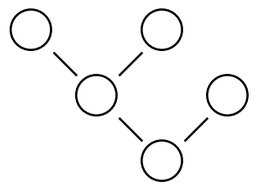
behavioral
complexity

```
def baking(oven, e):
  status = return_status.UNHANDLED
  if(e.signal == signals.ENTRY_SIGNAL):
    print("baking")
    status = return_status.HANDLED
  else:
    oven.temp.fun = heating
    status = return_status.SUPER
  return status
```

```
def toasting(oven, e):
  status = return_status.UNHANDLED
  if(e.signal == signals.ENTRY_SIGNAL):
    print("toasting")
    status = return_status.HANDLED
  else:
    oven.temp.fun = heating
    status = return_status.SUPER
  return status
```

The state callbacks describe a directed acyclic graph (DAG)



```
def heating(oven, e):
  status = return_status.UNHANDLED
  if(e.signal == signals.ENTRY_SIGNAL):
    oven.heater_on()
    status = return_status.HANDLED
  elif(e.signal == signals.EXIT_SIGNAL):
    oven.heater_off()
    status = return_status.HANDLED
  else:
    oven.temp.fun = door_closed
    status = return_status.SUPER
  return status
```

```
def off(oven, e):
  status = return_status.UNHANDLED
  if(e.signal == signals.ENTRY_SIGNAL):
    print("off")
    status = return_status.HANDLED
  else:
    oven.temp.fun = door_closed
    status = return_status.SUPER
  return status
```

```
def door_closed(oven, e):
  status = return_status.UNHANDLED
  if(e.signal == signals.ENTRY_SIGNAL):
    status = return_status.HANDLED
  elif(e.signal == signals.Baking):
    status = oven.trans(baking)
  elif(e.signal == signals.Toasting):
    status = oven.trans(toasting)
  elif(e.signal == signals.INIT_SIGNAL):
    status = oven.trans(off)
  elif(e.signal == signals.Off):
    status = oven.trans(off)
  else:
    oven.temp.fun = oven.top
    status = return_status.SUPER
  return status
```