# Exercise

Statecharts

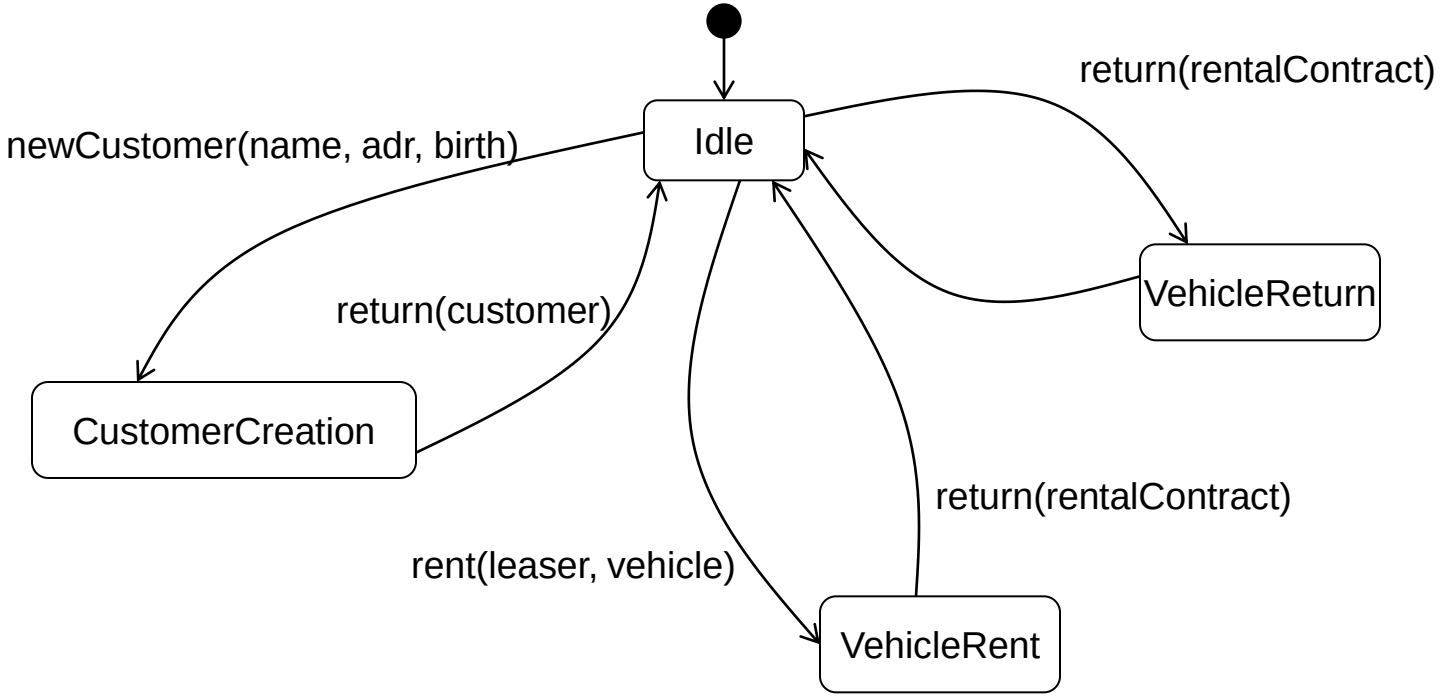

Prof. Dr. Bernhard Rumpe
Lehrstuhl für Software Engineering
RWTH Aachen

http://www.se-rwth.de/

# Exercise 7.1 – Statechart of Object Behavior

- Model a statechart which represents the states of the car rental during a customer service. Use at least the states “Idle”, “VehicleReturn”, “CustomerCreation” and “VehicleRent”.

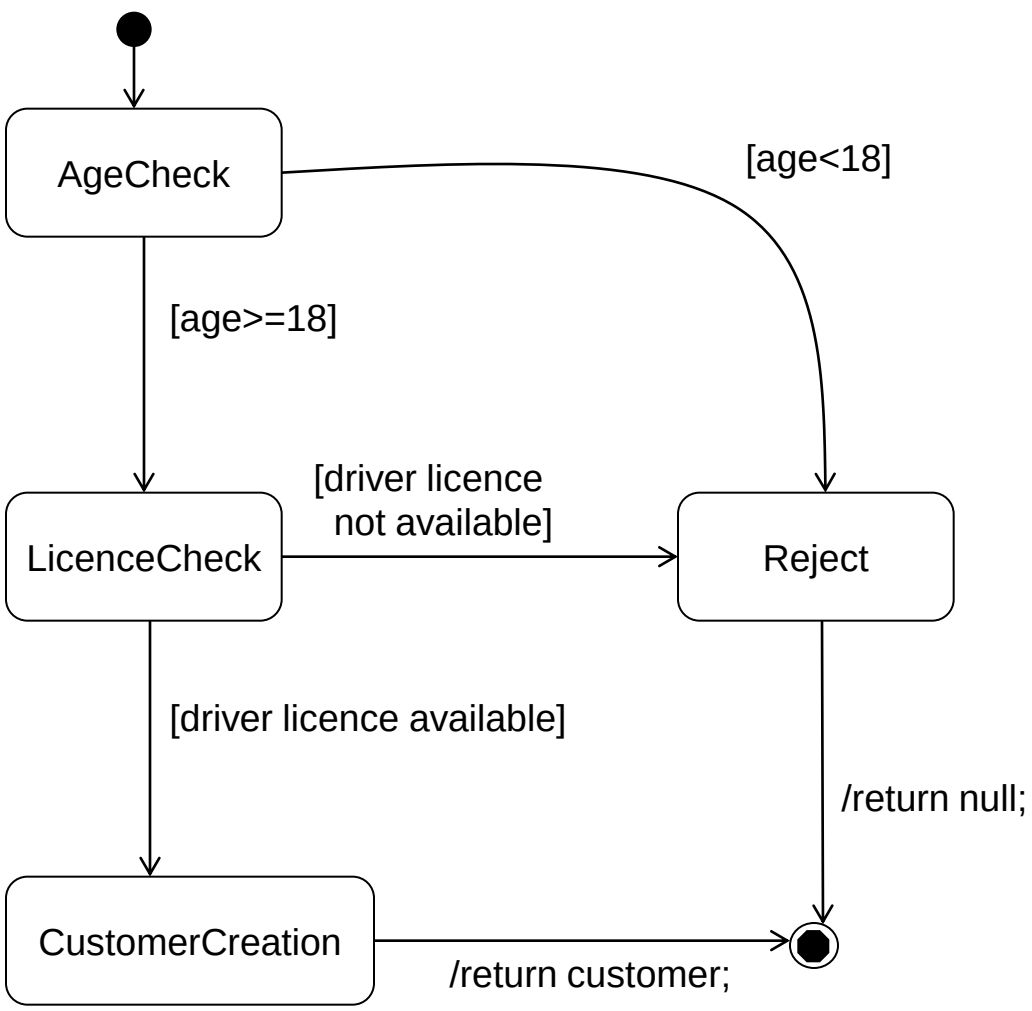
# Solution 7.1: Customer Service



# Exercise 7.2.1 – Method Statecharts

1. Model a statechart for the methods newCustomer(…), return(…) and rent(…). Consider thereby the following references:
   - newCustomer: In this method an examination of the age and the driving licence takes place.

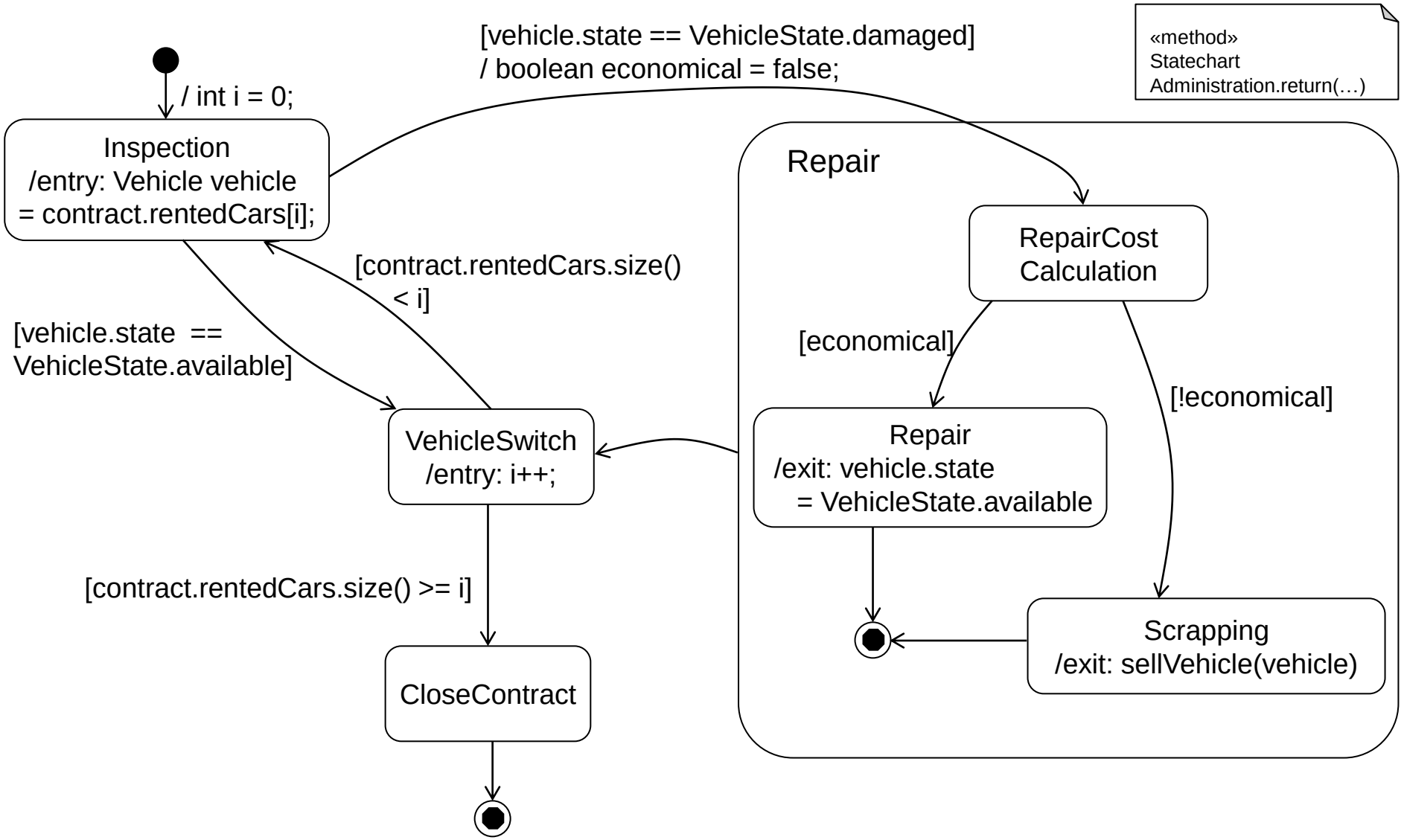
# Solution 7.2.1: newCustomer()



# Exercise 7.2.2 – Method Statecharts

2. Model a statechart for the return(…)-method.
   - In this method the vehicle is tested for damages.
   - In the case of damage, the vehicle will be repaired or discarded.

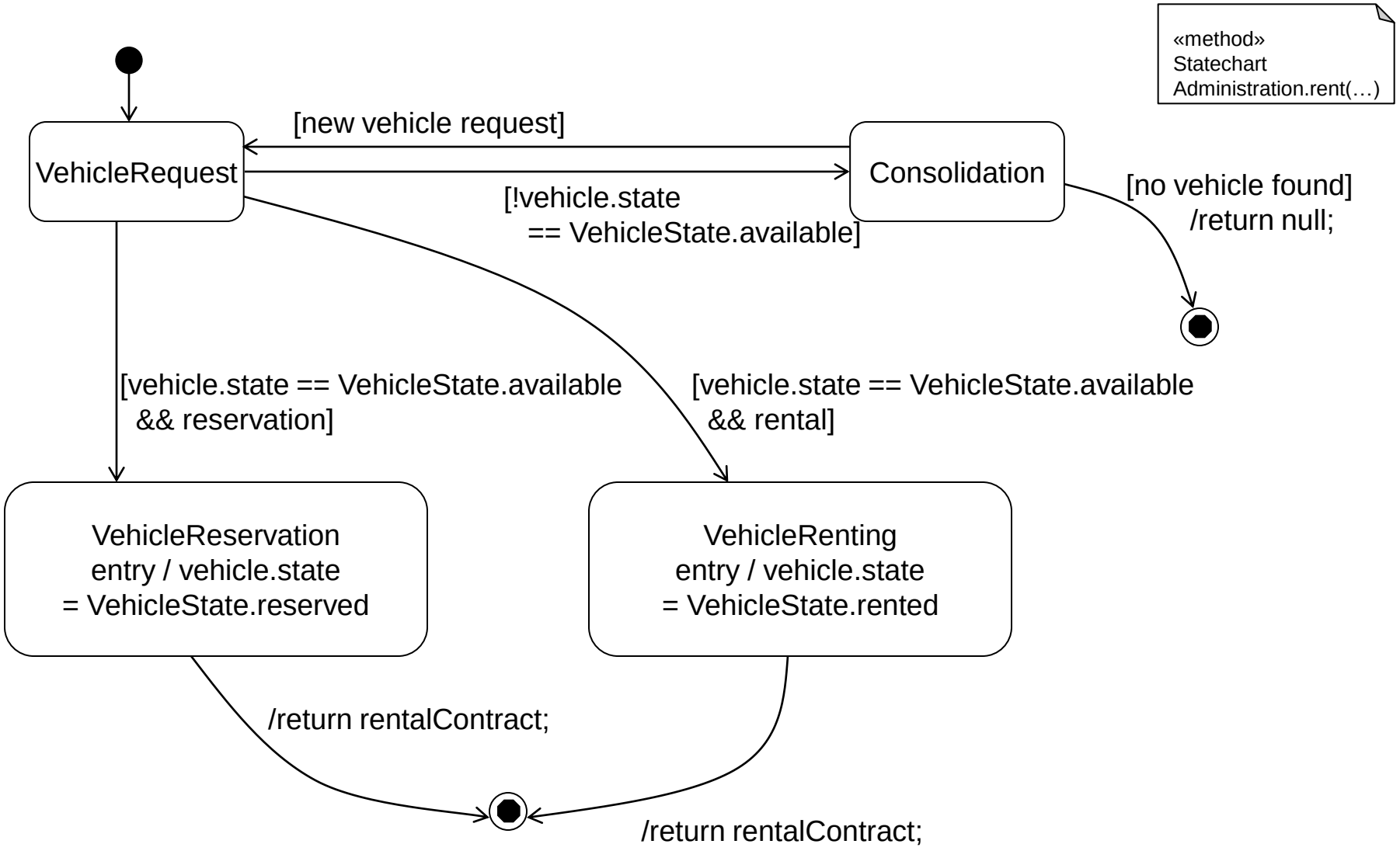
# Solution 7.2.2: return(RentalContract contract)



# Exercise 7.2.3 – Method Statecharts

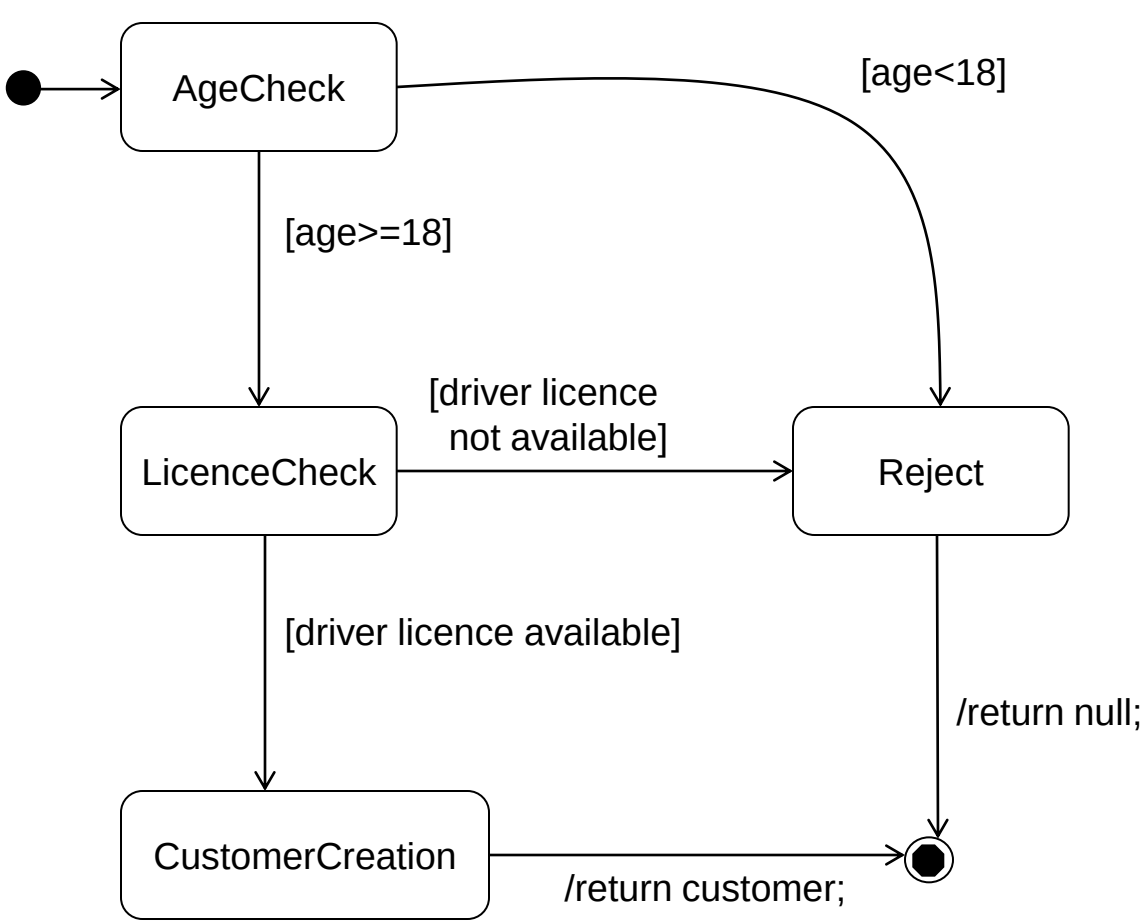
3. Model a statechart for the rent(…)-method.
   - On the basis of a customer's request, vehicles are reserved or rented.
   - If the desired vehicles are not available, a consolation takes place with the customer.

# Solution 7.2.3: rent(…)



# Exercise 7.3 – Code Generation

- Indicate now the generated code for the method newCustomer(). Use for it the not specified methods *getAge(Date date)* and *hasDrivingLicence(String name)*.



# Solution 7.3 – Part 1

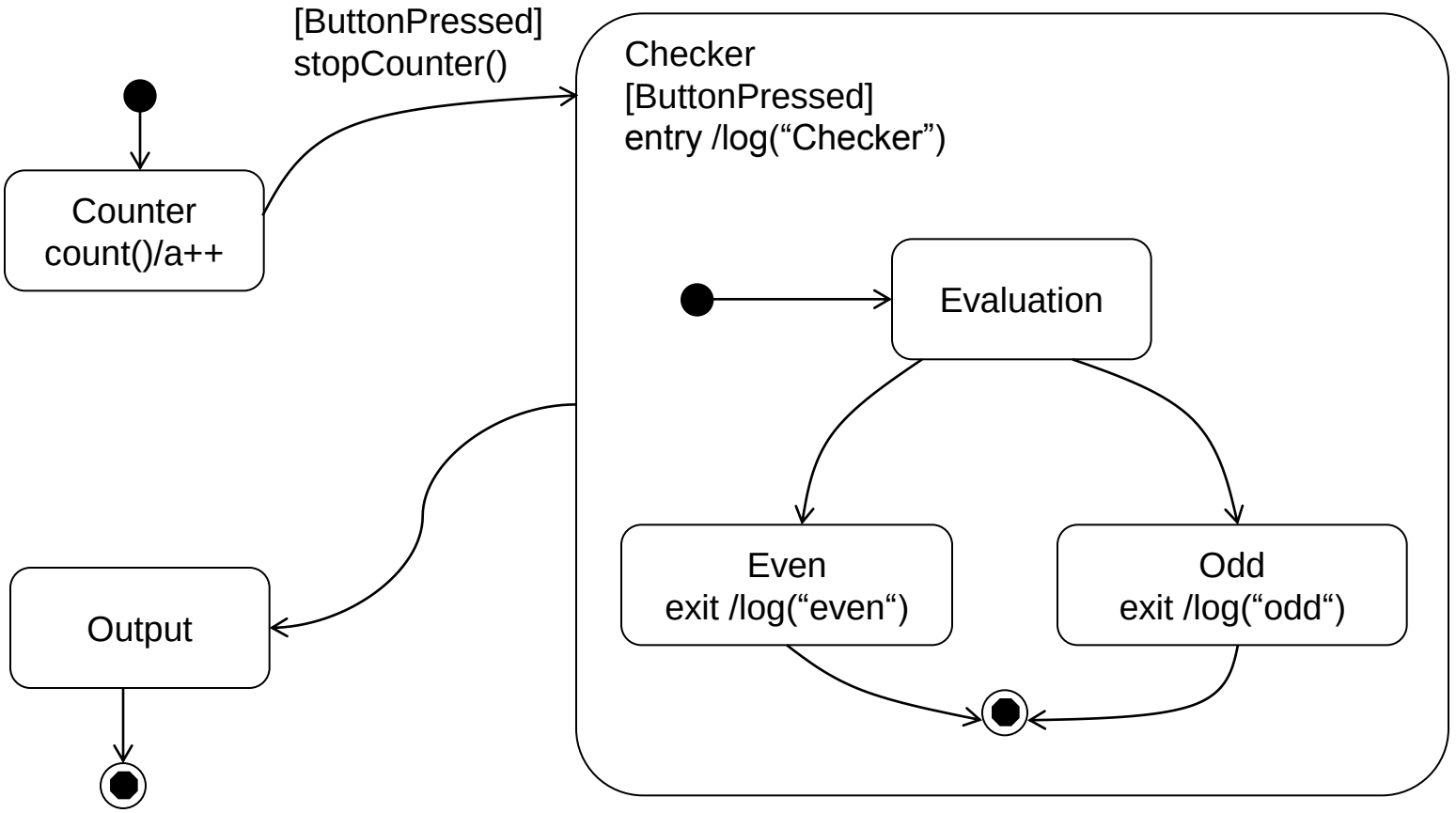
```
public class Administration {
  enum CustomerState {
    AGECHECK, LICENCECHECK, CUSTOMERCREATION, REJECT
  }

  public Customer newCustomer(String name, Address address,
                              Date birthday) {
    return intNewCustomer(name, address, birthday,
                          CustomerState.AGECHECK);
  }

  ...
```

# Solution 7.3 – Part 2

```
...
  protected Customer intNewCustomer(String name, Address address,
                                    Date birthday, CustomerState state) {
    switch(state) {
      case CustomerState.AGECHECK:
        if (getAge(birthday) >= 18)
          return intNewCustomer(…, CustomerState.LICENCECHECK);
        else if (getAge(birthday) < 18) {
          return intNewCustomer(…, CustomerState.REJECT);
        break;
      …
      case CustomerState.REJECT:
        return null;
        break;
      }
    return null;
  }
}
```

# Exercise 7.4 – Simplification of state charts

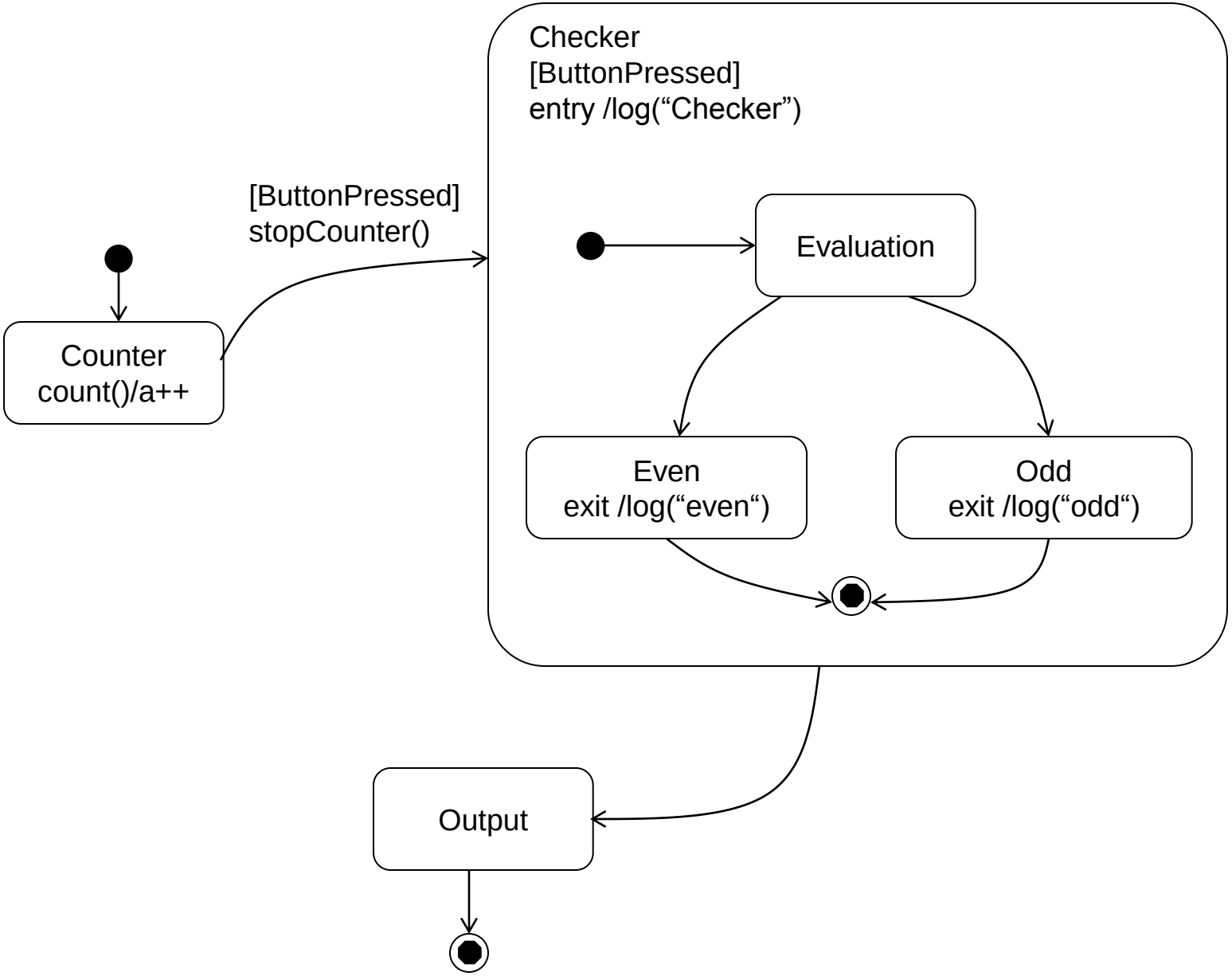
- Simplify the following state chart according to the algorithm of the lecture. Use only the steps 2-9.



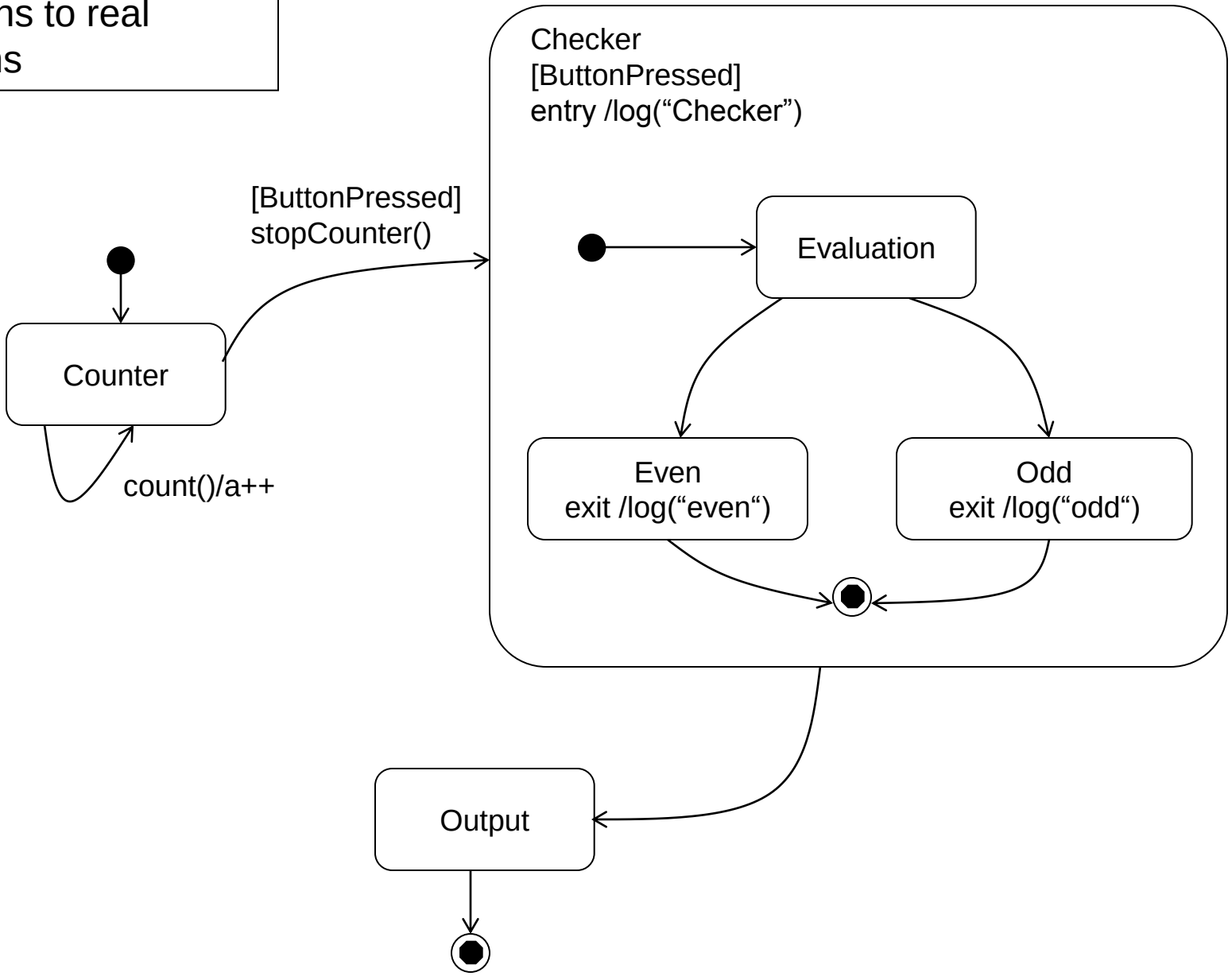
# Procedures to Simplify Statecharts: Steps 1-9: Remove Hierarchy

Subsequent steps were explained in the introduction of concepts:

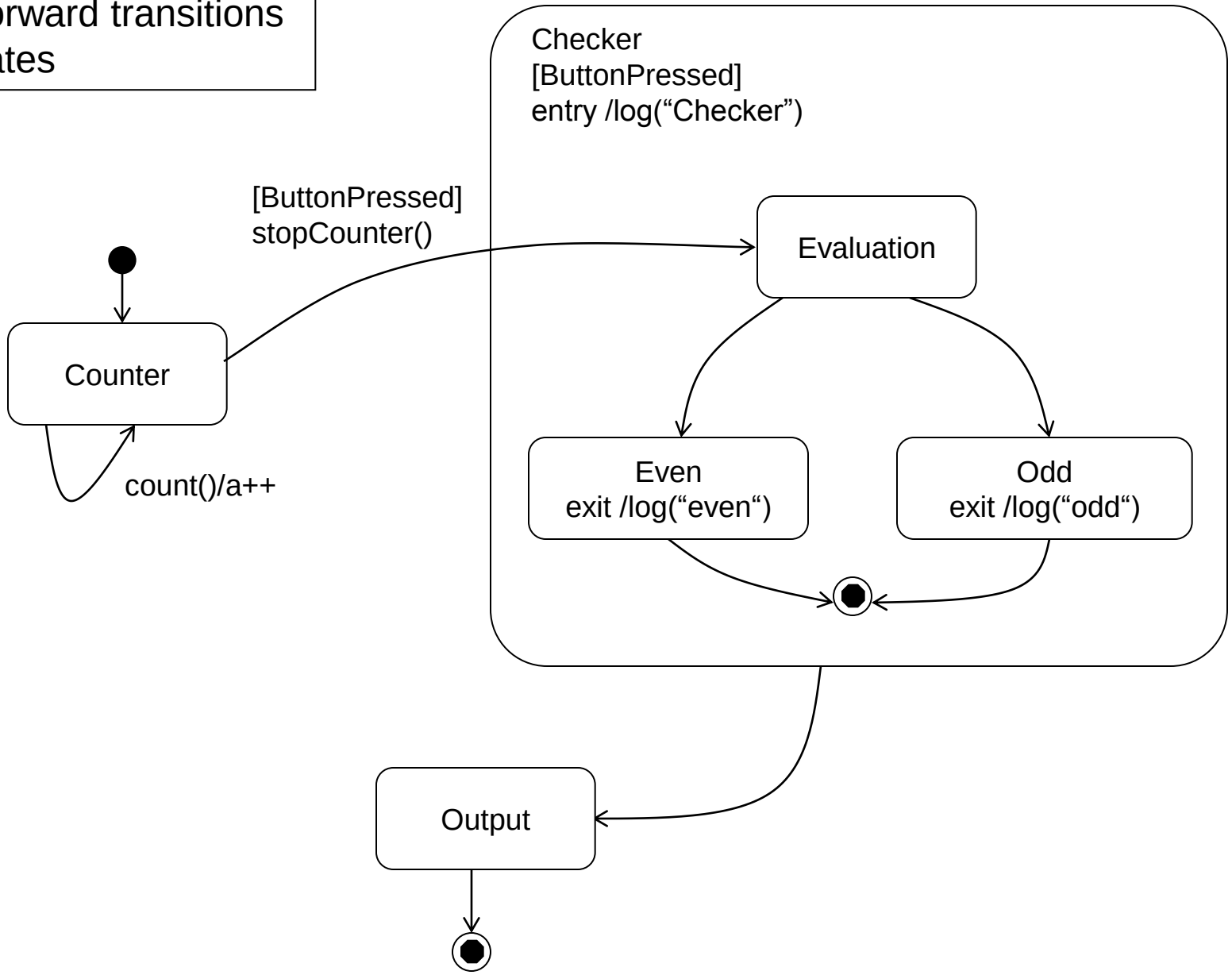
1. Eliminate Do-Activities
2. Transform internal Transitions to real transitions
3. Target states with sub-states: forward transitions to substates
4. Source state with sub-states: let analog transitions start from sub-states
5. Repeat 3.-4. at several levels of hierarchy till transitions have only atomic source and target states.
6. Add Exit-Actions of the action of each exiting transitions, and remove in the state.
7. Add Entry-Actions analogous to the incoming transitions
8. Include state invariants of super-states in the sub-state.
9. Remove hierarchically dissected states


Checker
[ButtonPressed]
entry /log(“Checker”)
[ButtonPressed]
stopCounter()
Counter
count()/a++
Evaluation
Even
exit /log(“even“)
Odd
exit /log(“odd“)
Output

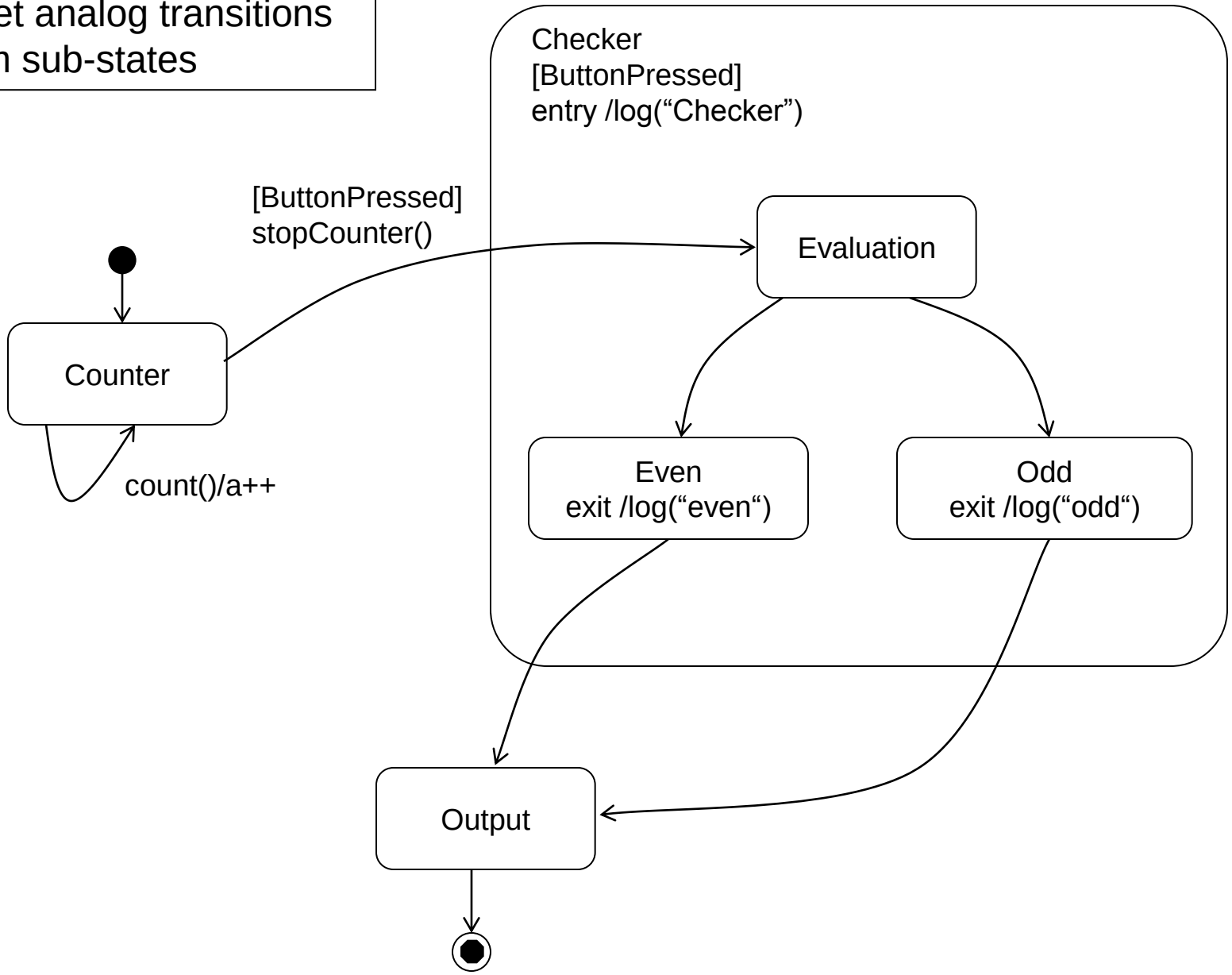
2. Transform internal Transitions to real transitions



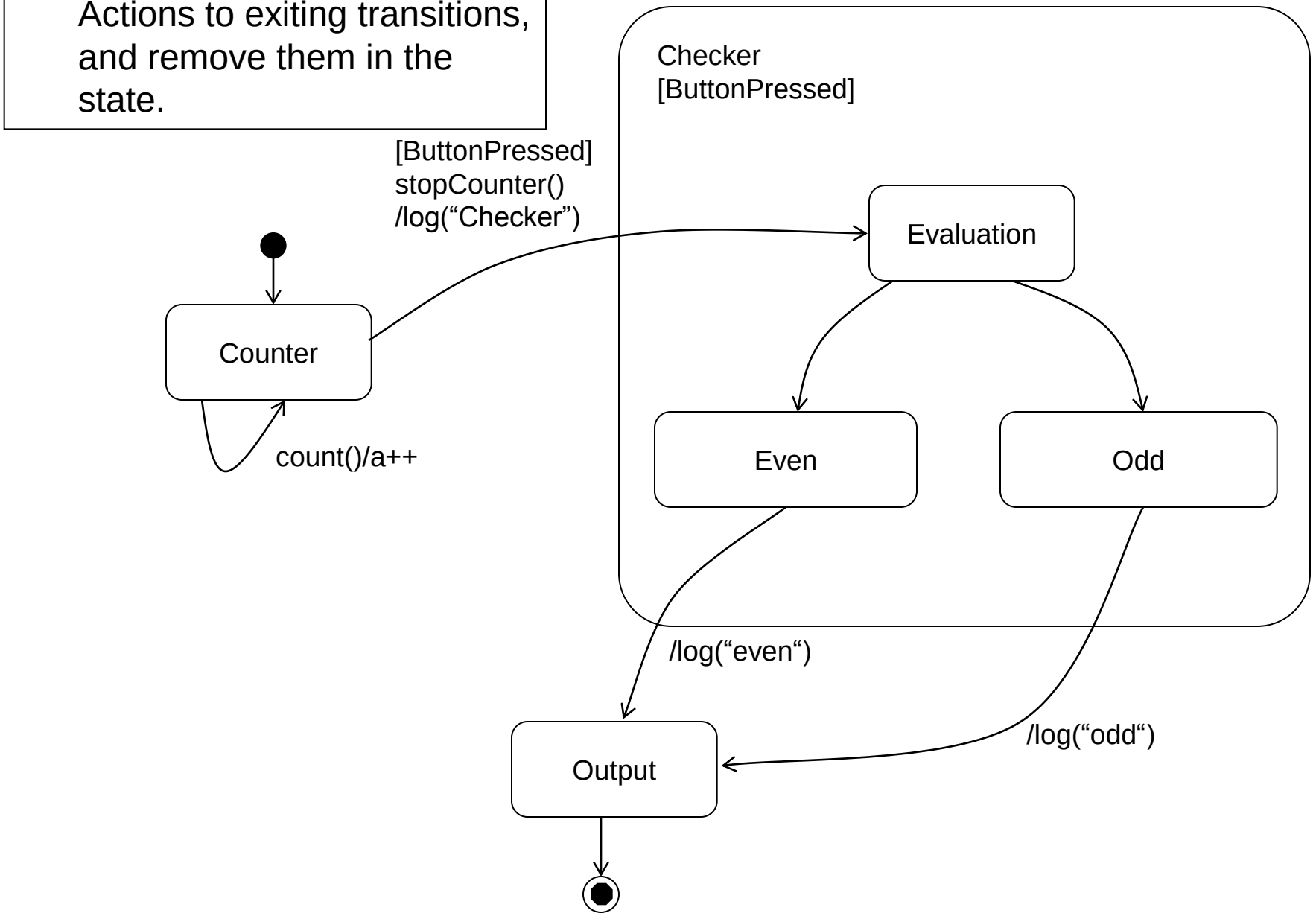
3. Target states with sub-states: forward transitions to substates



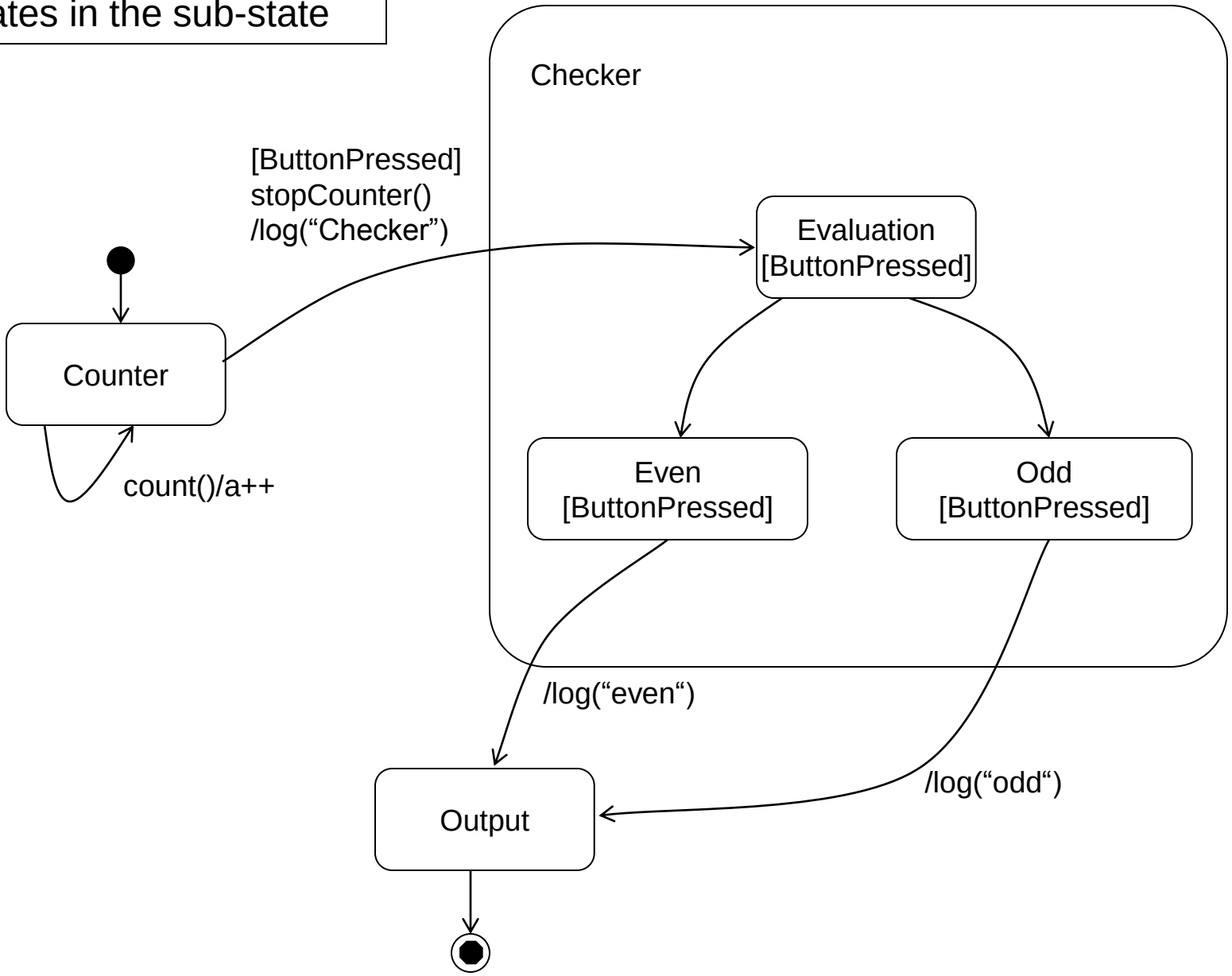
4. Source state with sub-states: let analog transitions start from sub-states



6. and 7.: Add Exit- and Entry-Actions to exiting transitions, and remove them in the state.



8. Include state invariants of super-states in the sub-state



# 9. Remove hierarchically dissected states