UML State Machines (Advanced)

Software Design (40007) – 2023/2024

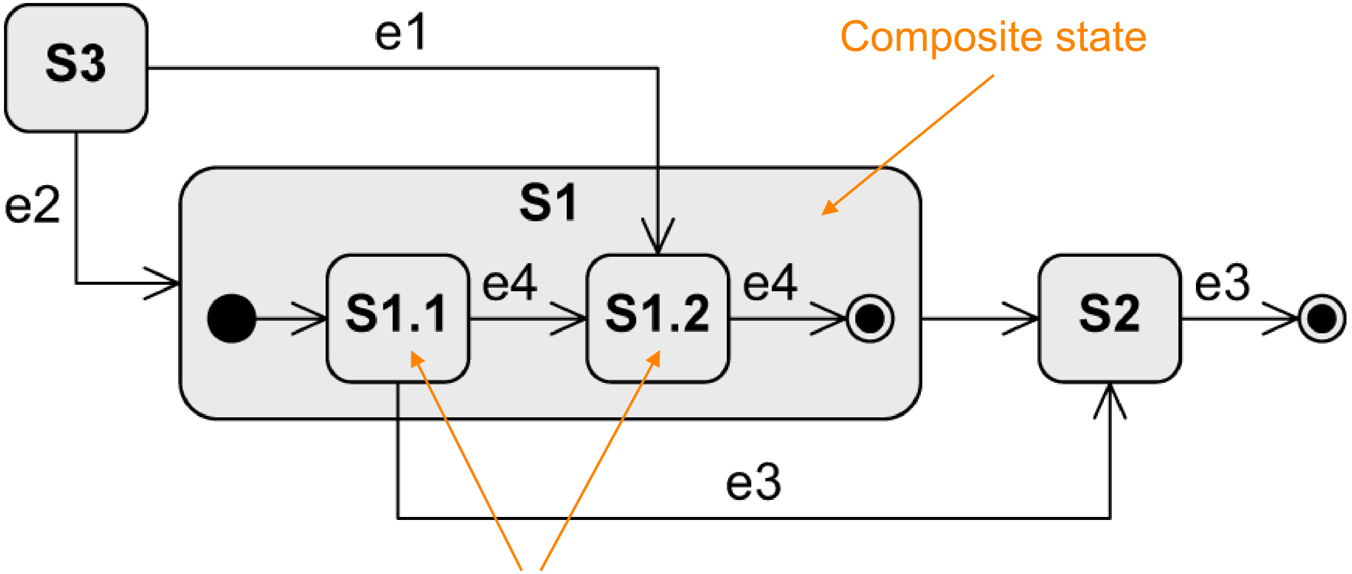
Justus Bogner
Ivano Malavolta

VRIJE UNIVERSITEIT AMSTERDAM

Software and Sustainability research group (S2)
Department of Computer Science, Faculty of Sciences

Composite state

- Contains other states called “substates“
  - Only one of its substates is active at any point in time
- Arbitrary nesting depth of substates allowed (but be careful!)



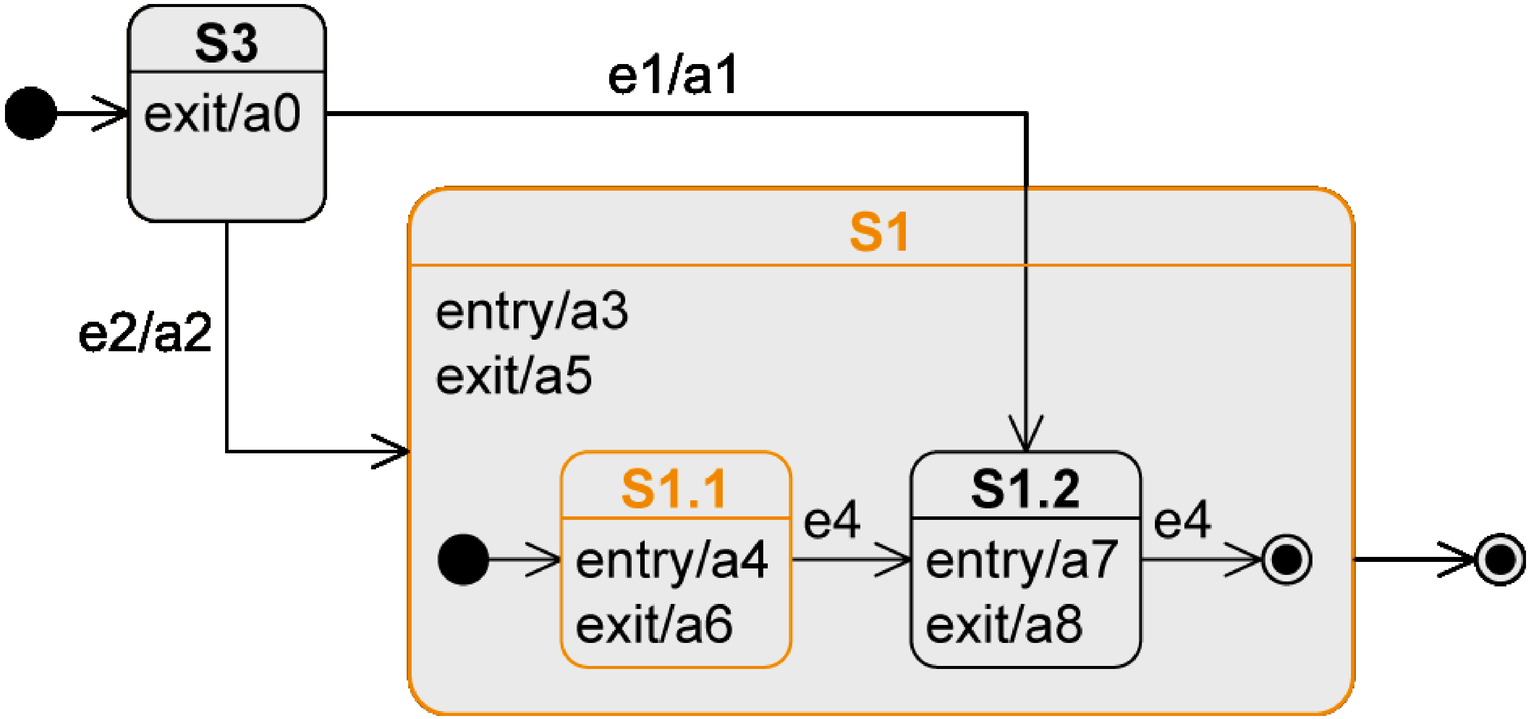
Composite state

Substates

Entering a composite state (1/2)

- Transition to the boundary
    - Initial node of composite state is activated

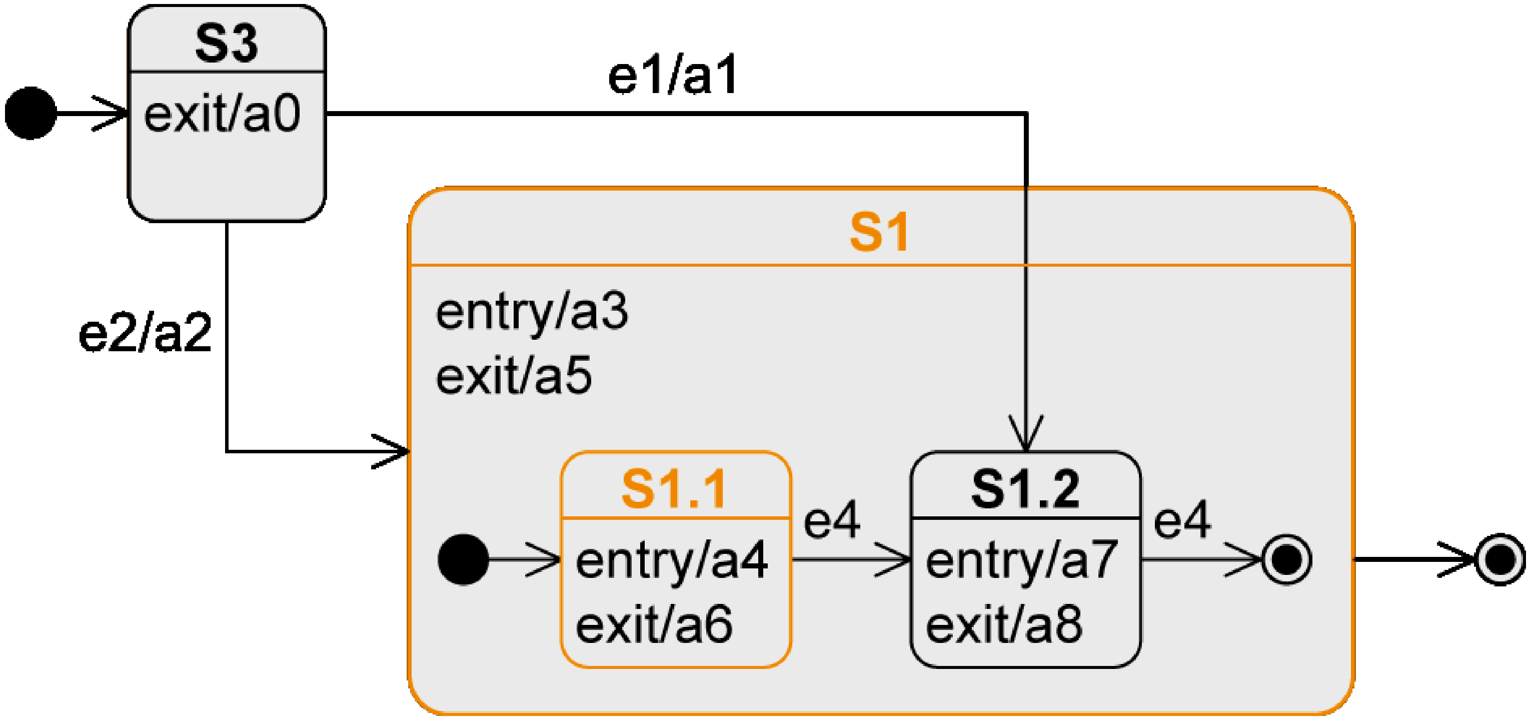
Event	State	Executed Activities
„Beginning“	S3	
e2	S1/S1.1	



Entering a composite state (1/2)

- Transition to the boundary
  - Initial node of composite state is activated

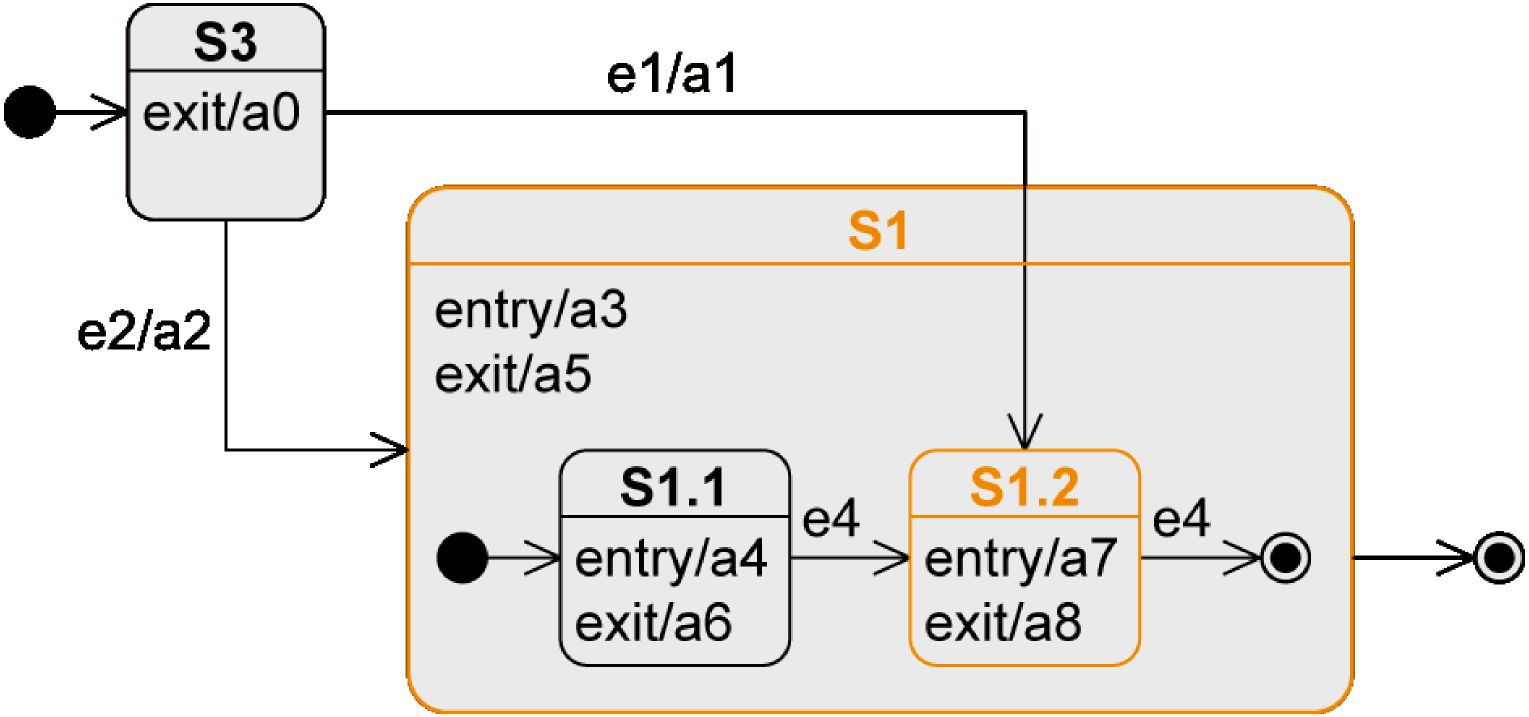
Event	State	Executed Activities
„Beginning“	S3	
e2	S1/S1.1	a0-a2-a3-a4



Entering a composite state (2/2)

- Transition to a substate
  - Substate is activated

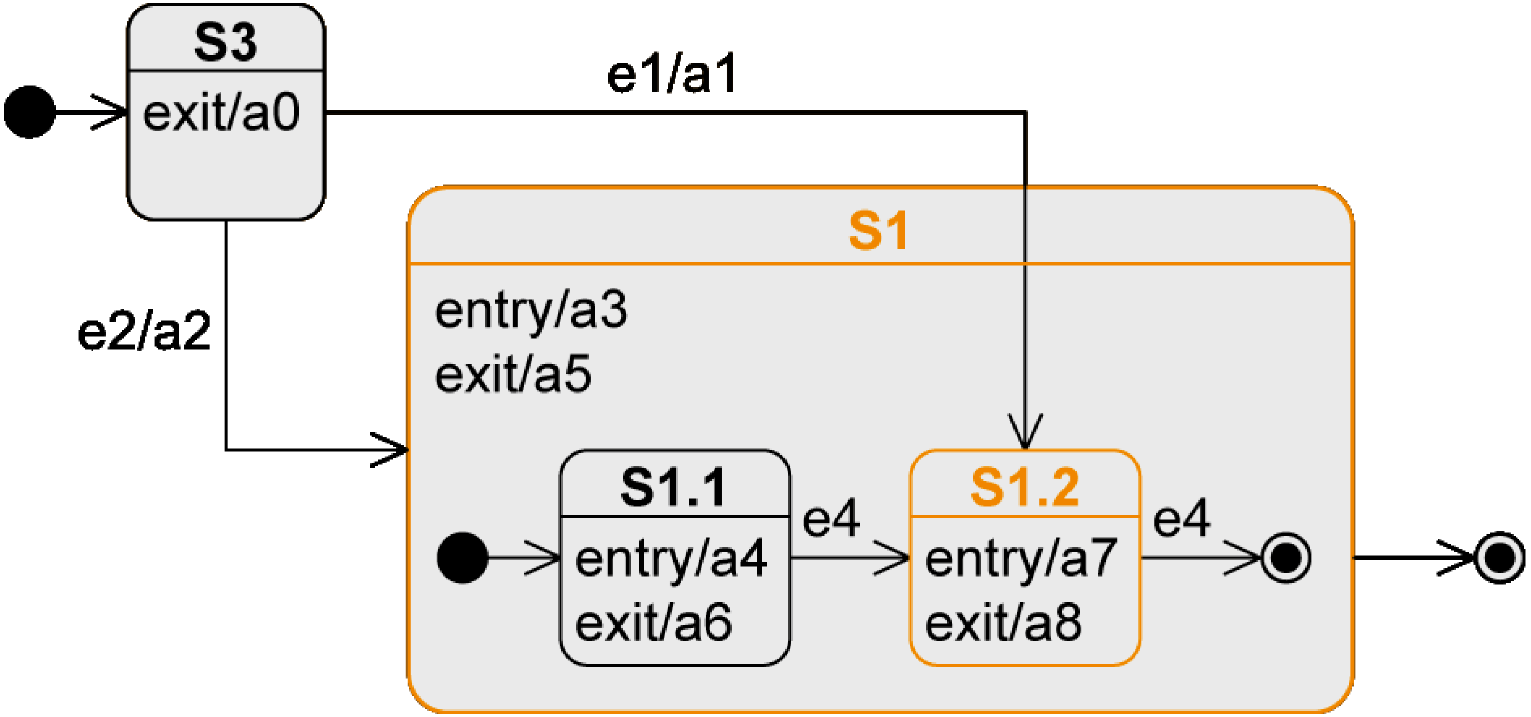
Event	State	Executed Activities
„Beginning“	S3	
e1	S1/S1.2	



Entering a composite state (2/2)

- Transition to a substate
  - Substate is activated

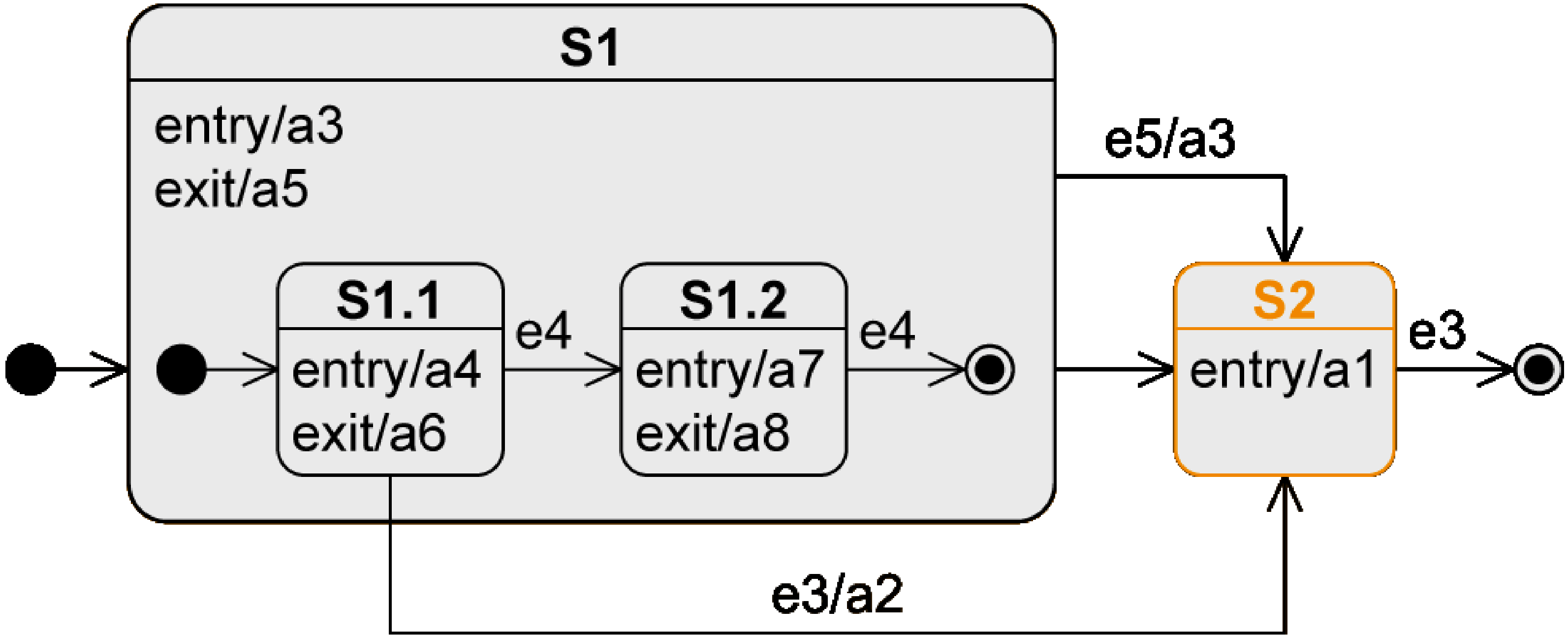
Event	State	Executed Activities
„Beginning“	S3	
e1	S1/S1.2	a0-a1-a3-a7



Exiting from a composite state (1/3)

- Transition from a substate

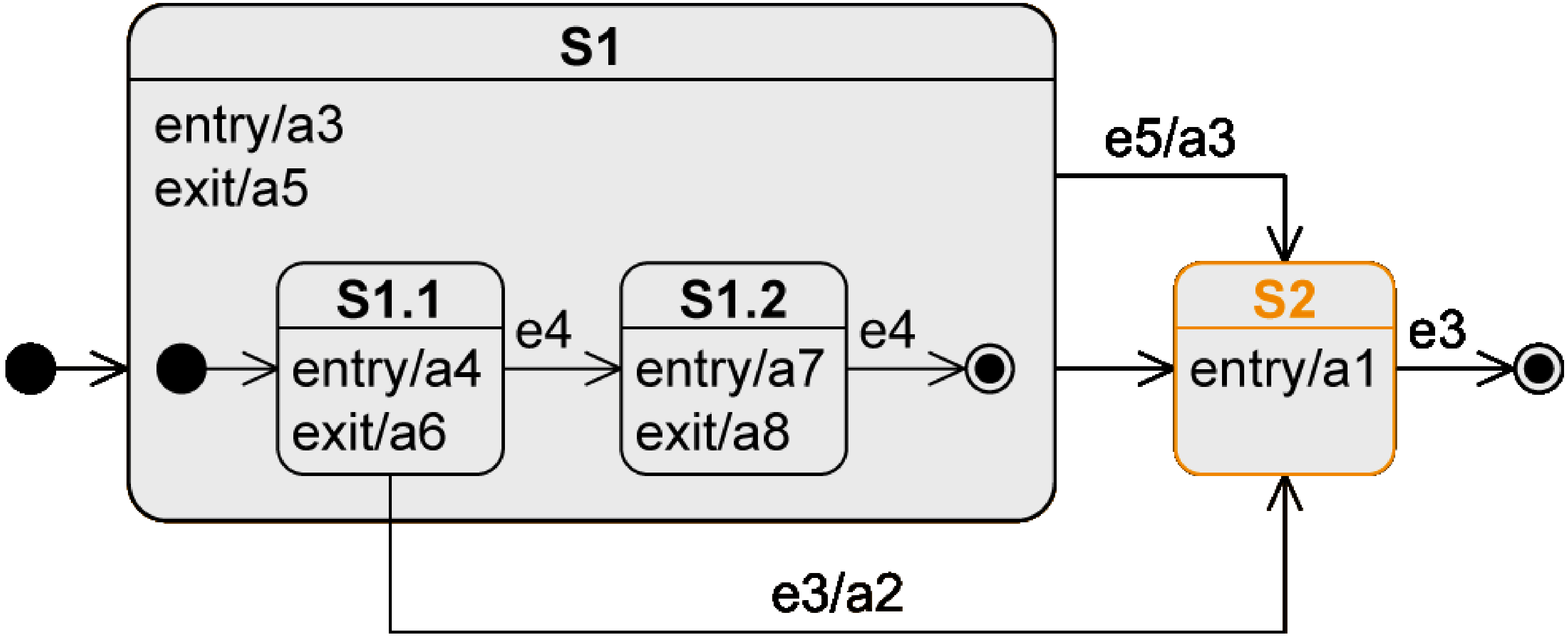
Event	State	Executed Activities
„Beginning“	…	…
e3	…	…



Exiting from a composite state (1/3)

- Transition from a substate

Event	State	Executed Activities
„Beginning“	S1/S1.1	a3-a4
e3	S2	a6-a5-a2-a1

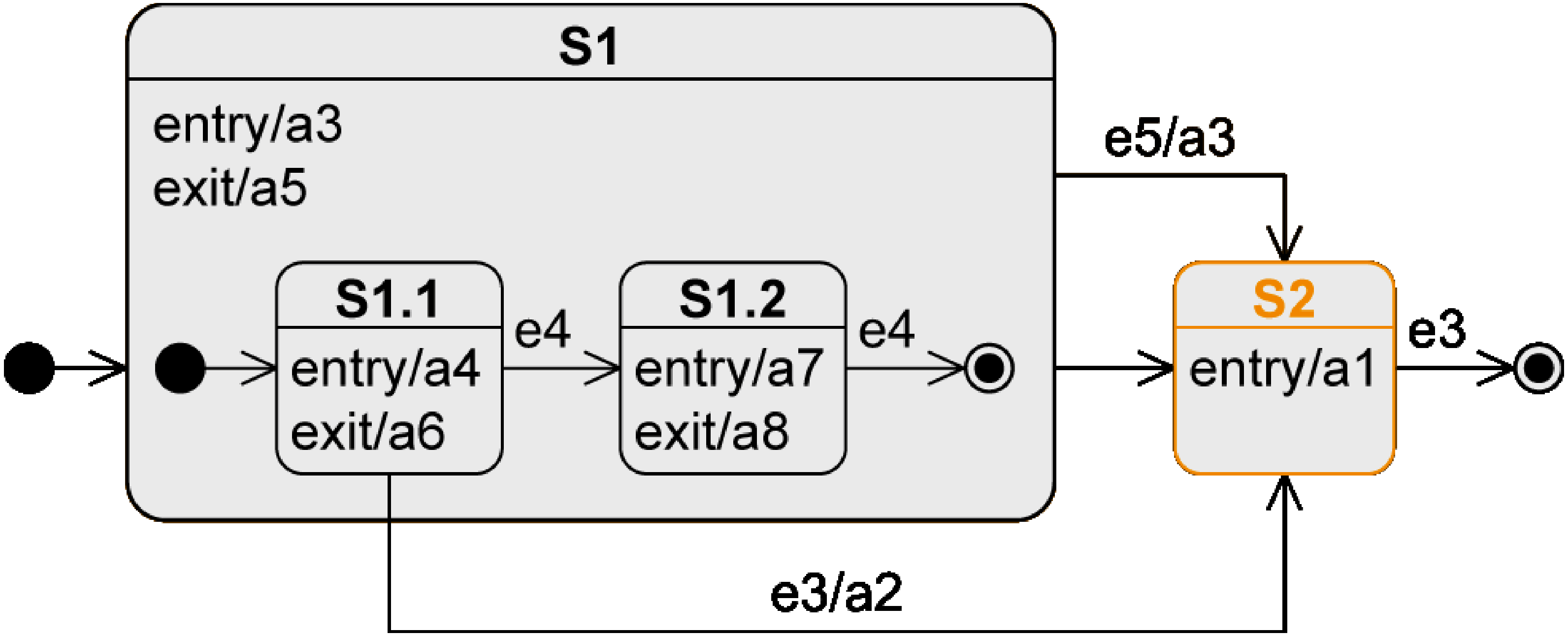


Exiting from a composite state (2/3)

- Transition from the composite state

No matter which substate of S1 is active, as soon as e5 occurs, the system changes to S2

Event	State	Executed Activities
„Beginning“	…	…
e5	…	…

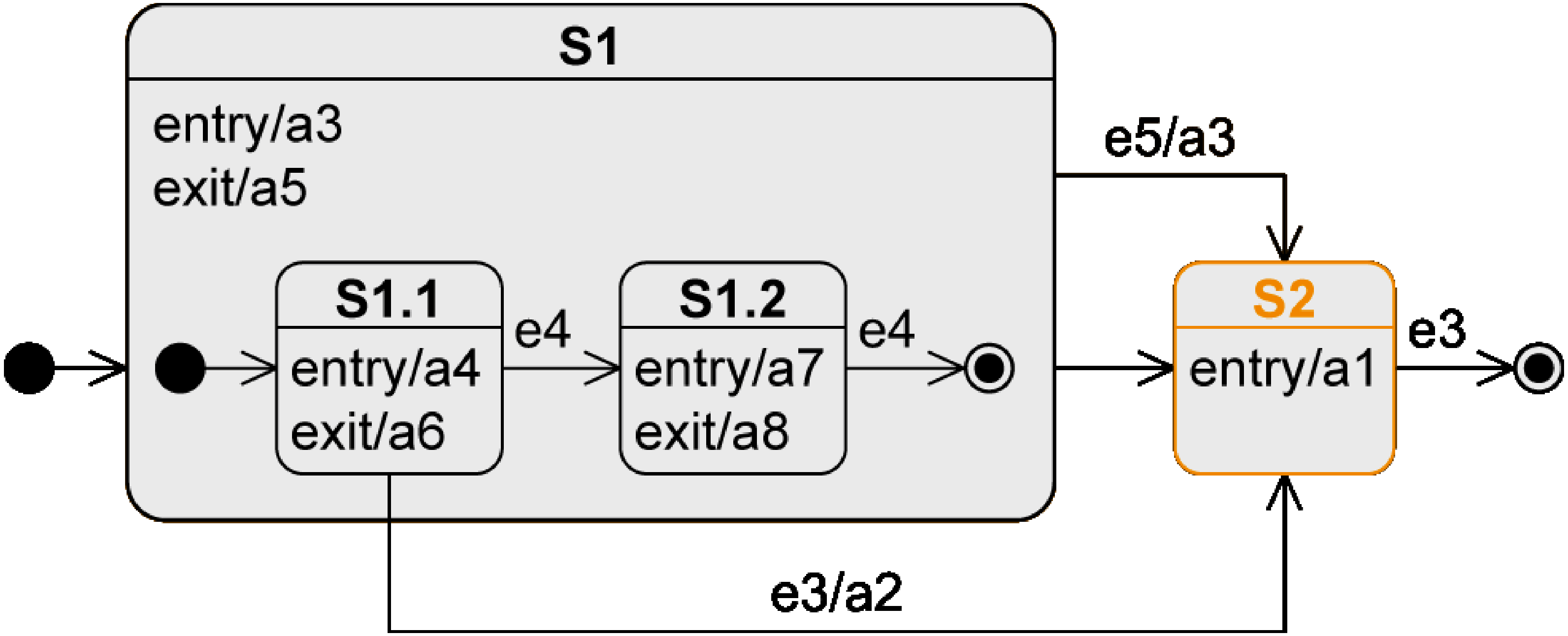


Exiting from a composite state (2/3)

- Transition from the composite state

No matter which substate of S1 is active, as soon as e5 occurs, the system changes to S2

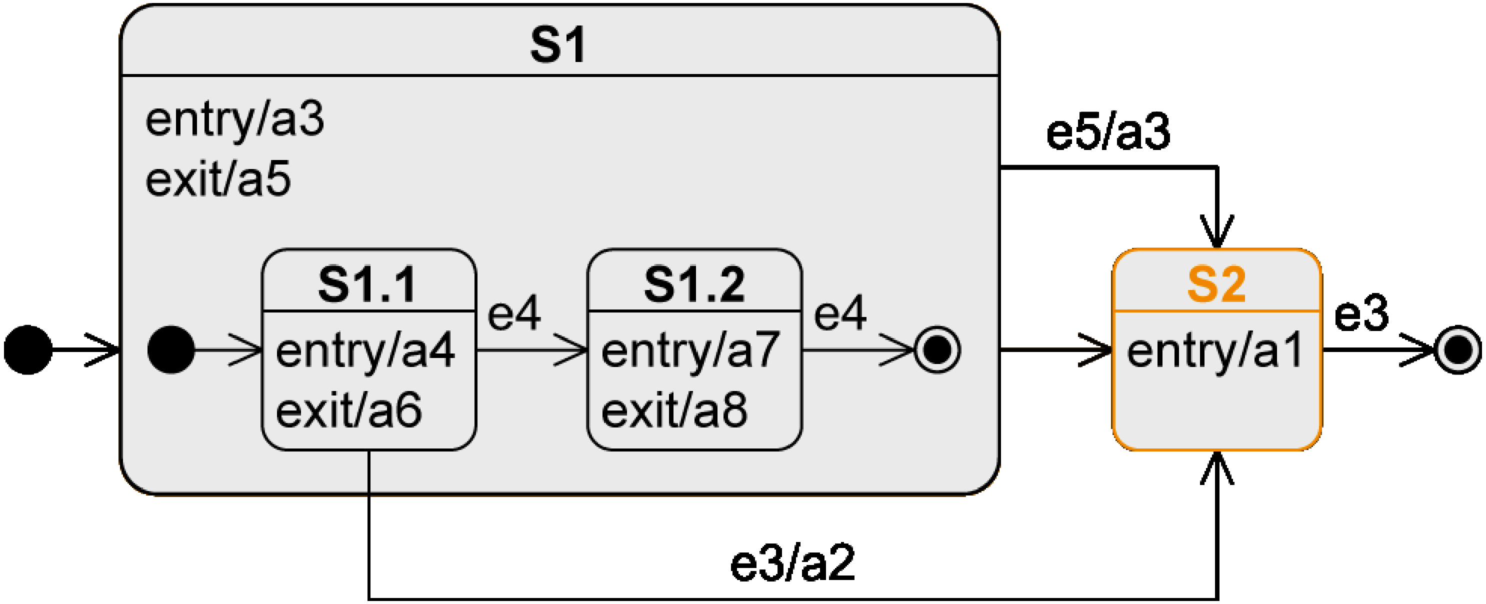
Event	State	Executed Activities
„Beginning“	S1/S1.1	a3-a4
e5	S2	a6-a5-a3-a1



Exiting from a composite state (3/3)

- Completion transition from the composite state

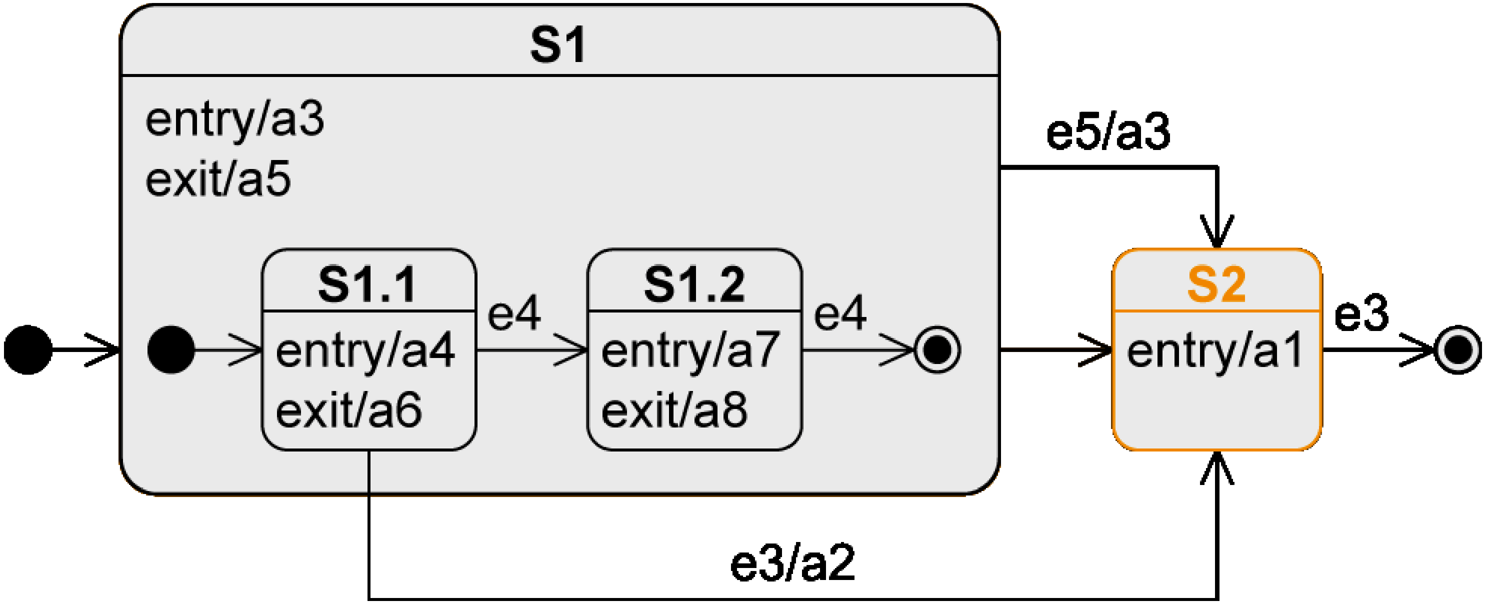
Event	State	Executed Activities
„Beginning“	…	…
e4	…	…
e4	…	…



Exiting from a composite state (3/3)

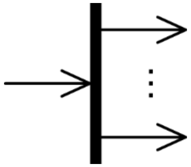
- Completion transition from the composite state

Event	State	Executed Activities
„Beginning“	S1/S1.1	a3-a4
e4	S1/S1.2	a6-a7
e4	S2	a8-a5-a1



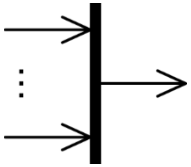
Parallelization and synchronization node

Parallelization node



- Pseudo state
- **Splits** the control flow into multiple concurrent flows
- 1 incoming edge
- At least 2 outgoing edges

Synchronization node

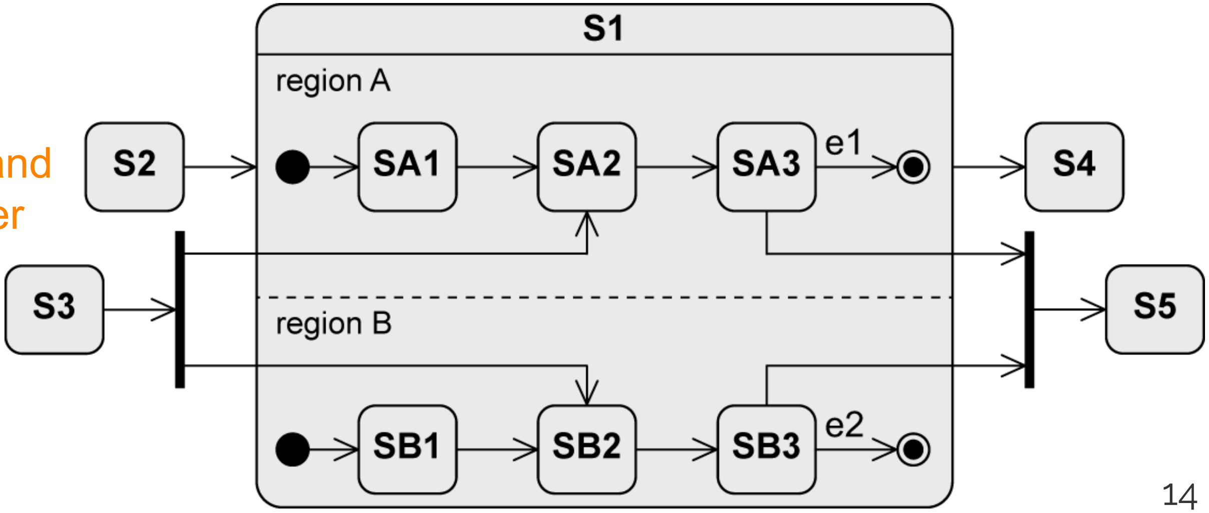


- Pseudo state
- **Merges** multiple concurrent flows
- At least 2 incoming edges
- 1 outgoing edge

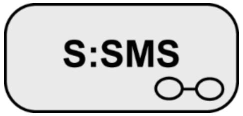
Orthogonal state

- Composite state is divided into two or more **regions** separated by a dashed line
- One state of each region is always active at any point in time, i.e., **concurrent substates**
- Entry: transition to the boundary of the orthogonal state activates the initial states of all regions
- Exit: final state must be reached in all regions to trigger completion event

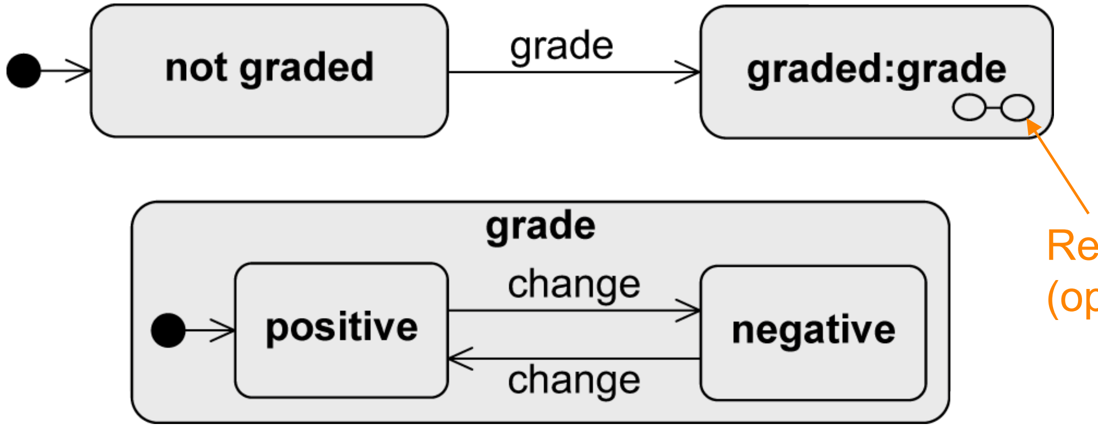
You can use parallelization and synchronization node to enter different substates



Submachine state (SMS)



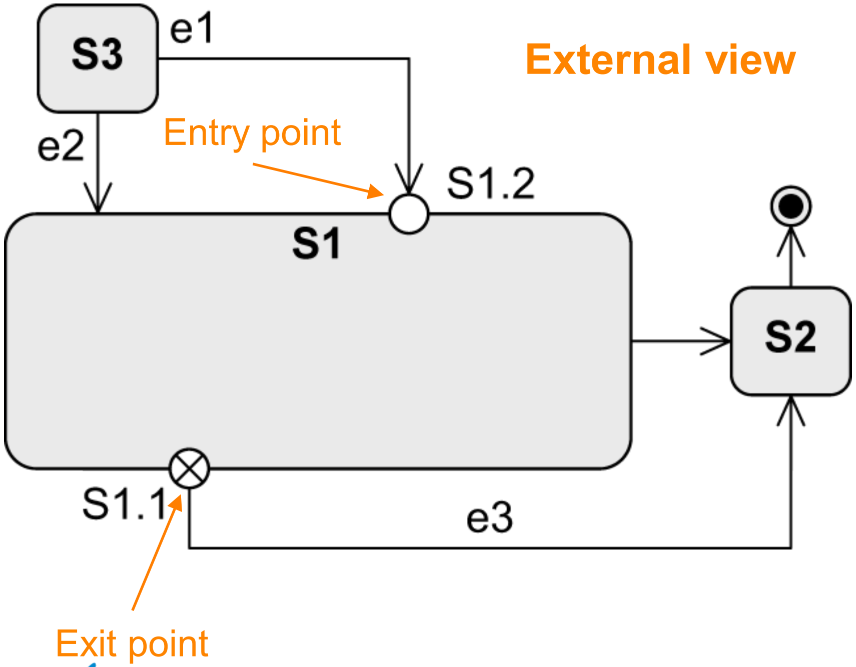
- Helpful for reusing parts of state machine diagrams in other state machine diagrams
- Notation: **`stateName:submachineStateName`**
- As soon as the submachine state is activated, the behavior of the submachine is executed
- Similar to calling a black-box method in Java



Refinement symbol (optional)

Entry and exit points

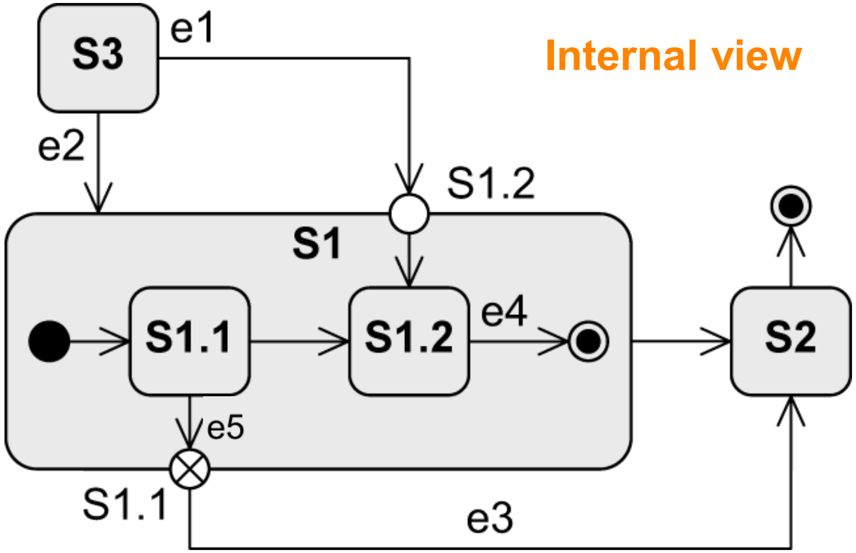
- Encapsulation mechanism
  - A composite state shall be entered or exited via a state other than the initial and final states
  - The external transition **does not know** the structure of the composite state



External view

Entry point

Exit point



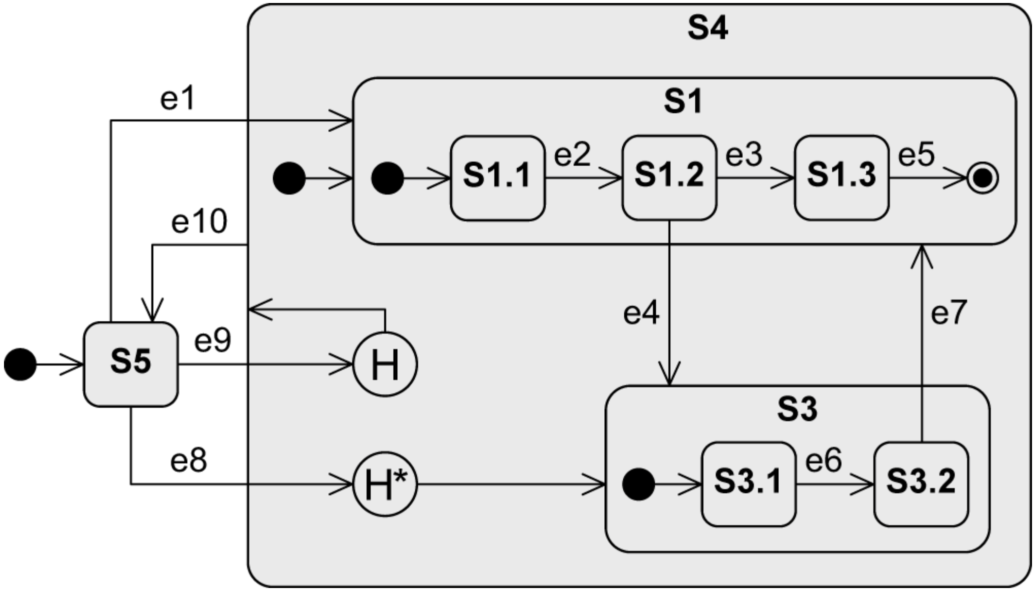
Internal view

History state

- Remembers which substate of a composite state was the **last active** one
- Activates the “old” substate and all entry activities are conducted sequentially from the outside to the inside of the composite state
- Exactly one outgoing edge of the history state points to a substate. It is used if:
  - the composite state was never active before

    OR
  - the composite state was exited via the final state
- Shallow vs. deep history state

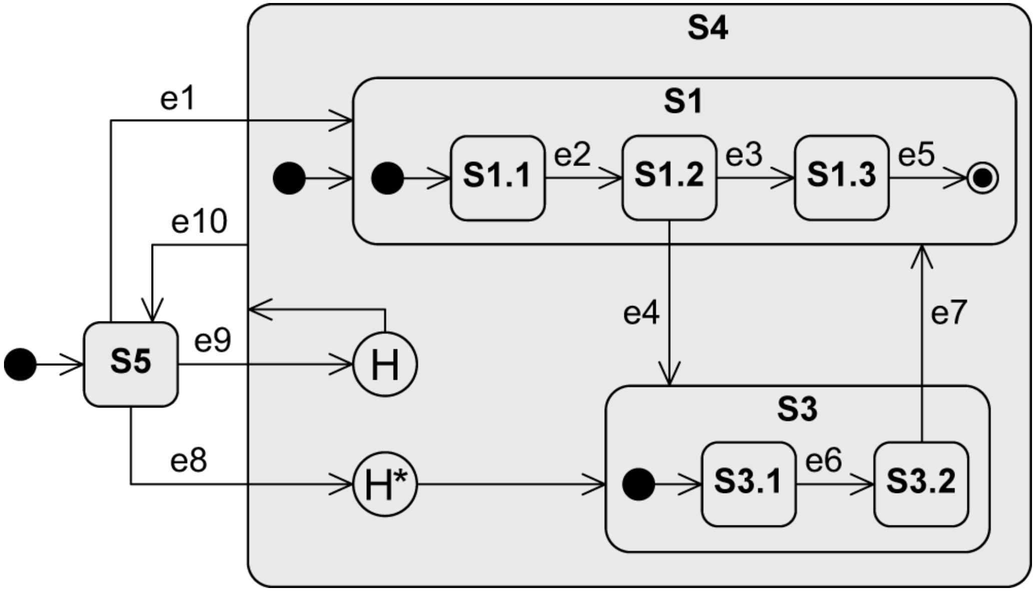
Example: history state (1/4)



Event	State
„Beginning“	S5
e1	S4/S1/S1.1
e2	S1.2
e10	S5
e9	(H→) S1/S1.1

(H) **Shallow history** state restores the last active state that is at the same level as the history state

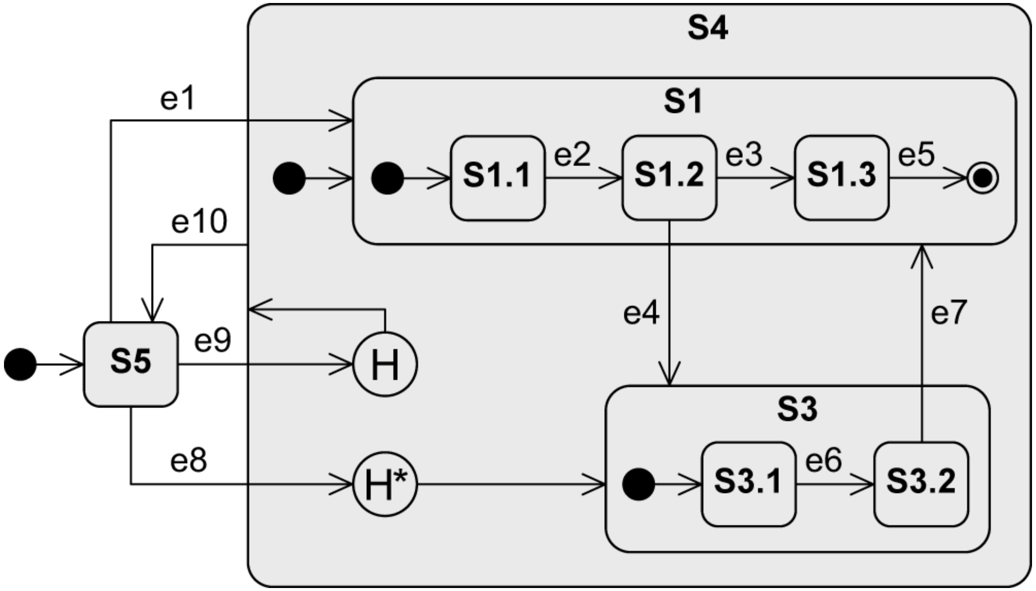
Example: history state (2/4)



Event	State
„Beginning“	S5
e1	S4/S1/S1.1
e2	S1.2
e10	S5
e8	(H*→) S1/S1.2

H* **Deep history** state restores the last active substate over the entire nesting depth

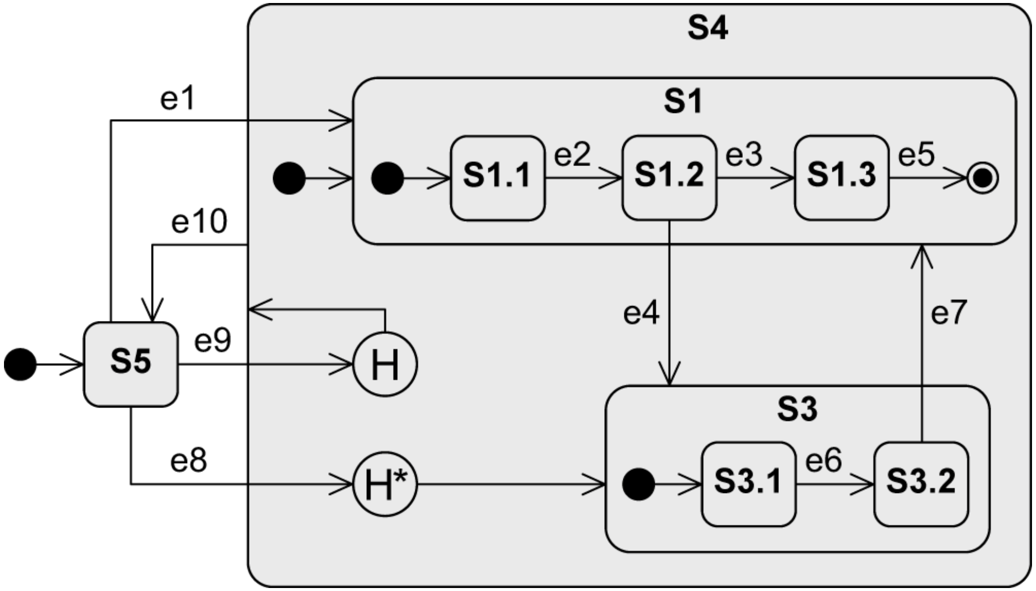
Example: history state (3/4)



Event	State
„Beginning“	S5
e9	(H→) S4/S1/S1.1

If no history exists, follow the outgoing transition.

Example: history state (4/4)



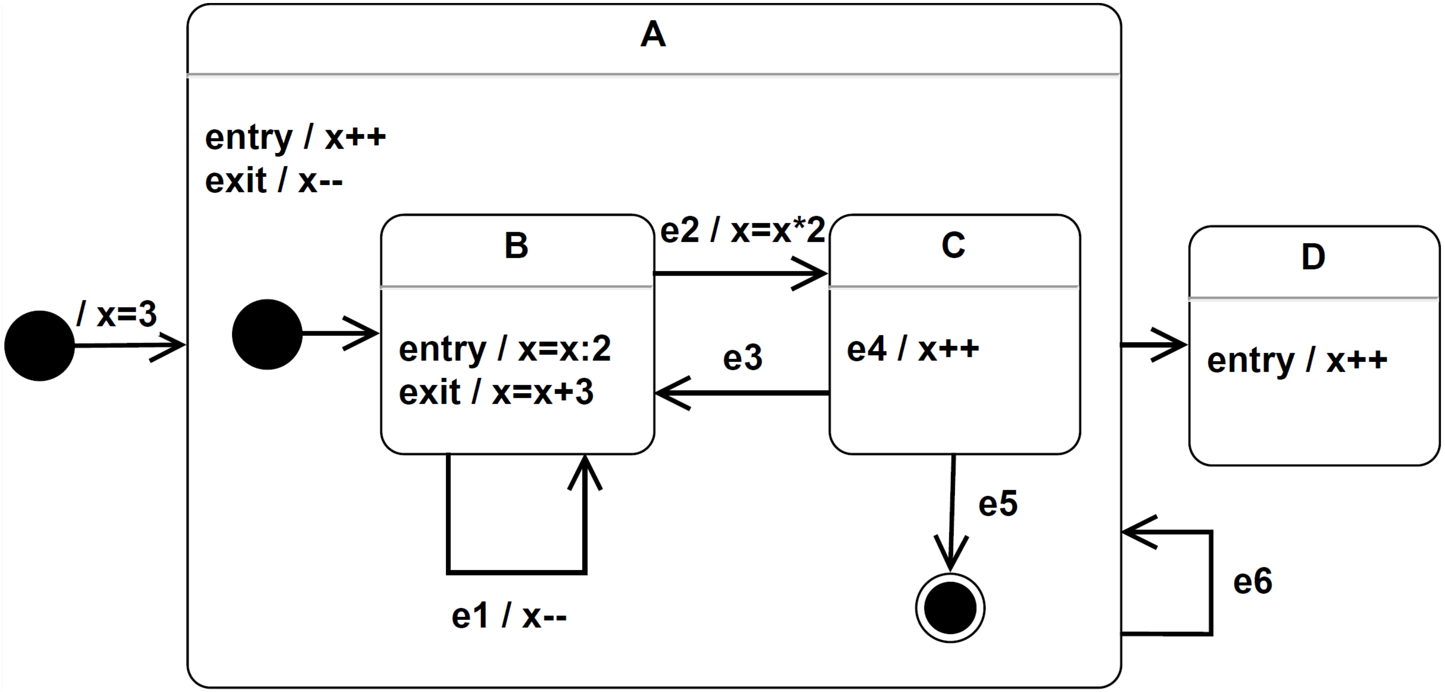
Event	State
„Beginning“	S5
e8	(H*→) S3/S3.1

If no history exists, follow the outgoing transition.

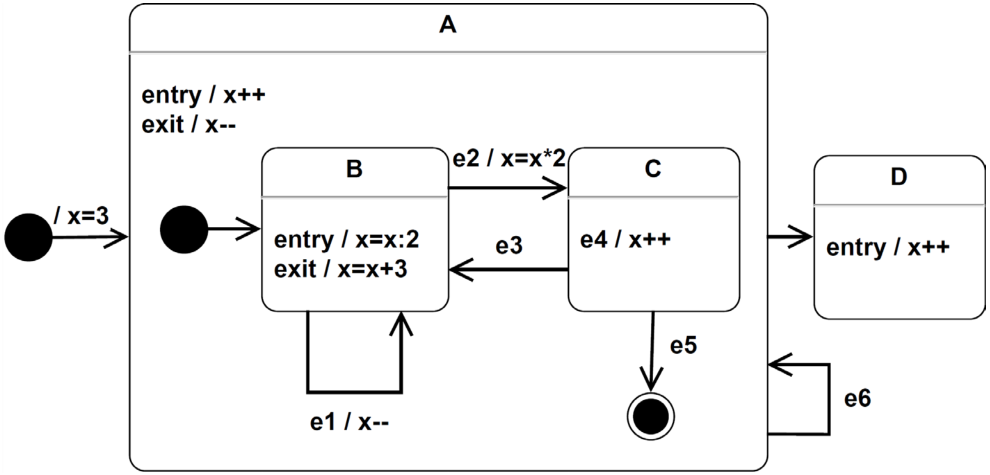
Exercise

Event sequence: e1, e2, e4, e4, e3, e1

What is the final value of x?

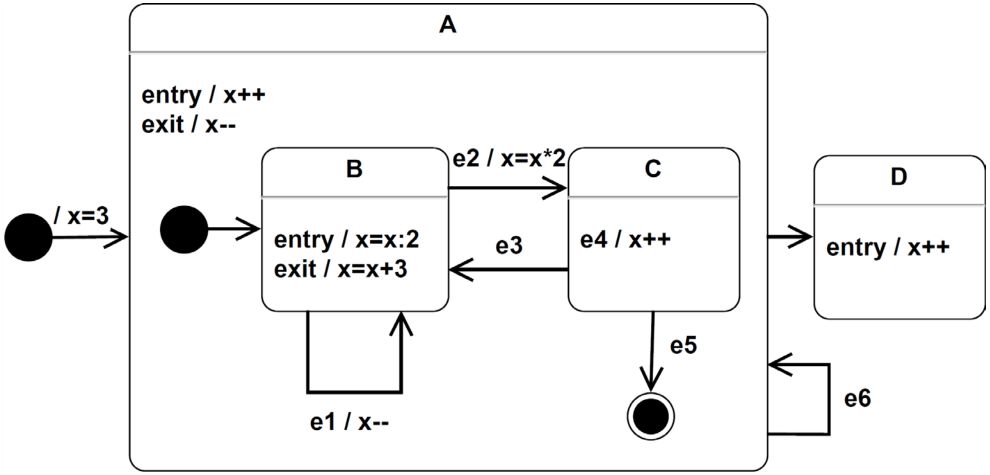


Solution



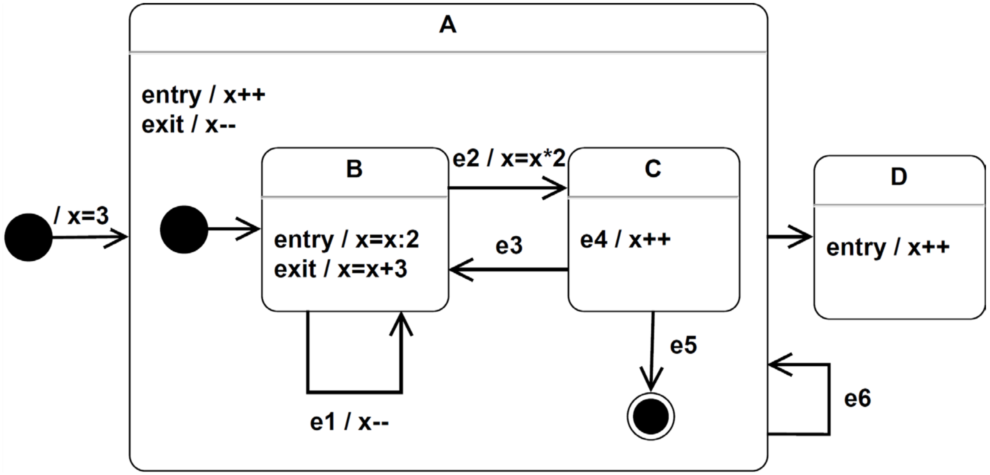
Event	State	Comment	x
		Start	3
e1			
e2			
e4			
e4			
e3			
e1			

Solution



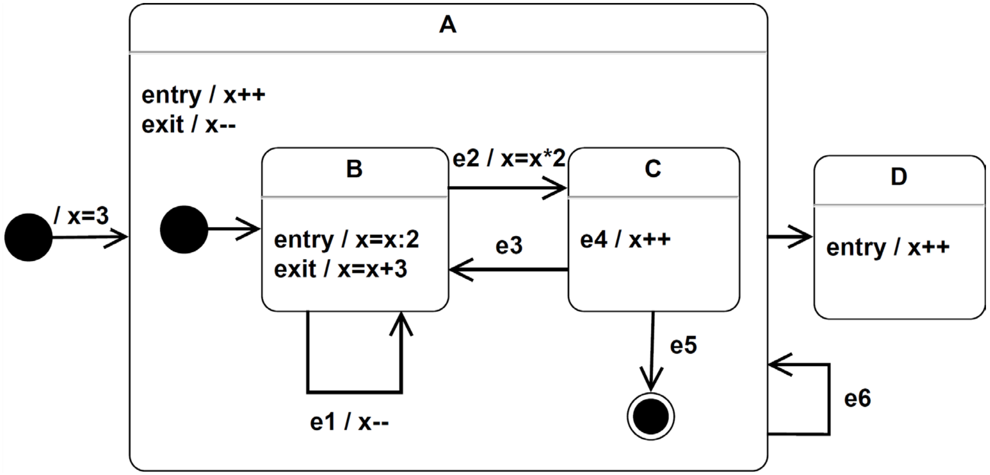
Event	State	Comment	x
		Start	3
	A	Entry of A	4
e1			
e2			
e4			
e4			
e3			
e1			

Solution



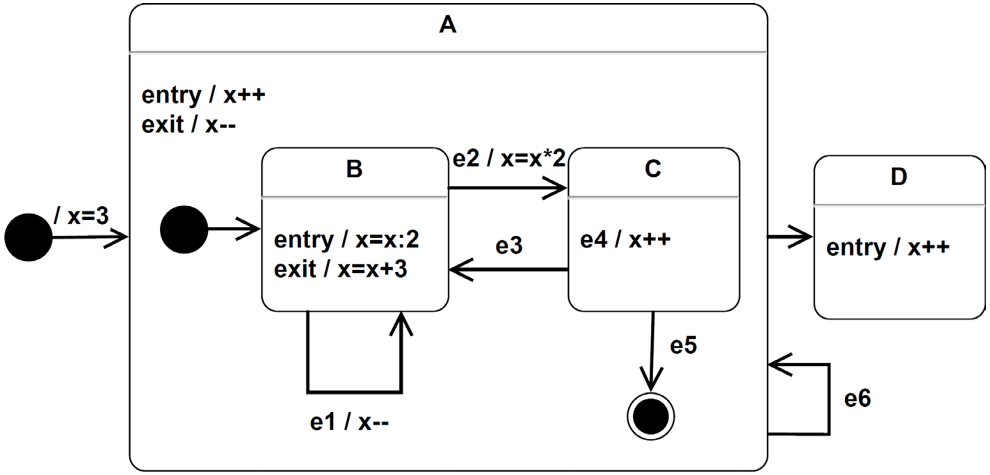
Event	State	Comment	x
		Start	3
	A	Entry of A	4
	A/B	Entry of B	2
e1			
e2			
e4			
e4			
e3			
e1			

Solution



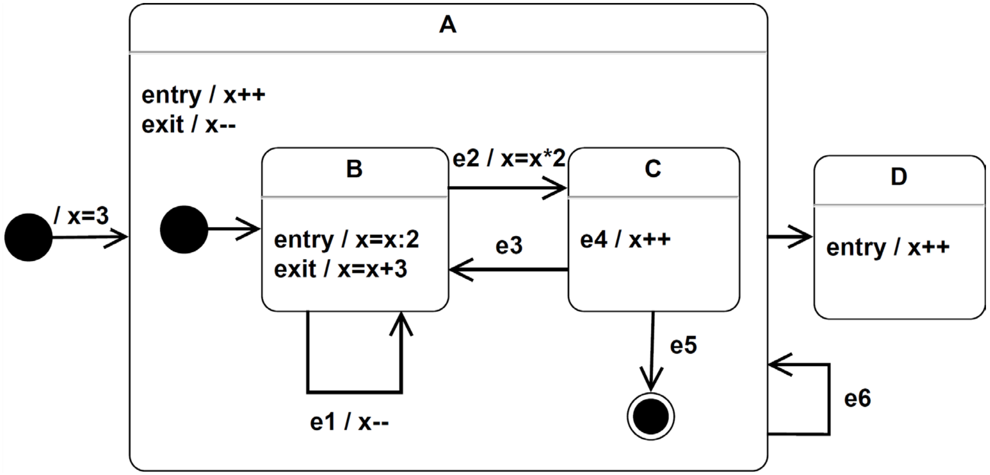
Event	State	Comment	x
		Start	3
	A	Entry of A	4
	A/B	Entry of B	2
e1	A/B	Exit of B x-- (self-transition) Entry of B	5 4 2
e2			
e4			
e4			
e3			
e1			

Solution



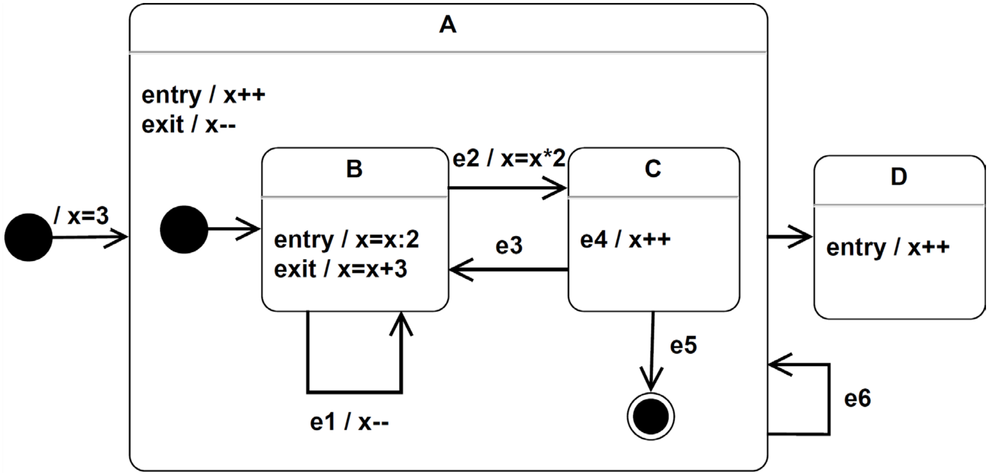
Event	State	Comment	x
		Start	3
	A	Entry of A	4
	A/B	Entry of B	2
e1	A/B	Exit of B x-- (self-transition) Entry of B	5 4 2
e2	A/C	Exit of B x=x*2	5 10
e4			
e4			
e3			
e1			

Solution



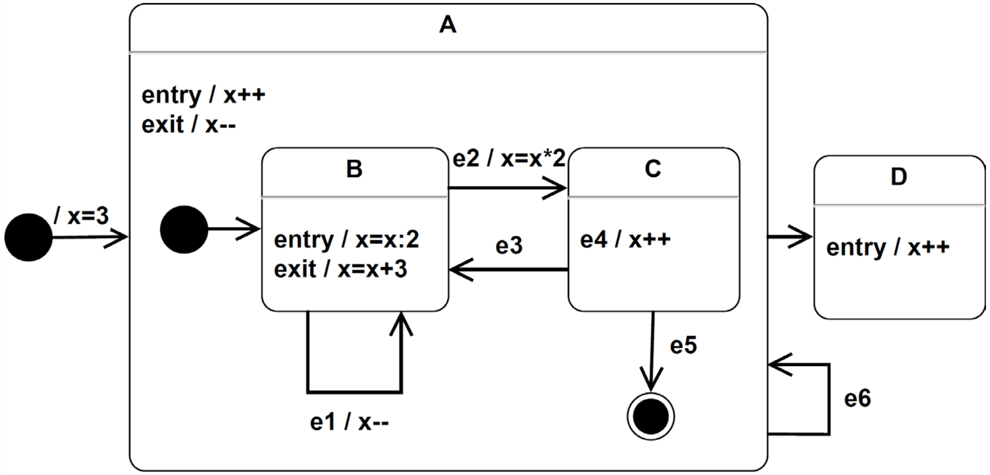
Event	State	Comment	x
		Start	3
	A	Entry of A	4
	A/B	Entry of B	2
e1	A/B	Exit of B x-- (self-transition) Entry of B	5 4 2
e2	A/C	Exit of B x=x*2	5 10
e4		x++ in C	11
e4		x++ in C	12
e3			
e1			

Solution



Event	State	Comment	x
		Start	3
	A	Entry of A	4
	A/B	Entry of B	2
e1	A/B	Exit of B x-- (self-transition) Entry of B	5 4 2
e2	A/C	Exit of B x=x*2	5 10
e4		x++ in C	11
e4		x++ in C	12
e3	A/B	Entry of B	6
e1			

Solution

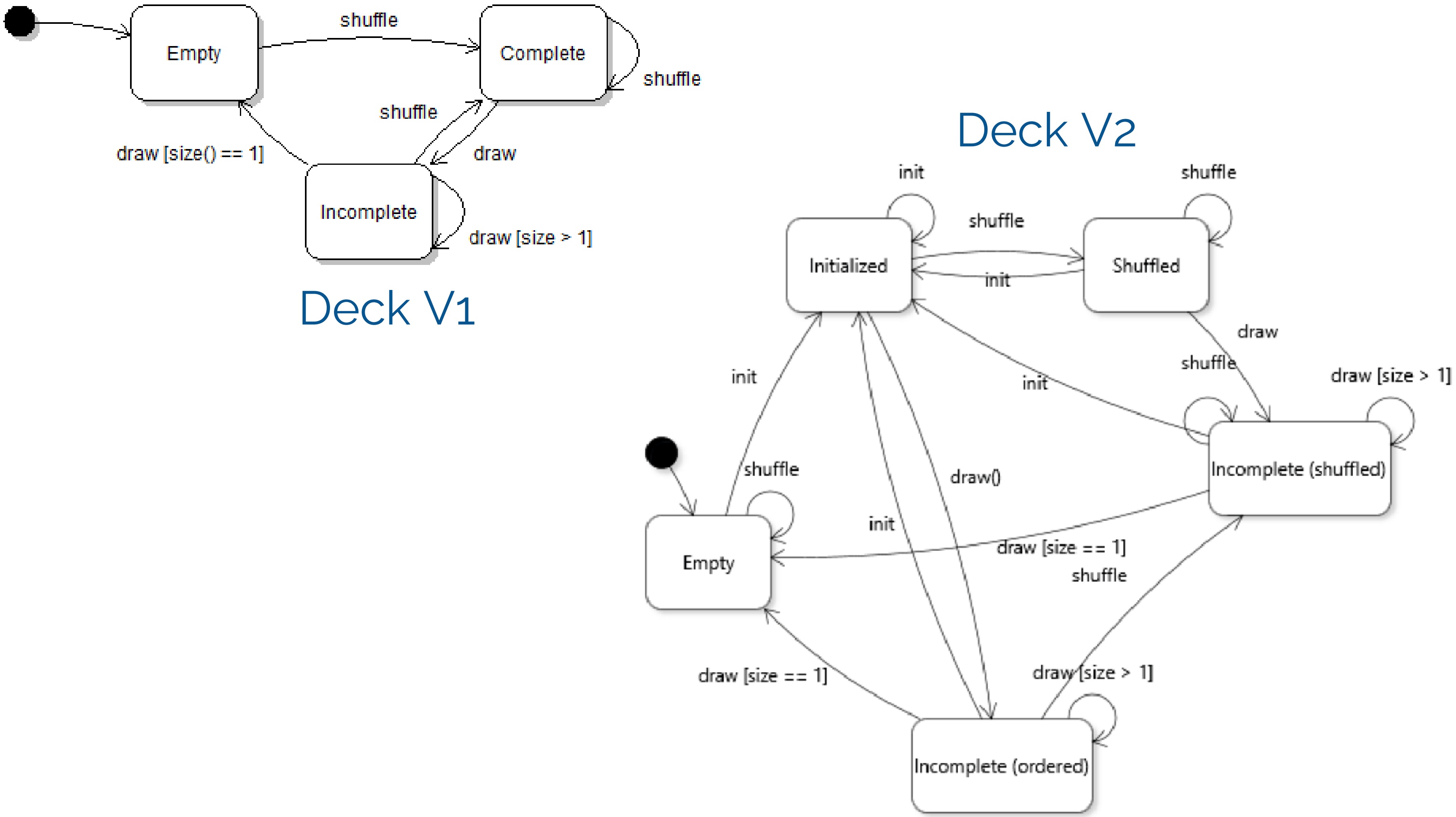


Event	State	Comment	x
		Start	3
	A	Entry of A	4
	A/B	Entry of B	2
e1	A/B	Exit of B x-- (self-transition) Entry of B	5 4 2
e2	A/C	Exit of B x=x*2	5 10
e4		x++ in C	11
e4		x++ in C	12
e3	A/B	Entry of B	6
e1		Exit of B x-- (self-transition) Entry of B	9 8 **4**

Design principles for state machines

- Minimize the state space of your objects
  - Put only the states that are strictly necessary for your object to satisfy their responsibilities
  - Objects with complex abstract state spaces are:
    - Difficult to use (i.e., to call their methods)
    - Difficult to test
    - Difficult to implement
- Avoid "speculative generalization"
  - *"What if we need this one day?"* may be a dangerous question
    - Less understandable code
    - More test cases to write
    - More sources of bugs
  - YAGNI principle: "You aren't gonna need it!"

Example of speculative generalization: a deck of cards



Deck V1

Deck V2

Key takeaways

- Design principles:
  - SRP
  - Encapsulation & immutability
  - Avoid complexity
- Your models are starting to move now!
- State machine used for modelling the **internal states** of objects in your system
- You may have **a state machine for each important class** in your class diagrams
  - You may need fewer, e.g., basic data structures are usually passive entities → no state machines for them
  - You model only the **key states** of the objects, not everything

Readings

- UML@Classroom: An Introduction to Object-Oriented Modeling" – Chapter 5
- Introduction to Software Design with Java – Chapters 2 and 4.1, 4.2, 4.3, 4.4
- Engineering Software Products, Chapter 8.1
- A philosophy of Software Design, Chapters 7, 8, 9
- [optional] Learning UML 2.0 – chapter 14