Simulation Komplexer Systeme

Scheduler

Übungsaufgaben zu Kapitel 13

Autor: Richard Grünert

3.4.2023

1 Teil A: 2-Phasen-Scheduler

1.1 Prototyp 2-Phasen-Scheduler sched2p

sched2p.py

```python
#!/usr/bin/env python3
#
# Author: Richard Grünert, 2023
#

from enum import Enum
from collections import deque
import sys
import csv


class Event:
    def __init__(self, start_time, callback):
        self.start_time = start_time
        self.callback = callback


class EventList(deque):
    """
    Verkettete Liste zum Sortieren der Ereignisse
    sortiert nach aufsteigender Zeit (früheste Ereignisse sind "links")
    """
    def __init__(self):
        super().__init__(self)

    def add_event(self, event):
        """
        Fügt der Eventliste ein Event-Objekt an der geeigneten Position hinzu
        """
        if len(self) == 0:
            self.appendleft(event)
            return

        i = 0
        for node in self:
            if event.start_time <= node.start_time:
                self.insert(i, event)
                return
            i += 1

        self.append(event)

    def pop_next_event(self):
        """
        Entfernt das nächste (linke) Ereignis der Liste und gibt es zurück
        """
        return self.popleft()
```

```
48
49
50  class Scheduler2Phase:
51      """
52      Scheduler Klasse ohne bedingte Ereignisse
53      """
54      def __init__(self, sim_state):
55          self.sim_state = sim_state
56          self.event_list = EventList()
57          self.current_time = 0.0
58          self.terminated = False
59
60      def terminate(self):
61          """
62          Teilt dem Scheduler mit, im nächsten Simulationsschritt zu
        terminieren
63          """
64          self.terminated = True
65
66      def schedule(self, event):
67          """
68          Fügt der Ereignisliste ein Ereignisobjekt hinzu, sofern es nicht
69          in der Vergangenheit liegt
70          """
71          if event.start_time < self.current_time:
72              raise ValueError('Cannot schedule an event in the past')
73
74          self.event_list.add_event(event)
75
76      def simulate_step(self):
77          """
78          Führt einen einzelnen Simulationsschritt aus, bestehend aus
79          - Terminierung prüfen
80          - nächstes Ereignis holen
81          - Simulationszeit erhöhen
82          - Eventfunktionen aufrufen
83
84          Gibt "False" zurück, wenn der Schritt zur Terminierung geführt hat
85          ansonsten "True"
86          """
87
88          if self.terminated:
89              return False
90
91          if len(self.event_list) == 0:
92              return False
93
94          event = self.event_list.pop_next_event()
95
96          self.current_time = event.start_time
97
98          event.callback(self)
99
```

```
100         return True
101
102 # ========================================
103
104 class TrafficLightStates(Enum):
105     """
106     Codierung für die Ampelzustände
107     """
108     RED = 0
109     GREEN = 1
110
111
112 class TrafficLightSimulationState:
113     """
114     Hält alle Zustandsvariablen
115     """
116     def __init__(self):
117         self.traffic_light_state = None
118
119
120 def state_red_function(scheduler):
121     """
122     Funktion, die mit dem Ereignis "Wechsel zu Rot" ausgeführt wird
123     """
124
125     scheduler.sim_state.traffic_light_state = TrafficLightStates.RED
126
127     scheduler.schedule(Event(
128         scheduler.current_time + 180.12345,
129         state_green_function))
130
131
132 def state_green_function(scheduler):
133     """
134     Funktion, die mit dem Ereignis "Wechsel zu Grün" ausgeführt wird
135     """
136
137     scheduler.sim_state.traffic_light_state = TrafficLightStates.GREEN
138
139     scheduler.schedule(
140         Event(scheduler.current_time + 60.56789,
141               state_red_function))
142
143
144 def state_terminate_function(scheduler):
145     """
146     Funktion, die mit dem Terminierungs-Ereignis ausgeführt wird
147     """
148
149     scheduler.terminate()
150
151 # ========================================
152
```

```
153 if __name__ == "__main__":
154
155     sim_state = TrafficLightSimulationState()
156
157     my_scheduler = Scheduler2Phase(sim_state)
158
159     my_scheduler.schedule(Event(0.0, state_red_function))
160     my_scheduler.schedule(Event(1.8144e6, state_terminate_function))
161
162     # write result to csv
163     writer = csv.writer(sys.stdout)
164
165     writer.writerow(["time", "state"])
166
167     while my_scheduler.simulate_step() is not False:
168
169         if not my_scheduler.terminated:
170             writer.writerow([
171                 my_scheduler.current_time,
172                 my_scheduler.sim_state.traffic_light_state
173             ])
```

Ausgabe sched2p bei Simulationszeit von 10 Minuten

```
1 time,state
2 0.0,TrafficLightStates.RED
3 180.12345,TrafficLightStates.GREEN
4 240.69134,TrafficLightStates.RED
5 420.81479,TrafficLightStates.GREEN
6 481.38268,TrafficLightStates.RED
```

1.2 Zeitgetriebener Simulator (sched2p_timedriven)

sched2p_timedriven.py

```
1  #!/usr/bin/env python3
2  #
3  # Author: Richard Grünert, 2023
4  #
5
6  from enum import Enum
7  from collections import deque
8  import sys
9  import csv
10
11
12 class Event:
13     def __init__(self, start_time, execute_fcn):
14         self.start_time = start_time
15         self.execute = execute_fcn
16
17
18 class EventList(deque):
```

```
    def __init__(self):
        super().__init__(self)

    def add_event(self, event):
        if len(self) == 0:
            self.appendleft(event)
            return

        i = 0
        for node in self:
            if event.start_time <= node.start_time:
                self.insert(i, event)
                return
            i += 1

        self.append(event)

    def pop_next_event(self):
        return self.popleft()


class SchedulerTimeDriven:
    def __init__(self, sim_state, time_step):
        self.sim_state = sim_state
        self.time_step = time_step
        self.event_list = EventList()
        self.current_time = 0.0
        self.terminated = False

    def terminate(self):
        self.terminated = True

    def schedule(self, event):
        if event.start_time < self.current_time:
            raise ValueError('Cannot schedule an event in the past')

        self.event_list.add_event(event)

    def simulate_step(self):

        if self.terminated:
            return False

        if len(self.event_list) == 0:
            return False

        while self.event_list[0].start_time < self.current_time:
            event = self.event_list.pop_next_event()
            event.execute(self)

        self.current_time += time_step

        return True
```

```
72
73  # ================================
74
75  class TrafficLightStates(Enum):
76      RED = 0
77      GREEN = 1
78
79
80  class TrafficLightSimulationState:
81      def __init__(self, initial):
82          self.traffic_light_state = initial
83
84
85  def state_red_function(scheduler):
86
87      scheduler.sim_state.traffic_light_state = TrafficLightStates.RED
88
89      scheduler.schedule(Event(
90          scheduler.current_time + 180.12345,
91          state_green_function))
92
93
94  def state_green_function(scheduler):
95
96      scheduler.sim_state.traffic_light_state = TrafficLightStates.GREEN
97
98      scheduler.schedule(
99          Event(scheduler.current_time + 60.56789,
100               state_red_function))
101
102
103 def state_terminate_function(scheduler):
104
105     print("terminated")
106
107     scheduler.terminate()
108
109 # ================================
110
111 if __name__ == "__main__":
112
113     sim_state = TrafficLightSimulationState(TrafficLightStates.RED)
114
115     time_step = 1.0
116
117     my_scheduler = SchedulerTimeDriven(sim_state, time_step)
118
119     my_scheduler.schedule(Event(0, state_red_function))
120     my_scheduler.schedule(Event(600.0, state_terminate_function))
121
122     # write result to csv
123     writer = csv.writer(sys.stdout)
124
```

```
    writer.writerow(["time", "state"])

    while my_scheduler.simulate_step() is not False:

        if not my_scheduler.terminated:
            writer.writerow([
                my_scheduler.current_time,
                my_scheduler.sim_state.traffic_light_state
            ])
```

Teil der Ausgabe sched2p_timedriven bei Simulationszeit von 10 Minuten. Schrittweite = 1 s

```
time,state
...
422.0,TrafficLightStates.RED
423.0,TrafficLightStates.RED
424.0,TrafficLightStates.RED
425.0,TrafficLightStates.GREEN
426.0,TrafficLightStates.GREEN
427.0,TrafficLightStates.GREEN
...
483.0,TrafficLightStates.GREEN
484.0,TrafficLightStates.GREEN
485.0,TrafficLightStates.GREEN
486.0,TrafficLightStates.RED
487.0,TrafficLightStates.RED
488.0,TrafficLightStates.RED
...
599.0,TrafficLightStates.RED
600.0,TrafficLightStates.RED
601.0,TrafficLightStates.RED
terminated
```

Die Ausgabe bei 601 Sekunden liegt daran, dass im Simulationsschritt die Zeit nach dem Ausführen des Ereignisses erhöht wird. Die eigentliche Zeit bei Eintreten des Ereignisses liegt also immer 1 s früher als es die Ausgabe zeigt.

1.3 Vergleich von sched2p und sched2p_timedriven

Man erkennt gut aus der Ausgabe von `sched2p_timedriven` , dass die Zeitpunkte der Ereigniswechsel nicht genau getroffen werden können und die Genauigkeit immer von der Zeitauflösung / Schrittweite abhängt. Höhere Genauigkeit bringt allerdings auch eine erhöhte Anzahl an Iterationen mit sich, wodurch die Simulationszeit (Echtzeit) verlängert wird.

Eine Ausgabe der Anzahl der Iterationen zeigt 600, wie erwartet ($\frac{600\,\mathrm{s}}{1\,\mathrm{s/step}}$). Im Gegensatz zum Ereignisbasierten Scheduler mit 5 Iterationen liegt der Unterschied also bei einem Faktor von 120.

Außerdem zeigt der zeitbasierte Ansatz (in diesem Fall), dass die meisten Iterationen keine Zustandsänderungen herbeiführen und damit nutzlos sind.

Simulationszeit von 1 Woche: Eine Woche entspricht 604800 Sekunden. Mittels `time()` -Funktion aus der `time` library in Python konnten die Ausführungszeiten beider Programme geprüft werden.

$$t_{\text{sched2p}} = 0.00652\,\text{s}$$
$$t_{\text{timedriven}} = 0.182\,\text{s}$$
$$\frac{t_{\text{timedriven}}}{t_{\text{sched2p}}} \approx 30$$

2 Teil B: 3-Phasen Scheduler

sched3p.py

```python
#!/usr/bin/env python3
#
# Author: Richard Grünert, 2023
#

from enum import Enum
from collections import deque
import random
import sys
import csv


class Event:
    """
    Ein Event hält Referenzen auf eine auszuführende Funktion sowie
    optional auf eine Bedingungsfunktion.
    """
    def __init__(self, execute_fcn, condition_fcn = None):
        self.execute = execute_fcn
        self.condition_fcn = condition_fcn


class EventNotice:
    """
    Ein EventNotice stellt einen Eintrag in der EventList dar.
    Es hält das Event und die Zeit, zu der es ausgeführt werden
    soll.
    """
    def __init__(self, event, start_time):
        self.event = event
        self.start_time = start_time


class EventList(deque):
    """
    Die EventList hält EventNotice-Objekte.
    Sie ist sortiert nach der Eintrittszeit des Events.
    Das zeitlich näheste Event ist "links" / vorne
    in der Liste.
    """
    def __init__(self):
        super().__init__(self)
```

```
43
44      def add_event_notice(self, notice):
45
46          if len(self) == 0:
47              self.appendleft(notice)
48              return
49
50          i = 0
51          for node in self:
52              if notice.start_time <= node.start_time:
53                  self.insert(i, notice)
54                  return
55              i += 1
56
57          self.append(notice)
58
59      def pop_next_notice(self):
60          return self.popleft()
61
62
63  class Scheduler3Phase:
64      """
65      Notices werden nur gescheduled, wenn ihre Bedinung
66      zum schedule-Zeitupunkt erfüllt ist
67      """
68      def __init__(self, sim_state):
69          self.sim_state = sim_state
70          self.event_list = EventList()
71          self.current_time = 0.0
72          self.terminated = False
73
74      def terminate(self):
75          self.terminated = True
76
77      def schedule(self, start_time, event):
78          if start_time < self.current_time:
79              raise ValueError('Cannot schedule an event in the past')
80
81          if event.condition_fcn is not None\
82             and event.condition_fcn(self.sim_state) is False:
83              return
84
85          self.event_list.add_event_notice(EventNotice(event, start_time))
86
87      def simulate_step(self):
88
89          if self.terminated:
90              return False
91
92          if len(self.event_list) == 0:
93              return False
94
95          notice = self.event_list.pop_next_notice()
```

```
96
97          self.current_time = notice.start_time
98          notice.event.execute(self)
99
100         return True
101
102 # ============================
103
104 class TrafficLightStates(Enum):
105     """
106     Codierung für die Ampelzustände
107     """
108     RED = 0
109     GREEN = 1
110
111
112 class TrafficLightSimulationState:
113     """
114     Hält alle Zustandsvariablen
115     """
116     def __init__(self):
117         self.traffic_light_state = None
118         self.previous_traffic_light_state = None
119         self.waiting = None
120         self.red_red_transitions = 0
121         self.red_green_transitions = 0
122
123
124 def state_red_conditional_function(scheduler):
125     if scheduler.sim_state.waiting < 0.5:
126         state_red_function(scheduler)
127         scheduler.sim_state.red_red_transitions += 1
128     else:
129         state_green_function(scheduler)
130         scheduler.sim_state.red_green_transitions += 1
131
132
133 def are_enough_people_waiting(sim_state):
134     return sim_state.waiting > 0.5
135
136
137 def state_red_function(scheduler):
138     """
139     Funktion, die mit dem Ereignis "Wechsel zu Rot" ausgeführt wird
140     """
141     scheduler.sim_state.traffic_light_state = TrafficLightStates.RED
142
143     prev_state = scheduler.sim_state.previous_traffic_light_state
144
145     # Übergänge zählen für Auswertung
146     if prev_state is not None:
147         if prev_state is TrafficLightStates.RED:
148             scheduler.sim_state.red_red_transitions += 1
```

```
149             else:
150                 scheduler.sim_state.red_green_transitions += 1
151 
152         scheduler.sim_state.waiting = random.uniform(0.0, 1.0)
153 
154         scheduler.schedule(
155             scheduler.current_time + 180.12345,
156             Event(state_green_function, are_enough_people_waiting))
157 
158         scheduler.schedule(
159             scheduler.current_time + 180.12345,
160             Event(state_red_function,
161                   lambda sim_state: not are_enough_people_waiting(sim_state)))
162 
163         scheduler.sim_state.previous_traffic_light_state = TrafficLightStates.
        RED
164 
165         ## Alternativ: mit Hilfsereignis
166         # scheduler.schedule(
167         #     scheduler.current_time + 180.12345,
168         #     Event(state_red_conditional_function))
169 
170 
171 def state_green_function(scheduler):
172     """
173     Funktion, die mit dem Ereignis "Wechsel zu Grün" ausgeführt wird
174     """
175     scheduler.sim_state.traffic_light_state = TrafficLightStates.GREEN
176 
177     scheduler.schedule(
178         scheduler.current_time + 60.56789,
179         Event(state_red_function))
180 
181     scheduler.sim_state.previous_traffic_light_state = TrafficLightStates.
    GREEN
182 
183 
184 def state_terminate_function(scheduler):
185     """
186     Funktion, die mit dem Terminierungs-Ereignis ausgeführt wird
187     """
188     scheduler.terminate()
189 
190 # ============================
191 
192 if __name__ == "__main__":
193 
194     #random.seed(123)
195 
196     sim_state = TrafficLightSimulationState()
197 
198     my_scheduler = Scheduler3Phase(sim_state)
199 
```

```
200     my_scheduler.schedule(0.0, Event(state_red_function))
201     my_scheduler.schedule(600.0, Event(state_terminate_function))
202
203     # write result to csv
204     writer = csv.writer(sys.stdout)
205
206     writer.writerow(["time", "state"])
207
208     while my_scheduler.simulate_step() is not False:
209
210         if not my_scheduler.terminated:
211             writer.writerow([
212                 my_scheduler.current_time,
213                 my_scheduler.sim_state.traffic_light_state
214             ])
215
216     print(my_scheduler.sim_state.red_red_transitions)
217     print(my_scheduler.sim_state.red_green_transitions)
218     print(my_scheduler.sim_state.red_green_transitions
219           / my_scheduler.sim_state.red_red_transitions)
```

Simulationslauf	Rot-Rot-Übergänge	Rot-Grün-Übergänge	Zustandsverlauf
1	1	1	R-R-G-R
2	0	2	R-G-R-G-R
3	1	1	R-G-R-R
4	1	1	R-R-G-R
5	2	0	R-R-R-G
6	1	1	R-R-G-R
7	1	1	R-R-G-R
8	3	0	R-R-R-R
9	0	2	R-G-R-G-R
10	1	1	R-G-R-R

Tabelle 1: Simulationsergebnis sched3p bei 10 Druchläufen mit Simulationszeit von 10 min

Simulationslauf	Rot-Rot-Übergänge	Rot-Grün-Übergänge	Verhältnis (R-R/R-G)
1	5851	5672	0.9694
2	5732	5761	1.0051
3	5695	5789	1.0165
4	5681	5799	1.0208
5	5749	5748	0.9998
6	5729	5763	1.0059
7	5743	5753	1.0017
8	5745	5751	1.0010
9	5733	5760	1.0047
10	5732	5761	1.0051
		Mittelwert	1.003

Tabelle 2: Simulationsergebnis sched3p bei 10 Druchläufen mit Simulationszeit von 4 Wochen