Foundations of the UML

Lecture 16: The Object Constraint Language

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

http://moves.rwth-aachen.de/i2/370

11. Januar 2010

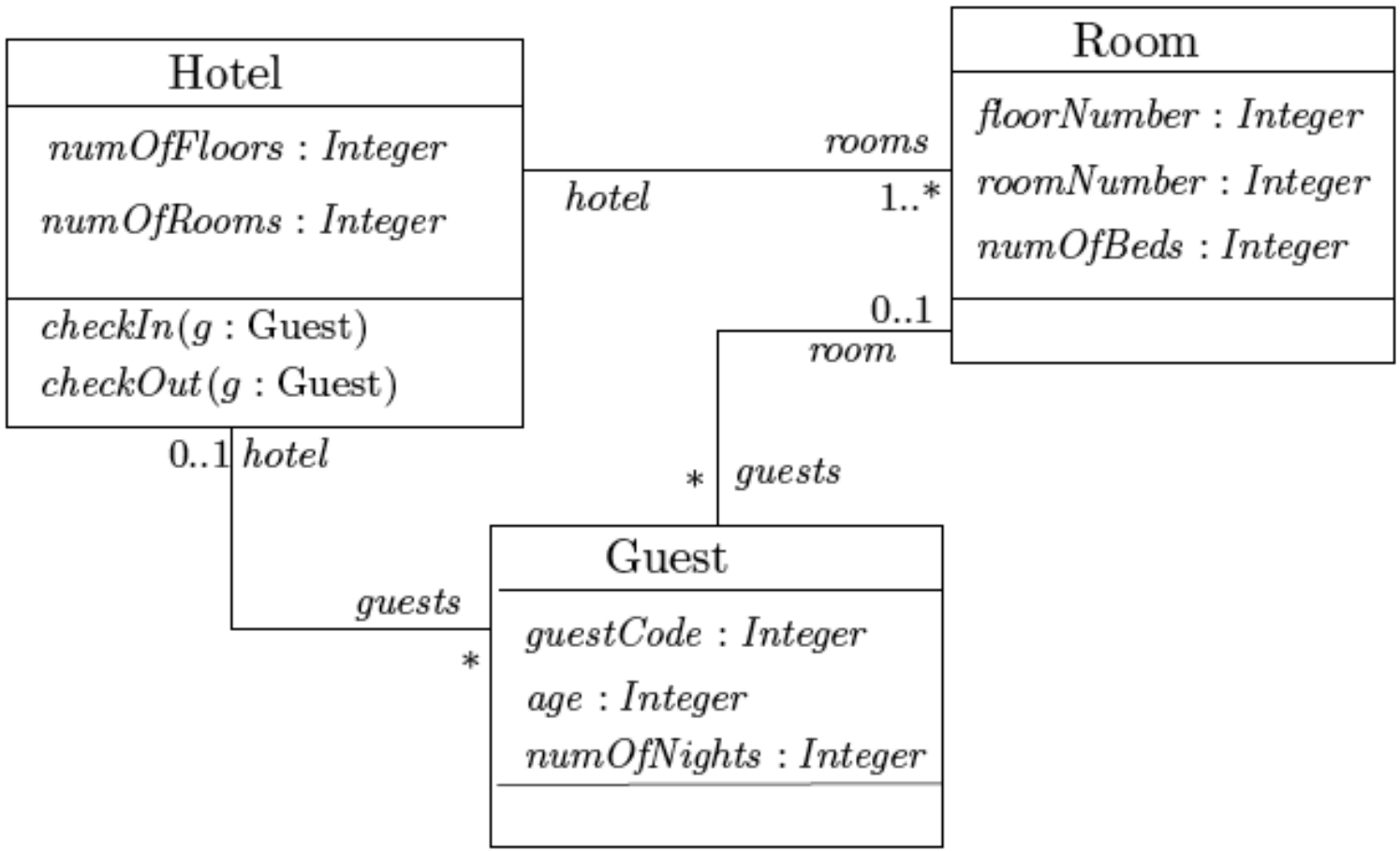
RWTH AACHEN UNIVERSITY

What is the OCL?

- Textual annotation to UML diagrams
  - such as class diagrams, statecharts, activity diagrams, sequence diagrams
- Strongly related to predicate and first-order logic
- OCL constraints impose restrictions on the UML diagrams
  - e.g., such as an invariant stating that x should always exceed 0
- OCL 2.0 has been adopted by the UML standardization bodies
  - originally developed by Jos Warmer and Anneke Kleppe (Klasse Objecten)
- Basic ingredients:
  - typed variables, expressions, navigations, constraints, and iterations

RWTH AACHEN UNIVERSITY

Running example



Class diagrams

- Class diagrams depict the static structure of classes and their associations
- A class is a template for the creation of its instances, i.e., its objects.
- It specifies its objects by providing its attributes and methods.
- A method is given by its name, formal parameters and return type.
- The object running a method is the owner and is indicated by self.
- Associations represent the relationship between classes.

RWTH AACHEN UNIVERSITY

OCL examples

- The number of guests in a room cannot exceed the room's capacity:

```
context Room
inv: guests → size ⩽ nrOfBeds
```

- The guests in the rooms of the hotel equal the guests in the hotel

```
context Hotel
inv: rooms.guests = guests
```

- These are invariants, i.e., they should hold in any system state

- The violation of an invariant can always be shown by a finite system run that ends in state that refutes the invariant condition.

RWTH AACHEN UNIVERSITY

OCL examples

- On checking in a guest g, say, the following conditions should hold:
  - g should not already be a hotel guest, and
  - after checking in, the number of guest is increased by one, and
  - the hotel's guest should include g

```
context  Hotel:: checkIn (g: Guest)
pre: not guests → includes(g)
post: guests → size = (guests@pre → size) + 1  and
      guests → includes(g)
```

where guests@`pre` refers to the value of the attribute guests at evaluting the `pre`condition

- On each invocation of the method checkIn, if the `pre`condition holds, then on termination of checkIn, the `post`condition holds

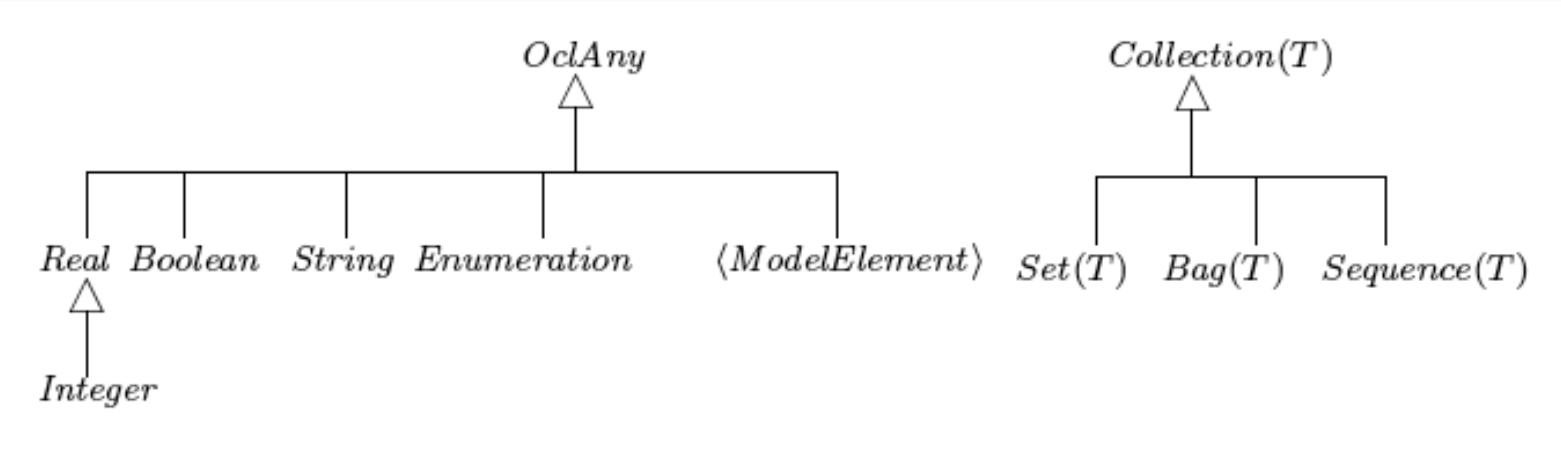
RWTH AACHEN UNIVERSITY

OCL types

OCL is a typed language. Its types are:

1. Predefined types, equipped with standard operations:
   - basic types (e.g., Int, String, Boolean, Char, ...)
   - collection types (e.g., Collection, Set, Bag, Sequence, ...)
   - OclAny
2. Model types: the types in the UML model (e.g., Hotel, Room, etc.)

Type hierarchy:



Assumption: all OCL terms are type-correct.

OCL constraints

Definition (OCL constraints)

The syntax of OCL constraints is defined by the grammar:

$$\chi ::= \texttt{context}\ C\ \texttt{inv}\ \xi \mid \texttt{context}\ C :: M(\vec{p})\ \texttt{pre}\ \xi\ \texttt{post}\ \xi$$

where C is a class, $M \in dom(C.meths)$ is a method of class C, $\vec{p}$ are parameters and ξ is an OCL expression (see later).

So:

- OCL constraints are built from OCL expressions and have type Boolean
- Invariants have as context a class
- Pre- and postconditions have as context a method of a class.

OCL expressions

Definition (OCL expressions)

The syntax of OCL expressions is defined by the grammar:

$$\xi ::= \texttt{self} \mid z \mid \texttt{result} \mid \xi\texttt{@pre} \mid \xi.a \mid \omega(\xi,\ldots,\xi) \mid$$
$$\xi.\omega(\xi,\ldots,\xi) \mid \xi \rightarrow \omega(\xi,\ldots,\xi) \mid \xi \rightarrow \texttt{iterate}(x_1; x_2 := \xi \mid \xi)$$

where:

- `self` refers to the context object of class C
- z represents either an attribute of the context object, or a formal parameter of a context method, or a logical variable
- `result` refers to the value returned by the context method (which is undefined if this method has not yet returned a value)
- `@pre` refers to the value of its operand on invoking this method
- expressions `result` and `@pre` can be used in `post`conditions only

OCL operations

- $\xi.a$ is an attribute/parameter navigation
  - ξ is an object reference to an object with attribute a, or
  - ξ is a reference to a method occurrence with a formal parameter a
  - $\xi.a$ denotes the value of this attribute/parameter
  - examples: x.rooms.guests, and hotel.rooms, etc.

- $\omega(\xi_1, \ldots, \xi_n)$ denotes the application of the n-ary operator ω to the arguments ξ_1 through ξ_n

  some examples are: isEqual(g_1, g_2) and ifThenElse(b, ξ_1, ξ_2), etc.

- $\xi.\omega(\xi_1, \ldots, \xi_n)$ represents an operator ω on basic types applied on ξ and arguments ξ_1 through ξ_n

- $\xi \rightarrow \omega(\xi_1, \ldots, \xi_n)$ represents an operator ω on collection types applied on collection ξ and arguments ξ_1 through ξ_n

RWTH AACHEN UNIVERSITY

OCL iterations

Iteraions

$\xi_1 \rightarrow \texttt{iterate}(x_1; x_2 := \xi_2 \mid \xi_3)$ is an OCL expression that binds logical variables x_1 and x_2 in the following way:

1. x_2 is initialised to the value of expression ξ_2
2. the, the value of expression ξ_3 is computed repeatedly and assigned to x_2
3. while x_1 successively takes as its value an element of the sequence ξ_1.

Example:

$[1, 2, 3] \rightarrow \texttt{iterate}(x_1; x_2 := 0 \mid x_1 + x_2)$ yields the sum of the elements in the sequence $[1, 2, 3]$

OCL deficiencies

Iterations over unordered collections

The evaluation of `iterate` expressions over unordered collections (such as Set and Bag) is problematic, e.g.,

$$\{1, 2, 3\} \rightarrow \texttt{iterate}(x_1; x_2 := 0 \mid -x_1 + x_2)$$

is not well-defined, as depending on the order of binding the elements in the set $\{1, 2, 3\}$ to the variable x_1, the result will be -1, 3 or 5.

Flattening of nested collections

In OCL, nested collections are automatically flattened, e.g.,

$$Set\{Set\{1, 2\}, Set\{3, 4, 5\}\} = Set\{1, 2, 3, 4, 5\}$$

but flattening of ordered collections such as in $Sequence\{Set\{1, 3, 7\}\}$ is not well-defined but may yield any of the possible 8 orderings.

OCL operational model: data types and values

The OCL semantics is defined using an operational model of an object-based system.

Definition (Data types for logical variables)

- VNAME is a countable set if variable names
- MNAME is a countable set of method names (ranged over by M)
- CNAME is a countable set of class names (ranged over by C)

RWTH AACHEN UNIVERSITY

OCL operational model: data types and values

Definition (Semantic types)

The language TYPE of data types is defined by the grammar:

$$\tau \ ::= \ \text{void} \mid \text{nat} \mid \text{bool} \mid \tau \ \text{list} \mid C \ \text{ref} \mid C.M \ \text{ref}$$

where $C \in \text{CNAME}$ and $M \in \text{MNAME}$.

- void represents the unit type with trivial value (),
- τ list denotes the type of lists of τ with elements [] (the empty list) and $h :: w$ (list with head h of type τ and tail w of type τ list); notation $1 :: 2 :: []$ as $[1, 2]$ and $(1 :: []) :: (2 :: []) :: []$ as $[[1], [2]]$
- C ref is the type of objects of class C
- $C.M$ ref is the type of method occurrences of method M of class C

RWTH AACHEN UNIVERSITY

OCL operational model

Definition (Variable, method and class definitions)

We define the following partial functions:

- $\textsc{Vdecl}$: $\textsc{VName} \to \textsc{Type}$ maps variable names to types
- $\textsc{Mdecl}$: $\textsc{MName} \to \textsc{Vdecl} \times \textsc{Type}$ maps method names onto the formal parameters and the return type
- $\textsc{Cdecl}$: $\textsc{CName} \to \textsc{Vdecl} \times \textsc{Mdecl}$ maps class names to their attributes and methods

Notations

- Let $D \in \textsc{Cdecl}$. For $C \in dom(D)$, let $C.attrs$ denote its attributes and $C.meths$ its methods
- For method M of class C, $M.fpars$ are its formal parameters, and $M.retty$ is its return type
- Thus: $C.meths(M) = (M.fpars, M.retty)$

Example

$$
\begin{aligned}
\text{Hotel}.attrs(v) &= \begin{cases} \mathsf{nat} & \text{if } v \in \{numOfFloors, numOfRooms\} \\ \text{Room list} & \text{if } v = rooms \\ \text{Guest list} & \text{if } v = guests \\ \bot & \text{otherwise} \end{cases} \\
\text{Room}.attrs(v) &= \begin{cases} \mathsf{nat} & \text{if } v \in \{floorNumber, roomNumber, numOfBeds\} \\ \text{Hotel} & \text{if } v = hotel \\ \text{Guest list} & \text{if } v = guests \\ \bot & \text{otherwise} \end{cases} \\
\text{Guest}.attrs(v) &= \begin{cases} \mathsf{nat} & \text{if } v \in \{guestCode, age, numOfNights\} \\ \text{Hotel} & \text{if } v = hotel \\ \text{Room} & \text{if } v = room \\ \bot & \text{otherwise} \end{cases} \\
checkIn.fpars(v) &= \begin{cases} \text{Guest} & \text{if } v = g \\ \bot & \text{otherwise} \end{cases} \\
checkOut.fpars(v) &= checkIn.fpars(v) \\
\\
\text{Hotel}.meths(M) &= \begin{cases} (checkIn.fpars, \mathsf{void}) & \text{if } M = checkIn \\ (checkOut.fpars, \mathsf{void}) & \text{if } M = checkOut \\ \bot & \text{otherwise} \end{cases} \\
\text{Room}.meths(M) &= \bot \\
\text{Guest}.meths(M) &= \bot
\end{aligned}
$$

Objects

Objects

Objects will be numbered instances of their class $C \in \text{CNAME}$.

Let

- The domain of object ids of class C is defined by $\text{OID}^C = \{C\} \times \mathbb{N}$.
- Let $\text{OID} = \bigcup_C \text{OID}^C$ denote the set of object identifiers.

Thus:

Elements of OID are pairs (C, n), denoting the n-th instance of class C.

RWTH AACHEN UNIVERSITY

Events

Method invocations

Method occurrences, also called events, will be numbered instances of method $M \in \text{MNAME}$ plus an indication of the object executing M.

Let:

- $\text{EVT}^{C,M} = \text{OID}^C \times \{M\} \times \mathbb{N}$ be the domain of method invocations (= events) of M of class C
- Let $\text{EVT} = \bigcup_C \bigcup_M \text{EVT}^{C,M}$ denote the set of events.

Thus:

Elements of EVT are tuples $((C, n), M, k)$ denoting the k-th method invocation of M which currently is executed by object (C, n).

RWTH AACHEN UNIVERSITY

Example

Example 3.2.1. Consider the Hotel class diagram of Figure 2.3. The following are instances of the class Hotel:

$$(\text{Hotel}, 1) \quad (\text{Hotel}, 2) \quad (\text{Hotel}, 31) \quad (\text{Hotel}, 127) \quad \ldots$$

The following are events related to the method *checkIn*:

$$((\text{Hotel}, 1), \mathit{checkIn}, 1) \quad ((\text{Hotel}, 1), \mathit{checkIn}, 2)$$
$$((\text{Hotel}, 31), \mathit{checkIn}, 1) \quad ((\text{Hotel}, 127), \mathit{checkIn}, 3) \quad \ldots$$

Note that the first two events represent different executions of method *checkIn* performed by the same object. □

RWTH AACHEN UNIVERSITY

Values and operations

The combined universe of values will be denoted by $\textsc{Val}$; the set of values of a given type $\tau \in \textsc{Type}$ is denoted by $\textsc{Val}^\tau$. We define:

$$
\begin{aligned}
\textsc{Val}^{\mathsf{void}} &= \{()\} \\
\textsc{Val}^{\mathsf{nat}} &= \mathbb{N} \\
\textsc{Val}^{\mathsf{bool}} &= \{\mathsf{ff}, \mathsf{tt}\} \\
\textsc{Val}^{\tau\ \mathsf{list}} &= \{[]\} \cup \{h :: w \mid h \in \textsc{Val}^\tau, w \in \textsc{Val}^{\tau\ \mathsf{list}}\} \\
\textsc{Val}^{C\ \mathsf{ref}} &= \{\mathsf{null}\} \cup \textsc{OId}^C \\
\textsc{Val}^{C.M\ \mathsf{ref}} &= \textsc{Evt}^{C,M} \quad .
\end{aligned}
$$

- $+ : \textsc{Val}^{\mathsf{nat}} \times \textsc{Val}^{\mathsf{nat}} \to \textsc{Val}^{\mathsf{nat}}$ is the standard sum on natural numbers.
- $\mathit{sort} : \textsc{Val}^{\tau\ \mathsf{list}} \to \textsc{Val}^{\tau\ \mathsf{list}}$ orders a given list of values of type τ.
- $\mathit{flat} : \textsc{Val}^{\tau\ \mathsf{list\ list}} \to \textsc{Val}^{\tau\ \mathsf{list}}$ flattens nested lists.

Finally, there is a special element $\bot \notin \textsc{Val}$ that is used to model the "undefined" value: we write $\textsc{Val}_\bot = \textsc{Val} \cup \{\bot\}$. All operations are extended to $\bot$ by requiring them to be *strict* (meaning that if any operand equals $\bot$, the entire expression equals $\bot$). For instance, for lists we have $\bot :: w = \bot$ and $h :: \bot = \bot$.

Configurations

Definition (Configuration)

A configuration is a tuple (O, E, σ, γ) with:

- $O \subseteq \textsc{Oid}$, the currently alive objects
- $E \subseteq \textsc{Evt}$, the currently running method invocations
- $\sigma : O \to \textsc{Vname} \to \textsc{Val}$, the local state of objects in O
- $\gamma : E \to (\textsc{Vname} \to \textsc{Val}) \times \textsc{Val}_\perp$, the state of method invocations

State information

- $\sigma(o)$ is the local state of object o such that $\sigma(o) = \ell$ with $o \in \textsc{Oid}^C$ implies $dom(\ell) = dom(C.attrs)$ and $\ell(a) \in \textsc{Val}^{C.attrs(a)}$ for each $a \in dom(\ell)$.
- σ is extended point-wise to lists of objects, i.e.,

$$\sigma([])(a) = [] \quad \text{and} \quad \sigma(h :: w)(a) = \sigma(h)(a) :: \sigma(w)(a).$$

Configurations

Method invocations

Recall: $\gamma : E \to (\textsc{Vname} \to \textsc{Val}) \times \textsc{Val}_\perp$

If $\gamma(e) = (\ell, v)$ for $e \in \textsc{Evt}^{C,M}$ then:

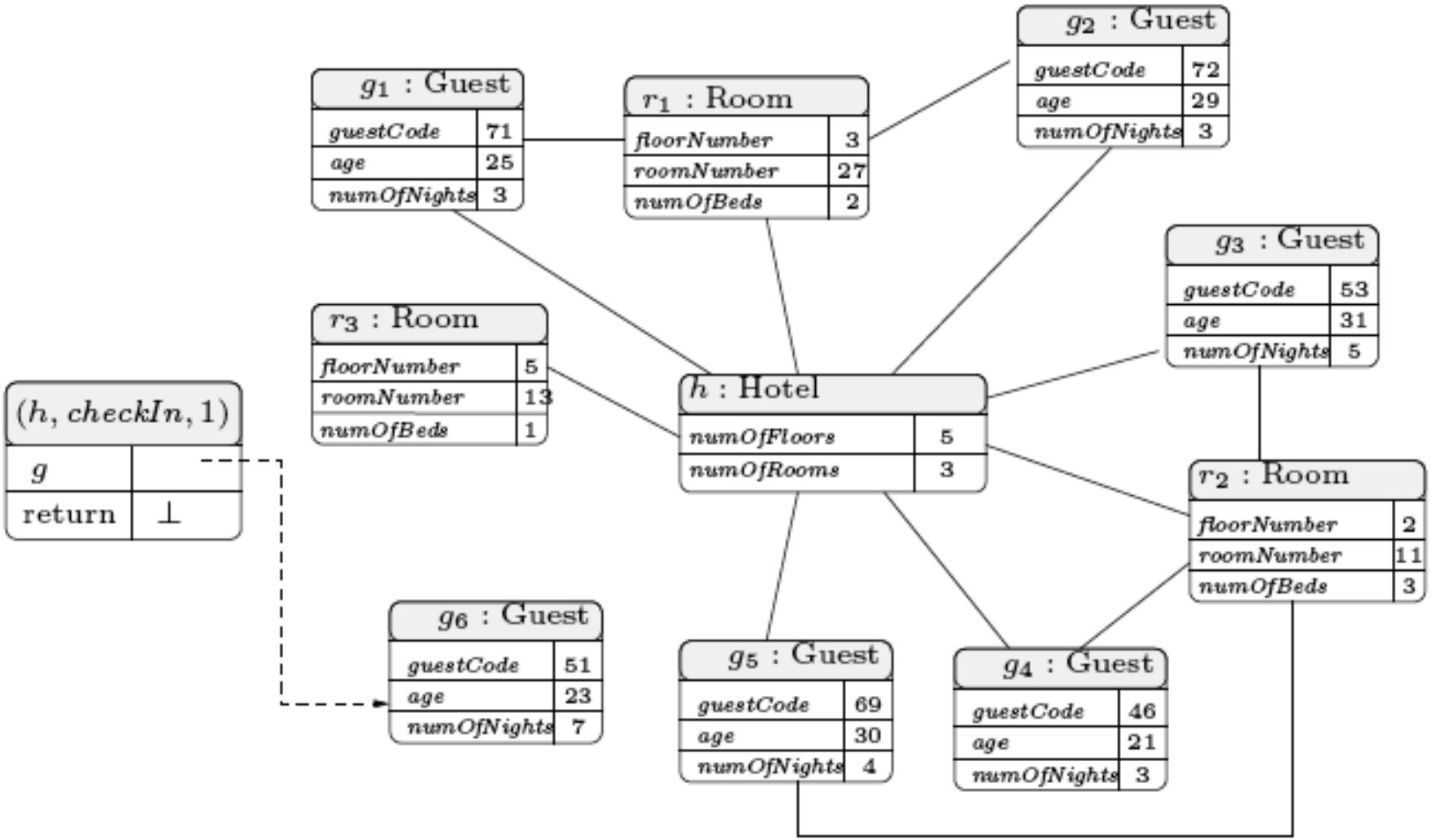
- $dom(\ell) = dom(M.fpars)$,
- $\ell(p) \in \textsc{Val}^{M.fpars(p)}$ for $p \in dom(\ell)$, the value of M's parameters
- $v \in \textsc{Val}_\perp^{M.retty}$, the returned value

Method termination

A method invocation has terminated in the current configuration if it becomes inactive in the next state. On termination, the method has a well-defined value (i.e., different from $\perp$).
If the system transits from configuration (O, E, σ, γ) to $(O', E', \sigma', \gamma')$ then:

$$e \in E - E' \quad \text{implies} \quad \exists v \in Val.\, \gamma(e) = (\ell, v)$$

Configurations: example



Configurations: example

Example 3.3.2. Figure 3.1 depicts a possible configuration of the Hotel model. In particular we have:

$$\begin{aligned} O &= \{h, r_1, r_2, r_3, g_1, g_2, g_3, g_4, g_5, g_6\} \\ E &= \{(h, checkIn, 1)\} \end{aligned}$$

where we have adopted the following abbreviation: $h = (\text{Hotel}, 1)$, $g_i = (\text{Guest}, i)$ and $r_i = (\text{Room}, i)$ for $i \in \mathbb{N}$. The objects show the values of the components ς and γ. For example, for object g_6 we have:

$$\begin{aligned} \varsigma(g_6)(guestCode) &= 51 \\ \varsigma(g_6)(age) &= 23 \\ \varsigma(g_6)(numOfNights) &= 7 \end{aligned}$$

For the other objects, ς can be obtained in a similar way. The γ component for the only active method is $\gamma(h, checkIn, 1) = (g \mapsto g_6,\ \bot)$. □

RWTH AACHEN UNIVERSITY

RWTH AACHEN UNIVERSITY