

*Software Design, Modelling and Analysis in UML*

*Lecture 14: Hierarchical State Machines I*

*2016-12-22*

**Prof. Dr. Andreas Podelski, Dr. Bernd Westphal**

**Albert-Ludwigs-Universität Freiburg, Germany**

- **Missing Pieces:** Create and Destroy Transformers
- **Putting It All Together (Again)**
  - **Initial States**
  - **Consistency** wrt. OCL Constraints
- **Hierarchical State Machines**
  - **Overview**
  - **Abstract Syntax: States**
    - pseudo-states, regions, ...
  - **(Legal) System Configurations**
  - **Abstract Syntax: Transitions**
  - **Enabledness of Fork/Join Transitions**
    - scope, priority, maximality, ...

# *Putting It All Together*

# Initial States

**Recall:** a labelled transition system is  $(S, A, \rightarrow, S_0)$ .

We **have**

- $S$ : system configurations  $(\sigma, \varepsilon)$
- $\rightarrow$ : labelled transition relation  $(\sigma, \varepsilon) \xrightarrow[u]{(cons, Snd)} (\sigma', \varepsilon')$ .

**Wanted:** initial states  $S_0$ .

**Proposal:**

Require a (finite) set of **object diagrams**  $\mathcal{OD}$  as part of a UML model

$$(\mathcal{CD}, \mathcal{IM}, \mathcal{OD}).$$



And set

$$S_0 = \{(\sigma, \varepsilon) \mid \sigma \in G^{-1}(\mathcal{OD}), \quad \mathcal{OD} \in \mathcal{OD}, \quad \varepsilon \text{ empty}\}.$$

**Other Approach:** (used by Rhapsody tool) multiplicity of classes (plus initialisation code).

We can read that as an abbreviation for an object diagram.

# Semantics of UML Model (So Far)

---

The **semantics** of the **UML model**

$$\mathcal{M} = (\mathcal{C}\mathcal{D}, \mathcal{S}\mathcal{M}, \mathcal{O}\mathcal{D})$$

where

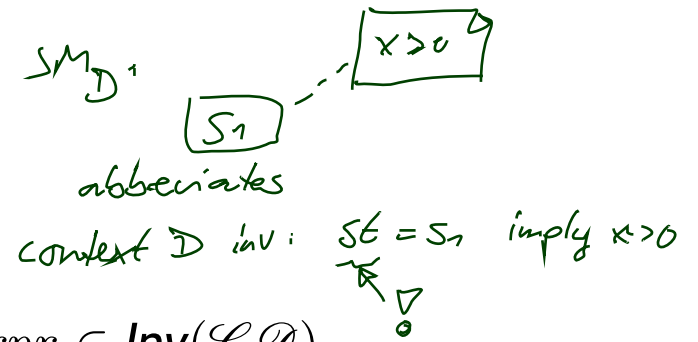
- some classes in  $\mathcal{C}\mathcal{D}$  are stereotyped as 'signal' (standard),  
some signals and attributes are stereotyped as 'external' (non-standard),
- there is a 1-to-1 relation between classes and state machines,
- $\mathcal{O}\mathcal{D}$  is a set of object diagrams over  $\mathcal{C}\mathcal{D}$ ,

is the **transition system**  $(S, A, \rightarrow, S_0)$  constructed on the previous slide(s).

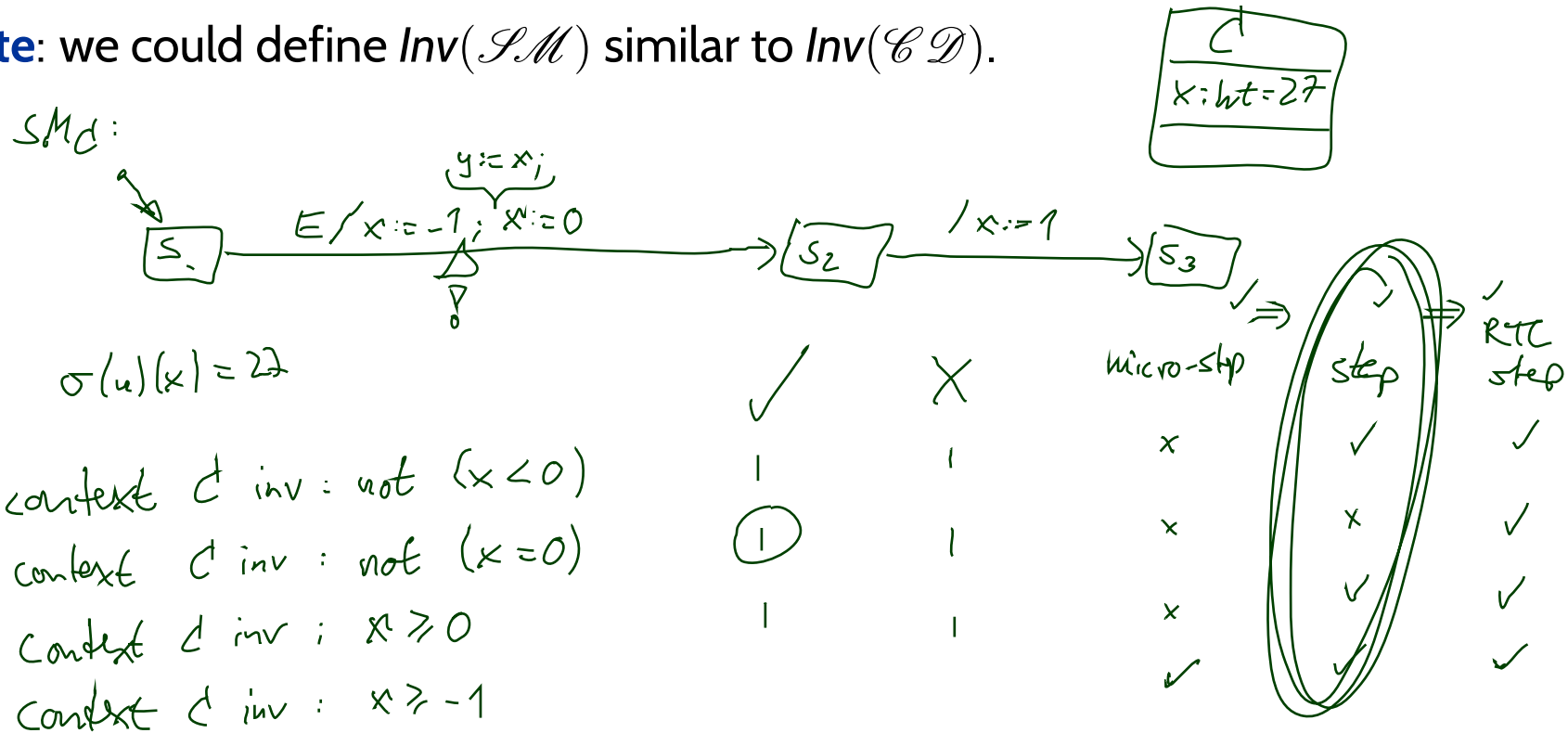
The **computations of**  $\mathcal{M}$  are the computations of  $(S, A, \rightarrow, S_0)$ .

# OCL Constraints and Behaviour

- Let  $\mathcal{M} = (\mathcal{CD}, \mathcal{SM}, \mathcal{OD})$  be a UML model.
- We call  $\mathcal{M}$  **consistent** iff, for each OCL constraint  $expr \in Inv(\mathcal{CD})$ ,  
 $\sigma \models expr$  for each “reasonable point”  $(\sigma, \varepsilon)$  of computations of  $\mathcal{M}$ .  
 (Cf. tutorial for discussion of “reasonable point”)



**Note:** we could define  $Inv(\mathcal{SM})$  similar to  $Inv(\mathcal{CD})$ .



# *Last Missing Piece: Create and Destroy Transformer*

# Transformer: Create

<b>abstract syntax</b> $\text{create}(C, \text{expr}, v)$	<b>concrete syntax</b> $\text{expr}.v := \text{new } C$
<b>intuitive semantics</b> Create an object of class $C$ and assign it to attribute $v$ of the object denoted by expression $\text{expr}$ .	
<b>well-typedness</b> $\text{expr} : T_D, v \in \text{atr}(D),$ $\text{atr}(C) = \{ \langle v_i : T_i, \text{expr}_i^0 \rangle \mid 1 \leq i \leq n \}$	
<b>semantics</b> ...	
<b>observables</b> ...	
<b>(error) conditions</b> $I[\text{expr}](\sigma, \beta)$ not defined.	

$x = (\text{new } C).y + (\text{new } D).z;$

can be written as

$\text{tmp}_1 := \text{new } C;$

$\text{tmp}_2 := \text{new } D;$

$x = \text{tmp}_1.y + \text{tmp}_2.z;$



# Transformer: Create

<b>abstract syntax</b>	<b>concrete syntax</b>
$\text{create}(C, \text{expr}, v)$	
<b>intuitive semantics</b>	
<i>Create an object of class <math>C</math> and assign it to attribute <math>v</math> of the object denoted by expression <math>\text{expr}</math>.</i>	
<b>well-typedness</b>	
$\text{expr} : T_D, v \in \text{atr}(D),$ $\text{atr}(C) = \{ \langle v_1 : T_1, \text{expr}_i^0 \rangle \mid 1 \leq i \leq n \}$	
<b>semantics</b>	...
<b>observables</b>	...
<b>(error) conditions</b>	$I[\![\text{expr}]\!](\sigma, \beta)$ not defined.

- We use an “and assign”-action for simplicity – it doesn’t add or remove expressive power, but moving creation to the expression language raises all kinds of other problems since then expressions would need to modify the system state.
- Also for simplicity: no parameters to construction ( $\sim$  parameters of constructor). Adding them is straightforward (but somewhat tedious).

# How To Choose New Identities?

---

- **Re-use**: choose any identity that is not alive now, i.e. not in  $\text{dom}(\sigma)$ .
  - Doesn't depend on history.
  - May “undangle” dangling references – may happen on some platforms.
- **Fresh**: choose any identity that has not been alive **ever**, i.e. not in  $\text{dom}(\sigma)$  and any predecessor in current run.
  - Depends on history.
  - Dangling references remain dangling – could mask “dirty” effects of platform.

# Transformer: Create

## abstract syntax

$\text{create}(C, \text{expr}, v)$

## concrete syntax

## intuitive semantics

Create an object of class  $C$  and assign it to attribute  $v$  of the object denoted by expression  $\text{expr}$ .

## well-typedness

$$\text{expr} : T_D, v \in \text{atr}(D),$$

$$\text{atr}(C) = \{\langle v_1 : T_1, \text{expr}_i^0 \rangle \mid 1 \leq i \leq n\}$$

## semantics

$$((\sigma, \varepsilon), (\sigma', \varepsilon')) \in t_{\text{create}(C, \text{expr}, v)}[u_x]$$

similar to update iff similar to read

$$\sigma' = \sigma[u_0 \mapsto \sigma(u_0)[v \mapsto u]] \cup \{u \mapsto \{v_i \mapsto d_i \mid 1 \leq i \leq n\}\},$$

$$\varepsilon' = [u](\varepsilon); \quad u \in \mathcal{D}(C) \text{ fresh, i.e. } u \notin \text{dom}(\sigma);$$

$$u_0 = I[\text{expr}](\sigma, u_x); \quad d_i = I[\text{expr}_i^0](\sigma, \emptyset) \text{ if } \text{expr}_i^0 \neq * \text{ and}$$

arbitrary value from  $\mathcal{D}(T_i)$  otherwise.

clean  
edges

## observables

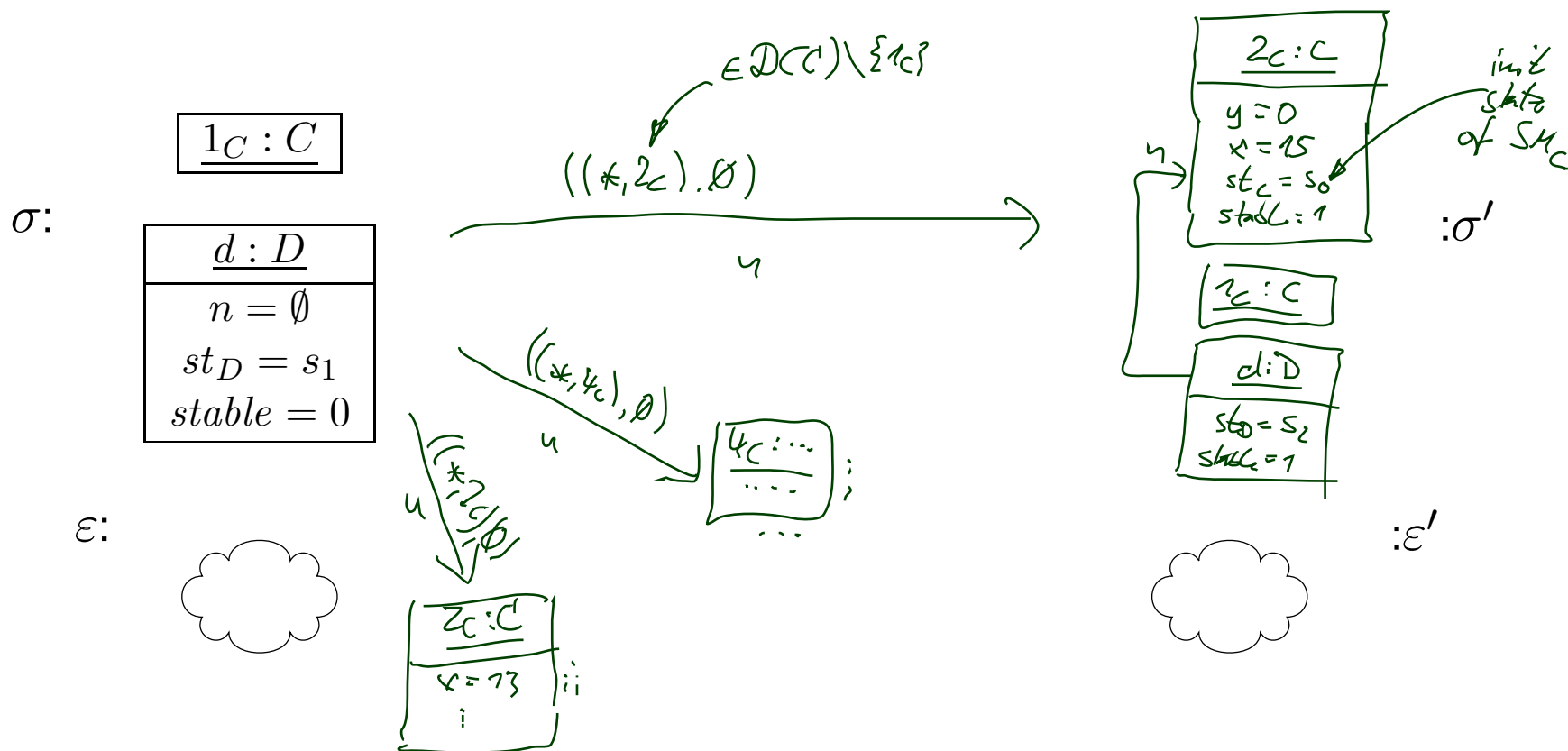
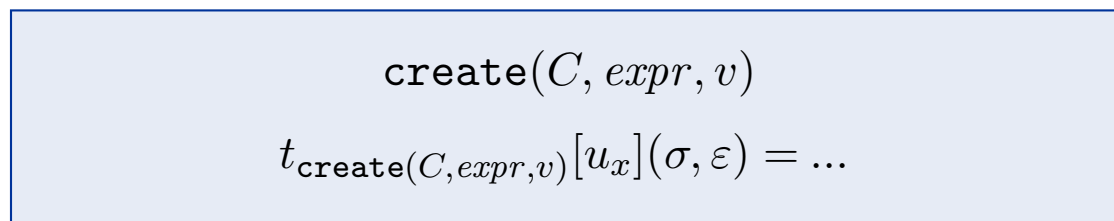
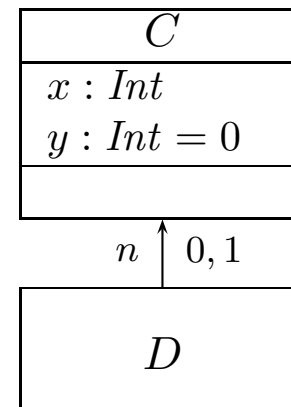
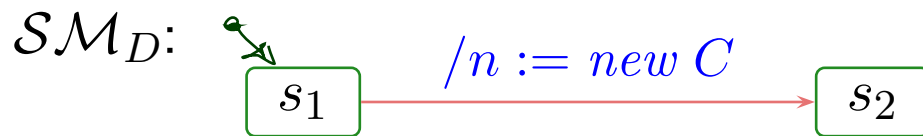
$$\text{Obs}_{\text{create}}[u_x] = \{(*, u)\}$$

creation

## (error) conditions

$$I[\text{expr}_i^0](\sigma, u_x) \text{ not defined.}$$

# Create Transformer Example



# Transformer: Destroy

abstract syntax	concrete syntax
$\text{destroy}(expr)$	
<b>intuitive semantics</b>	<i>Destroy the object denoted by expression <math>expr</math>.</i>
<b>well-typedness</b>	$expr : T_C, C \in \mathcal{C}$
<b>semantics</b>	...
<b>observables</b>	$Obs_{\text{destroy}}[u_x] = \{(u_x, \perp, (+, \emptyset), u)\}$
<b>(error) conditions</b>	$I \llbracket expr \rrbracket (\sigma, \beta)$ not defined.

# What to Do With the Remaining Objects?

---

Assume object  $u_0$  is destroyed...

- object  $u_1$  may still refer to it via association  $r$ :
  - allow dangling references?
  - or remove  $u_0$  from  $\sigma(u_1)(r)$ ?
- object  $u_0$  may have been the last one linking to object  $u_2$ :
  - leave  $u_2$  alone?
  - or remove  $u_2$  also? (garbage collection)
- Plus: (temporal extensions of) OCL may have dangling references.

**Our choice:** Dangling references and no garbage collection!

This is in line with “expect the worst”, because there are target platforms which don’t provide garbage collection – and models shall (in general) be correct without assumptions on target platform.

**But:** the more “dirty” effects we see in the model, the more expensive it often is to analyse. Valid proposal for simple analysis: monotone frame semantics, no destruction at all.

# Transformer: Destroy

## abstract syntax

$\text{destroy}(expr)$

## concrete syntax

## intuitive semantics

Destroy the object denoted by expression  $expr$ .

## well-typedness

$expr : T_C, C \in \mathcal{C}$

## semantics

$t_{\text{destroy}(expr)}[u_x](\sigma, \varepsilon) = \{(\sigma', \varepsilon')\}$      $\varepsilon' = [u](\varepsilon)$

*function restriction*

where  $\sigma' = \sigma|_{\text{dom}(\sigma) \setminus \{u\}}$  with  $u = I[expr](\sigma, u_x)$ .

## observables

$Obs_{\text{destroy}(expr)}[u_x] = \{(+, u)\}$

## (error) conditions

$I[expr](\sigma, u_x)$  not defined.

# Destroy Transformer Example

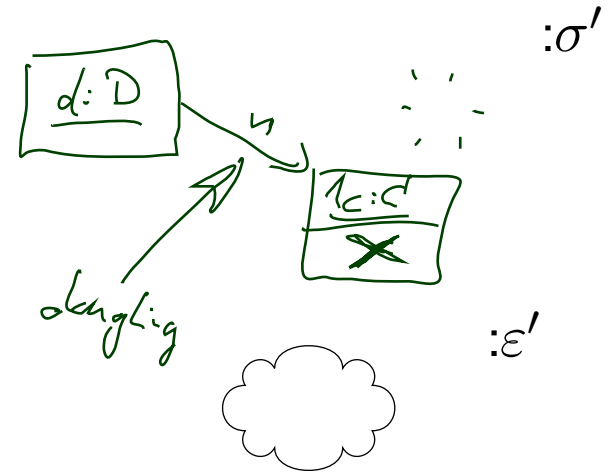
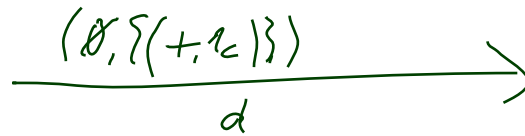
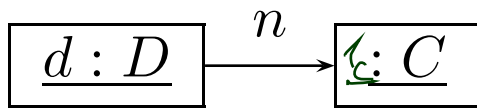
$SM_C$ :



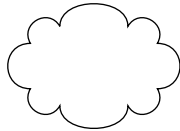
$destroy(expr)$

$t_{destroy(expr)}[u_x](\sigma, \varepsilon) = \dots$

$\sigma$ :



$\varepsilon$ :





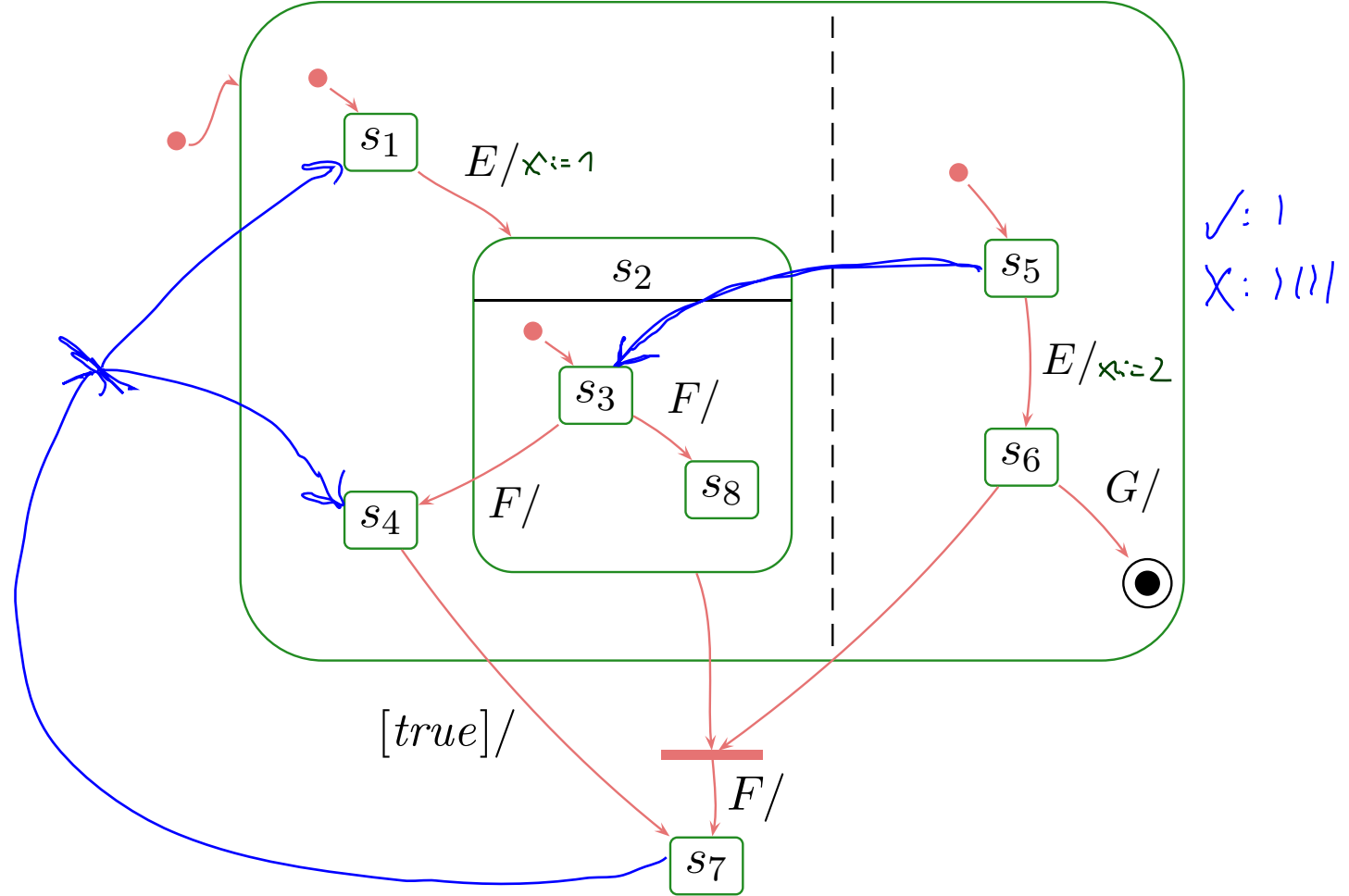
# *Hierarchical State-Machines*

# The Full Story

UML distinguishes the following **kinds of states**:

	example		example
<b>simple state</b>		<b>pseudo-state</b>	
<b>final state</b>		<b>fork/join</b>	
<b>composite state</b>		<b>junction, choice</b>	
<b>OR</b>		<b>entry point</b>	
<b>AND</b>		<b>exit point</b>	
		<b>terminate</b>	
		<b>submachine state</b>	

# Blessing or Curse...?



# Blessing or Curse...?

## Plan:

### States / Syntax:

- What is the abstract syntax of a diagram?

### States / Semantics:

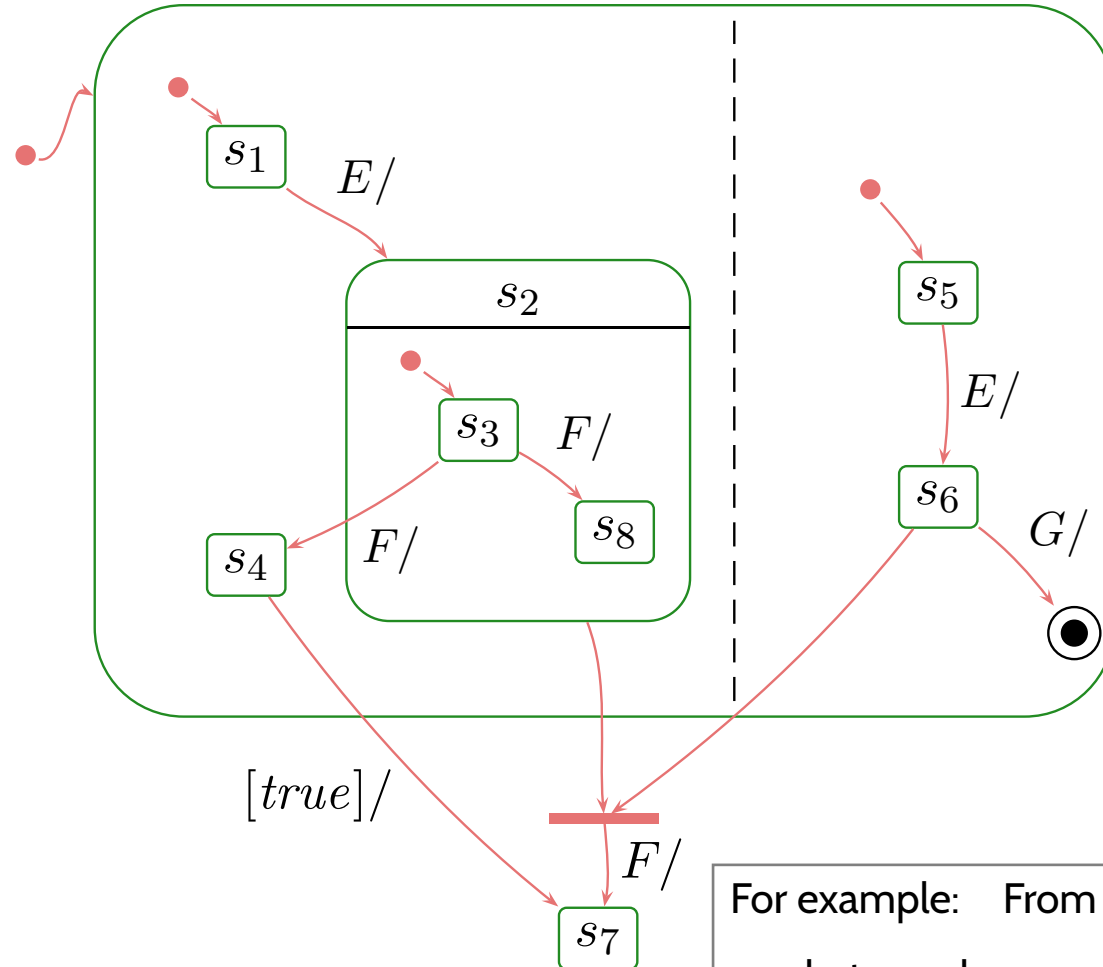
- what is the type of the implicit *st* attribute?
- what are **legal system configurations**?

### Transitions / Syntax:

- what are **legal** / well-formed transitions?

### Transitions / Semantics:

- when is a legal transition enabled?
- which effects do transitions have?

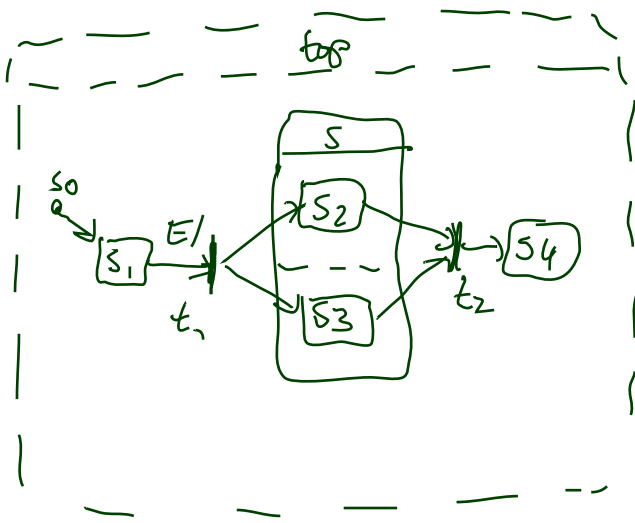
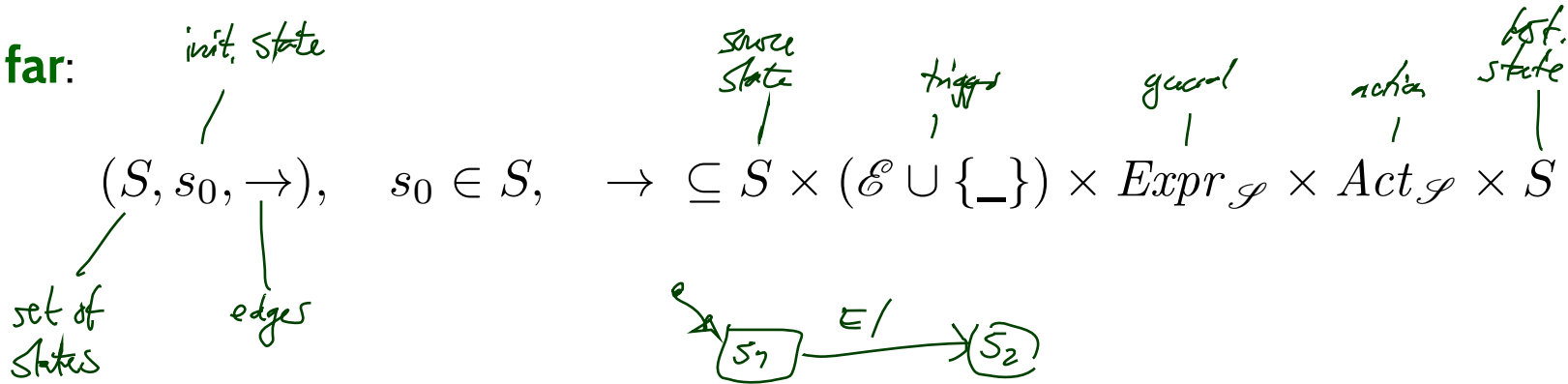


For example: From  $s_1, s_5$ ,

- what may happen on  $E$ ?
- what may happen on  $\underline{E}, F$ ?
- can  $\underline{E}, G$  kill the object?
- ...

# Representing All Kinds of States

• So far:



kind: init  
 $s_0$   
 kind: st  
 $\{s, s_1, \dots, s_4, top\},$   
 $\{t_1, t_2\},$   
 $\{t_1 \mapsto (\{s_1\}, \{s_2, s_3\}), \dots\}$   
 $\{t_1 \mapsto (E, tree, ship)\}$   
 $s \mapsto \{ \{s_2\}, \{s_3\} \},$   
 $s_2 \mapsto \emptyset$   
 $top \mapsto \{ \{s_0, s_1, s, s_4\} \}$

# Representing All Kinds of States

- **So far:**

$$(S, s_0, \rightarrow), \quad s_0 \in S, \quad \rightarrow \subseteq S \times (\mathcal{E} \cup \{\_ \}) \times Expr_{\mathcal{J}} \times Act_{\mathcal{J}} \times S$$

- **From now on: (hierarchical) state machines**

$$(S, kind, region, \rightarrow, \psi, annot)$$

where

- $S \supseteq \{top\}$  is a finite set of states (new: *top*),
- $kind : S \rightarrow \{st, init, fin, shist, dhist, fork, join, junc, choi, ent, exi, term\}$   
is a function which labels states with their **kind**, (new)
- $region : S \rightarrow 2^{2^S}$  is a function which characterises the **regions** of a state, (new)
- $\rightarrow$  is a set of transitions, (changed)
- $\psi : (\rightarrow) \rightarrow 2^S \times 2^S$  is an **incidence function**, and (new)
- $annot : (\rightarrow) \rightarrow (\mathcal{E} \cup \{\_ \}) \times Expr_{\mathcal{J}} \times Act_{\mathcal{J}}$   
provides an annotation for each transition. (new)

( $s_0$  is then redundant – replaced by proper state (!) of kind ‘*init*’.)

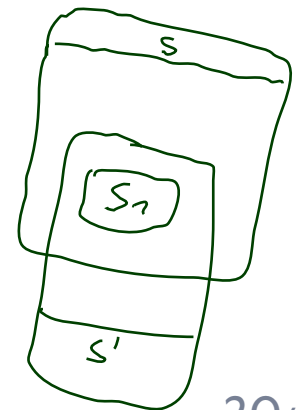
# Well-Formedness: Regions

	$\in S$	<i>kind</i>	$region \subseteq 2^S, S_i \subseteq S$	$child \subseteq S$
<b>final state</b>	$s$	<i>fin</i>	$\emptyset$	$\emptyset$
<b>pseudo-state</b>	$s$	<i>init, ...</i>	$\emptyset$	$\emptyset$
<b>simple state</b>	$s$	<i>st</i>	$\emptyset$	$\emptyset$
<b>composite state</b>	$s$	<i>st</i>	$\{S_1, \dots, S_n\}, n \geq 1$	$S_1 \cup \dots \cup S_n$
<b>implicit top state</b>	<i>top</i>	<i>st</i>	$\{S_1\}$	$S_1$

- Final and pseudo states **must not comprise** regions.
- States  $s \in S$  with  $kind(s) = st$  **may comprise** regions.



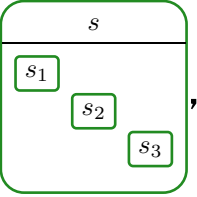
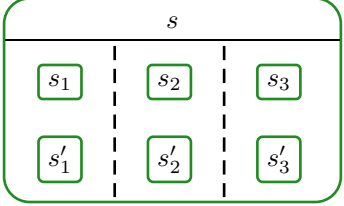

Naming conventions can be defined based on regions:

- No region: simple state.
- One region: OR-state.
- Two or more regions: AND-state.
- Each state (except for *top*) **must** lie in exactly one region.
- Note**: The region function induces a **child** function.
- Note**: Diagramming tools (like Rhapsody) can ensure well-formedness.



# From UML to Hierarchical State Machine: By Example

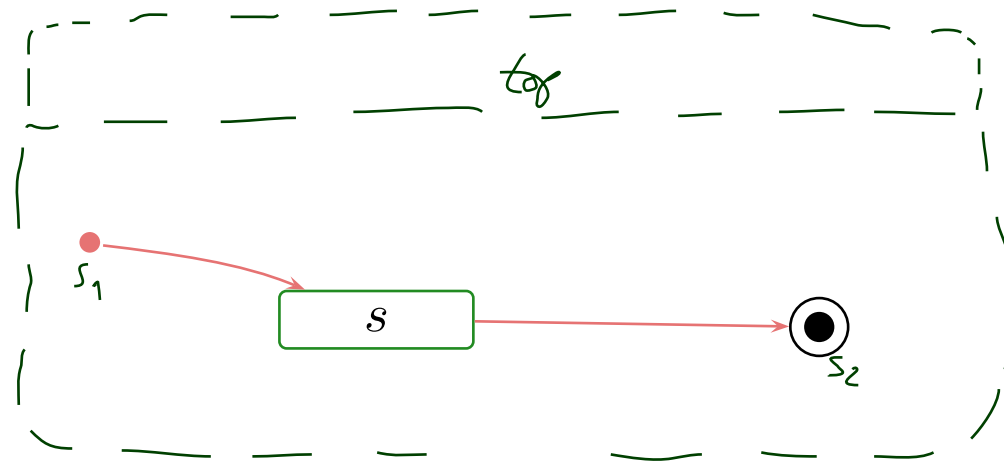
$(S, kind, region, \rightarrow, \psi, annot)$

	example	$\in S$	kind	region
<b>simple state</b>		$s$	$st$	$\emptyset$
<b>final state</b>		$q$	$fin$	$\emptyset$
<b>composite state</b>				
<b>OR</b>		$s$	$st$	$\{ \{s_1, s_2, s_3\} \}$
<b>AND</b>		$s$	$st$	$\{ \{s_1, s_1'\}, \{s_2, s_2'\}, \{s_3, s_3'\} \}$
<b>submachine state</b>	(later) -	-	-	
<b>pseudo-state</b>		$q$	$int, \dots$	$\emptyset$

$(s, kind(s))$  for short



# From UML to Hierarchical State Machine: By Example



... denotes  $(S, kind, region, \rightarrow, \psi, annot)$  with

- $S = \{top, s_1, s, s_2\}$
- $kind = \{top \mapsto \mathbf{st}, s_1 \mapsto \mathbf{init}, s \mapsto \mathbf{st}, s_2 \mapsto \mathbf{fin}\}$
- $or(S, kind) = \{(top, \mathbf{st}), (s_1, \mathbf{init}), (s, \mathbf{st}), (s_2, \mathbf{fin})\}$
- $region = \{top \mapsto \{\{s_1, s, s_2\}\}, s_1 \mapsto \emptyset, s \mapsto \emptyset, s_2 \mapsto \emptyset\}$
- $\rightarrow, \psi, annot$ : in a minute.

# Recall

## Plan:

### States / Syntax:

- What is the abstract syntax of a diagram? ✓

### States / Semantics:

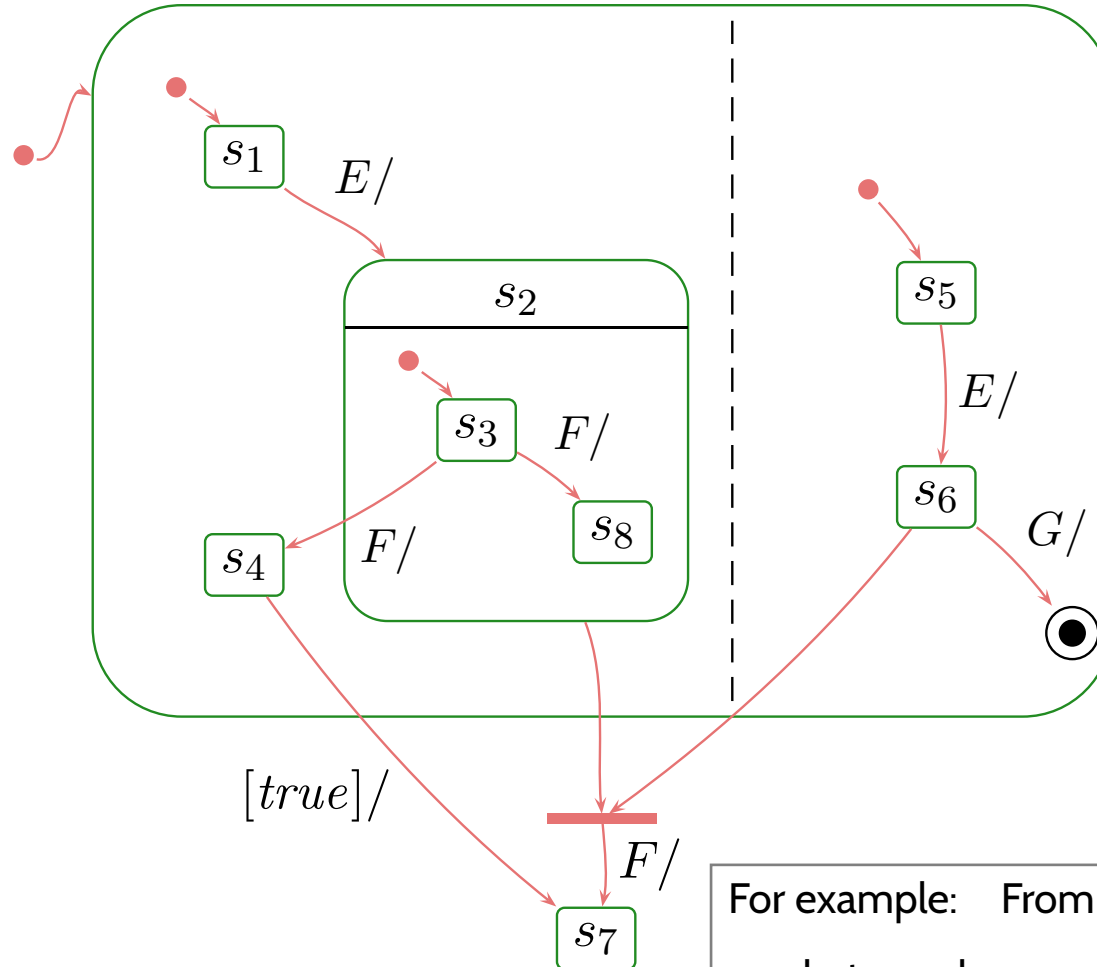
- what is the type of the implicit *st* attribute?
- what are **legal system configurations**?

### Transitions / Syntax:

- what are **legal** / well-formed transitions?

### Transitions / Semantics:

- when is a legal transition enabled?
- which effects do transitions have?



For example: From  $s_1, s_5$ ,

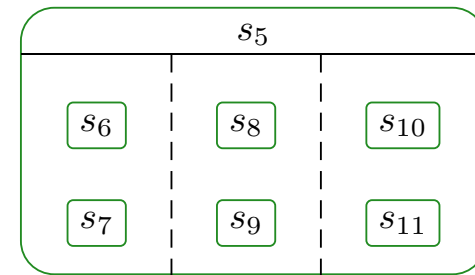
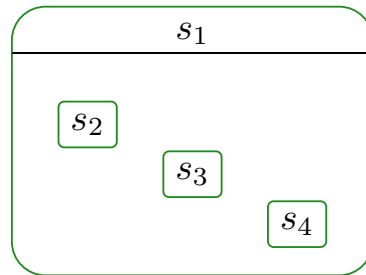
- what may happen on  $E$ ?
- what may happen on  $\underline{E}, F$ ?
- can  $\underline{E}, G$  kill the object?
- ...

# Semantics: State Configuration

- The type of (implicit attribute)  $st$  is from now on **a set of** states, i.e.  $\mathcal{D}(S_{MC}) = 2^S$
- A set  $S_1 \subseteq S$  is called **(legal) state configuration** if and only if
  - $top \in S_1$ , and
  - for each region  $R$  of a state in  $S_1$ , exactly one (non pseudo-state) element of  $R$  is in  $S_1$ , i.e.

$$\forall s \in S_1 \forall R \in region(s) \bullet |\{s \in R \mid kind(s) \in \{st, fin\}\} \cap S_1| = 1.$$

## Examples:



$$S_1 = \{s_0\} \quad \times$$

$$S_2 = \{s_0, top\} \quad \checkmark$$

$$S_3 = \{s_1, top\} \quad \times$$

$$S_4 = \{s_1, top, s_3, s_4\} \quad \times$$

$$S_5 = \{s_1, top, s_4\} \quad \checkmark$$

$$S_6 = \{s_5, top, s_6, s_9\} \quad \times$$

$$S_7 = \{s_5, top, s_6, s_7, s_{10}\} \quad \checkmark$$

$$S_8 = \{top, s_0, s_1, s_2\} \quad \times$$

# Recall

## Plan:

### States / Syntax:

- What is the abstract syntax of a diagram? ✓

### States / Semantics:

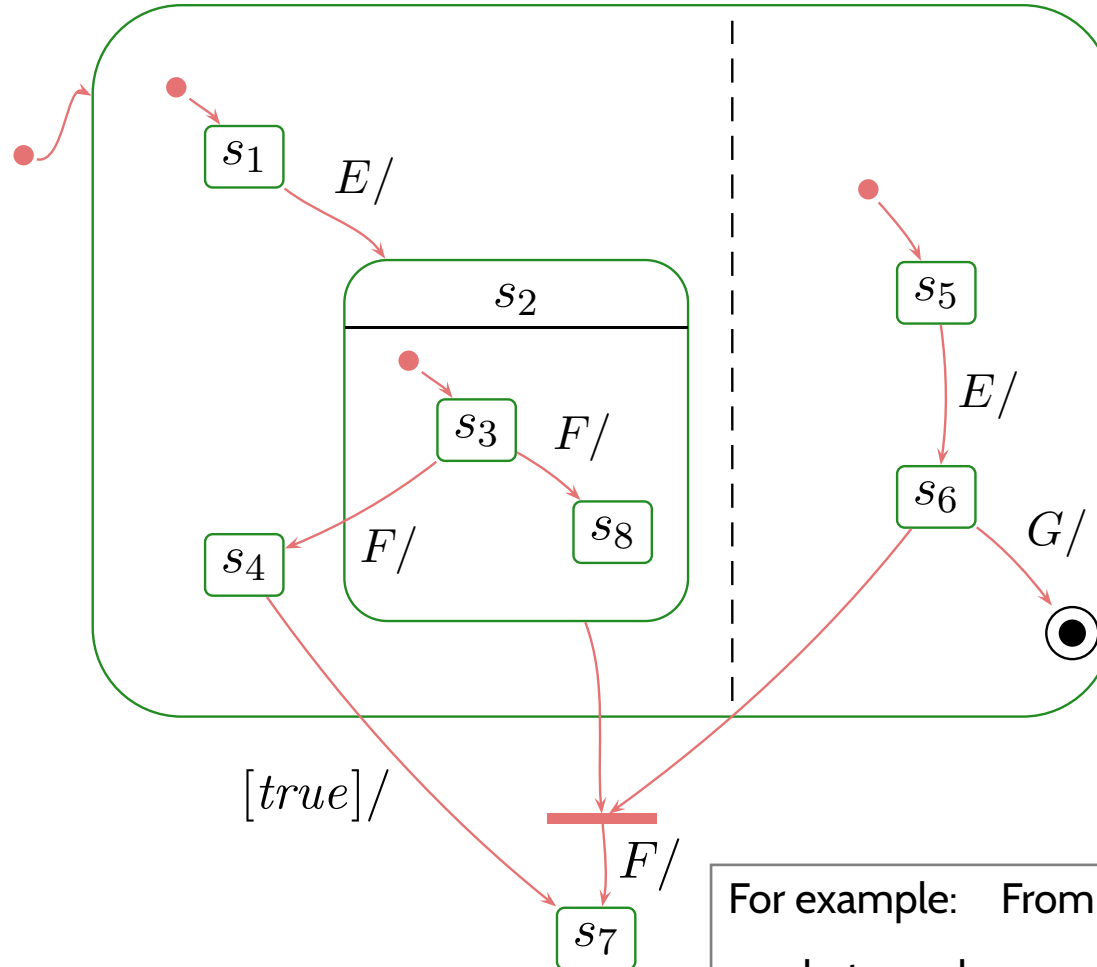
- what is the type of the implicit *st* attribute? ✓
- what are **legal system configurations**? ✓

### Transitions / Syntax:

- what are **legal** / well-formed transitions?

### Transitions / Semantics:

- when is a legal transition enabled?
- which effects do transitions have?



For example: From  $s_1, s_5$ ,

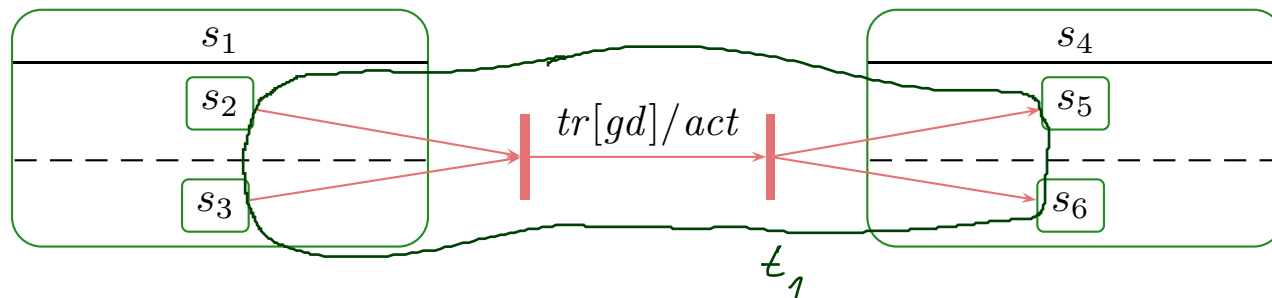
- what may happen on  $E$ ?
- what may happen on  $\underline{E}, F$ ?
- can  $\underline{E}, G$  kill the object?
- ...

# Transitions Syntax: Fork/Join

- For simplicity, we consider transitions with (possibly) multiple sources and targets, i.e.

$$\psi : (\rightarrow) \rightarrow (2^S \setminus \emptyset) \times (2^S \setminus \emptyset)$$

- For instance,



translates to

$$(S, kind, region, \underbrace{\{t_1\}}_{\rightarrow}, \underbrace{\{t_1 \mapsto (\{s_2, s_3\}, \{s_5, s_6\})\}}_{\psi}, \underbrace{\{t_1 \mapsto (tr, gd, act)\}}_{annot})$$

- Naming convention:  $\psi(t) = (source(t), target(t))$ .

# Orthogonal States

---

- Two states  $s_1, s_2 \in S$  are called **orthogonal**, denoted  $s_1 \perp s_2$ , if and only if
  - they “live” in different regions of **one** AND-state, i.e.

$$\exists s, \text{region}(s) = \{S_1, \dots, S_n\}, 1 \leq i \neq j \leq n : s_1 \in \text{child}(S_i) \wedge s_2 \in \text{child}(S_j),$$

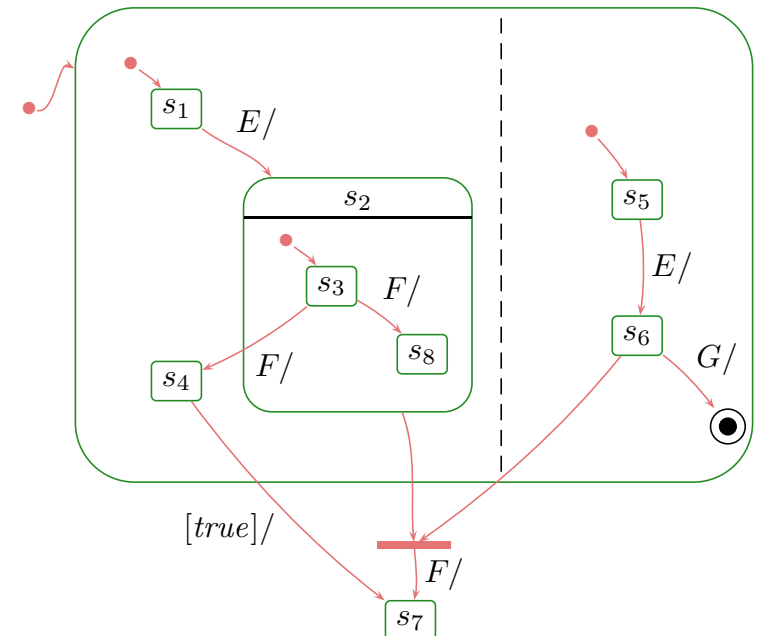
# Legal Transitions

A hierarchical state-machine  $(S, kind, region, \rightarrow, \psi, annot)$  is called **well-formed** if and only if for all transitions  $t \in \rightarrow$ ,

- source (and destination) states are pairwise orthogonal, i.e.
  - $\forall s, s' \in source(t) (\in target(t)) \bullet s \perp s'$ ,
- the top state is neither source nor destination, i.e.
  - $top \notin source(t) \cup target(t)$ .

**Recall:** final states are not sources of transitions.

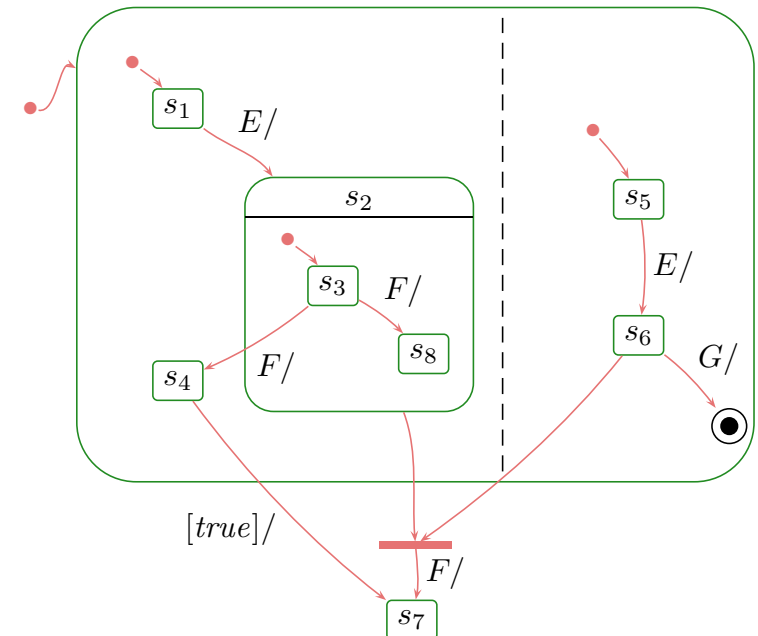
**Example:**



# Plan

	example		example
<b>simple state</b>	<pre> s1 entry/act1^entry do/act1^do exit/act1^exit E1/actE1 ... En/actEn                     </pre>	<b>pseudo-state</b>	
<b>final state</b>		initial	
<b>composite state</b>		(shallow) history	
OR		deep history	
AND		fork/join	
		junction, choice	
		entry point	
		exit point	
		terminate	
		<b>submachine state</b>	

- Transitions involving non-pseudo states.
- Initial pseudostate, final state.
- Entry/do/exit actions, internal transitions.
- History and other pseudostates, the rest.





# Tell Them What You've Told Them. . .

---

- For the **Create Action**, we have two main choices:
  - **re-use** identities (“nasty semantics”),
  - use **fresh** identities (“clean semantics”, depends on history).Similar for **Destroy**.
- **Hierarchical State Machines** introduce **Regions**.
  - Thereby, **states** can lie within **states** as **children**.
  - The implicit variable *st* becomes set-valued.
- **Transitions** may now have
  - **multiple** source states, **multiple** destination states,
  - but need to adhere to **well-formedness conditions**.
- **Enabledness** of a set (!) of transitions is **a bit tricky to define** (→ scope, priority, maximality).
- **Steps** are a proper generalisation of core state machines.

# *References*

# *References*

---

OMG (2011a). Unified modeling language: Infrastructure, version 2.4.1. Technical Report formal/2011-08-05.

OMG (2011b). Unified modeling language: Superstructure, version 2.4.1. Technical Report formal/2011-08-06.