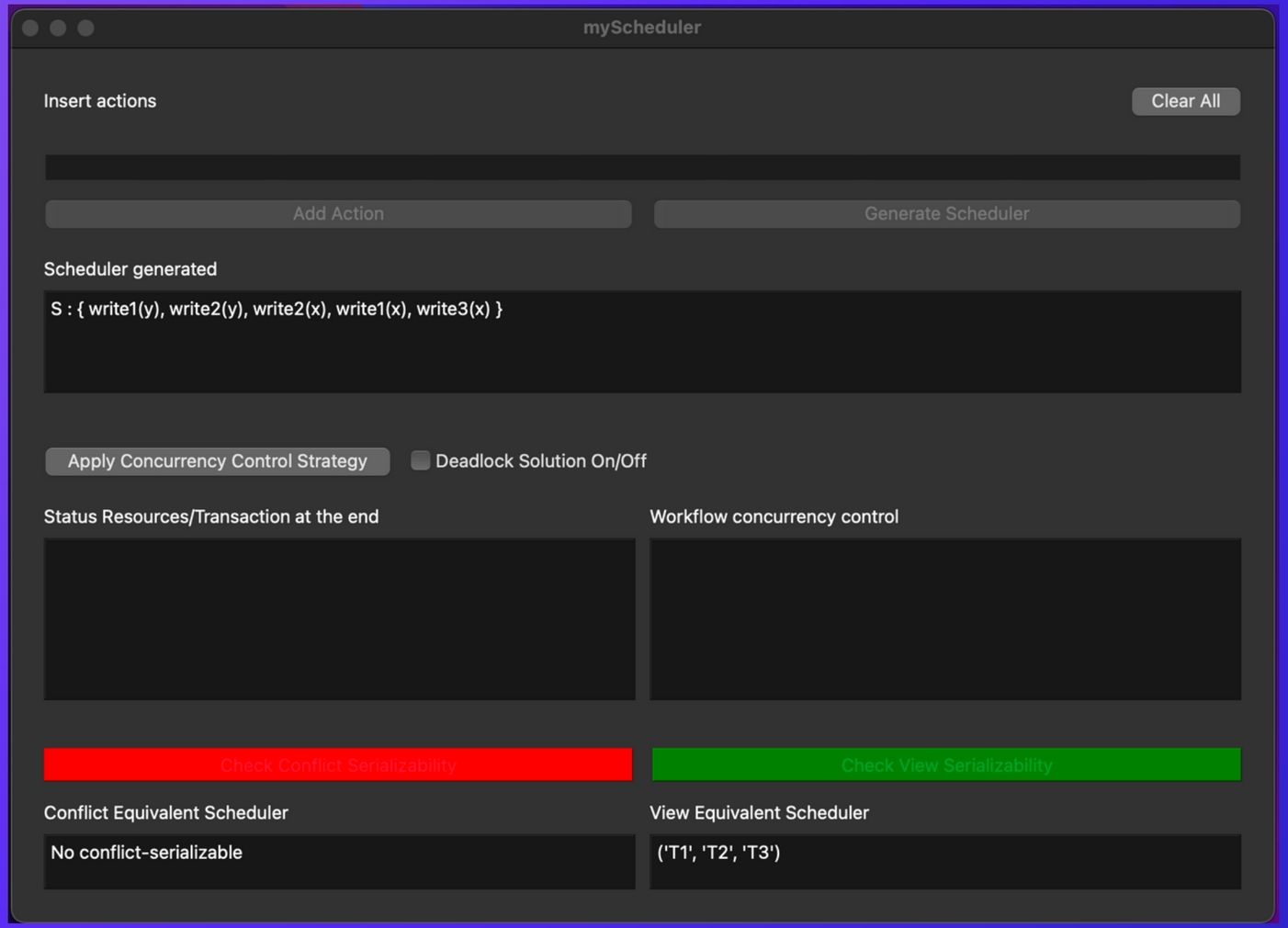
GUI

Concurrency



GUI

Extra Features



TEST AND RESULTS

RESULTS AND COMPARISON

SCHEDULER = [R1(X) R2 (X) W3(X) W3(Z) C3 R4(Z) W4(Y) C4 W1(Y) C1 R2(Y) C2]

01

Status Resources/Transaction at the end

```
{'name': 'x', 'rts': 2, 'wts': 3, 'wts_c': 3, 'cb': True}
{'name': 'y', 'rts': 0, 'wts': 4, 'wts_c': 4, 'cb': True}
{'name': 'z', 'rts': 4, 'wts': 3, 'wts_c': 3, 'cb': True}
----------------------------
{'name': 'x', 'rts': 2, 'wts': 3, 'wts_c': 3, 'cb': True}
{'name': 'y', 'rts': 0, 'wts': 4, 'wts_c': 4, 'cb': True}
{'name': 'z', 'rts': 4, 'wts': 3, 'wts_c': 3, 'cb': True}
----------------------------
{'name': 'T1', 'active': True, 'rollback': False, 'waiting': False}
{'name': 'T3', 'active': True, 'rollback': False, 'waiting': False}
{'name': 'T2', 'active': True, 'rollback': True, 'waiting': False}
{'name': 'T4', 'active': True, 'rollback': False, 'waiting': False}
```

Workflow concurrency control

```
Action :read Transaction :T1 over element :x STATUS = OK
Action :read Transaction :T2 over element :x STATUS = OK
Action :write Transaction :T3 over element :x STATUS = OK
Action :write Transaction :T3 over element :z STATUS = OK
Action :commit Transaction :T3 COMMIT EVENT
Action :read Transaction :T4 over element :z STATUS = OK
Action :write Transaction :T4 over element :y STATUS = OK
Action :commit Transaction :T4 COMMIT EVENT
Action :write Transaction :T1 over element :y STATUS = IGNORE - THOMAS RULE
Action :commit Transaction :T1 COMMIT EVENT
Action :read Transaction :T2 over element :y STATUS = ROLLBACK
```

02

$r_1(x) \rightarrow \text{OK} \rightarrow \text{rts}(x) = 1$

$r_2(x) \rightarrow \text{OK} \rightarrow \text{rts}(x) = 2$

$w_3(x) \rightarrow \text{OK} \rightarrow \text{wts}(x) = 3,\ \text{cb}(x) = \text{false}$

$w_3(z) \rightarrow \text{OK} \rightarrow \text{wts}(z) = 3,\ \text{cb}(z) = \text{false}$

$c_3 \rightarrow \text{OK} \rightarrow \text{wts-c}(x) = 3,\ \text{wts-c}(z) = 3,\ \text{cb}(x) = \text{true},\ \text{cb}(z) = \text{true}$

$r_4(z) \rightarrow \text{OK} \rightarrow \text{rts}(z) = 4$

$w_4(y) \rightarrow \text{OK} \rightarrow \text{wts}(y) = 4,\ \text{cb}(y) = \text{false}$

$c_4 \rightarrow \text{OK} \rightarrow \text{wts-c}(y) = 4,\ \text{cb}(y) = \text{true}$

$w_1(y) \rightarrow \text{OK}\ (\text{Thomas rule})$

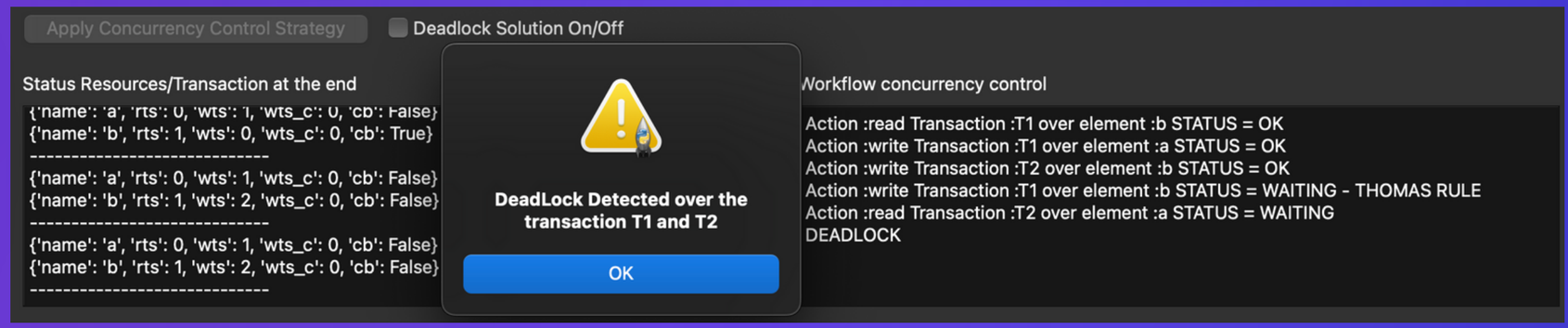
$c_1 \rightarrow \text{OK}$

$r_2(y) \rightarrow \text{read too late} \rightarrow T_2\ \text{aborted}$

RESULTS AND COMPARISON

SCHEDULER = [R1(B) W1(A) W2(B) W1(B) R2(A)]

01



02

r1(B) → ok → ts(T1)=1, rts(B)=1 -- because ts(T1)>= wts(B), cb(B)=true and rts(B)=max(ts(T1),rts(B))

w1(A) → ok → wts(A)=1 and cb(A)=false -- because ts(T1)>= wts(A), cb(A)=true and ts(T1)>= rts(A)

w2(B) → ok → ts(T2)=3, wts(B)=3 and cb(B)=false -- because ts(T2)>= wts(B), cb(B)=true and ts(T2)>= rts(B)

w1(B) → T1 waiting for the commit or rollback of T2 -- because cb(B)=false, ts(T1) = rts(B) and ts(T1)<wts(B)

r2(A) → T2 waiting for the commit or rollback of T1 because cb(A)=false and ts(T2)>wts(A) ==> **DEADLOCK!**

RESULTS AND COMPARISON

SCHEDULER = [R1(Z) R1(Y) W3(Y) R1(X) R2(X) C1 W4(Z) W2(X) W3(X) C3 R4(U) C4 W2 (U) C2]

01

Status Resources/Transaction at the end

```
{'name': 'y', 'rts': 1, 'wts': 3, 'wts_c': 3, 'cb': True}
{'name': 'z', 'rts': 1, 'wts': 4, 'wts_c': 4, 'cb': True}
----------------------------
{'name': 'u', 'rts': 4, 'wts': 0, 'wts_c': 0, 'cb': True}
{'name': 'x', 'rts': 2, 'wts': 3, 'wts_c': 3, 'cb': True}
{'name': 'y', 'rts': 1, 'wts': 3, 'wts_c': 3, 'cb': True}
{'name': 'z', 'rts': 1, 'wts': 4, 'wts_c': 4, 'cb': True}
----------------------------
{'name': 'T1', 'active': True, 'rollback': False, 'waiting': False}
{'name': 'T3', 'active': True, 'rollback': False, 'waiting': False}
{'name': 'T2', 'active': True, 'rollback': True, 'waiting': False}
{'name': 'T4', 'active': True, 'rollback': False, 'waiting': False}
```

Workflow concurrency control

```
Action :read Transaction :T1 over element :z STATUS = OK
Action :read Transaction :T1 over element :y STATUS = OK
Action :write Transaction :T3 over element :y STATUS = OK
Action :read Transaction :T1 over element :x STATUS = OK
Action :read Transaction :T2 over element :x STATUS = OK
Action :commit Transaction :T1 COMMIT EVENT
Action :write Transaction :T4 over element :z STATUS = OK
Action :write Transaction :T2 over element :x STATUS = OK
Action :write Transaction :T3 over element :x STATUS = WAITING
Action :read Transaction :T4 over element :u STATUS = OK
Action :commit Transaction :T4 COMMIT EVENT
Action :write Transaction :T2 over element :u STATUS = ROLLBACK
```

02

$r_1(z)$	→ OK,	rts(z)=1
$r_1(y)$	→ OK,	rts(y)=1
$w_3(y)$	→ OK,	wts(y)=3, cb(y)=`false`
$r_1(x)$	→ OK,	rts(x)=1
$r_2(x)$	→ OK,	rts(x)=2
c_1	→ OK,	
$w_4(z)$	→ OK,	wts(z)=4
$w_2(x)$	→ OK,	wts(x)=2, cb(x)=`false`
$w_3(x)$	→ OK,	transaction 3 suspended
$r_4(u)$	→ OK,	rts(u)=4
c_4	→ OK,	wts-c(z)=4, cb(z)=`true`
$w_2(u)$	→ write too late,	transaction 2 rollbacks
$w_3(x)$	→ OK,	wts(x)=3
c_3	→ OK,	wts-c(x)=3, cb(x)=`true`

THANK YOU!

GitHub Repository