Foundations of the UML

Lecture 18: Semantics of OCL Constraints

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

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

25. Januar 2010

RWTH AACHEN UNIVERSITY

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.

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.

Temporal expressions

- Static expressions are statements about the current system configuration
- Temporal expressions are statements about configuration sequences (system runs)
- Any static expression is a temporal expression and is valid if the first configuration in the configuration sequence satisfies it
- Temporal expressions can be build using connectives such as negation and disjunction
- $\bigcirc \Phi$ holds in a run if the next configuration in the run satisfies Φ
- A run satisfies $\Phi \,\mathsf{U}\, \Psi$ if contains a configuration satisfying Ψ and all configurations prior to this one satisfy Φ

RWTH AACHEN UNIVERSITY

Temporal expressions

Definition (Temporal expressions)

The syntax of temporal expressions is defined by the grammar:

$$\Phi \ ::= \ \xi \mid \neg\Phi \mid \Phi \wedge \Phi \mid \exists x \in \tau.\, \Phi \mid \bigcirc \Phi \mid \Phi \,\mathsf{U}\, \Phi$$

where ξ is an OCL expression, x is a logical variable, and τ a type.

- $\Diamond \Phi \ = \ \text{true}\,\mathsf{U}\,\Phi$, eventually Φ
- $\Box \Phi \ = \ \neg\Diamond\neg\Phi$, always Φ

RWTH AACHEN UNIVERSITY

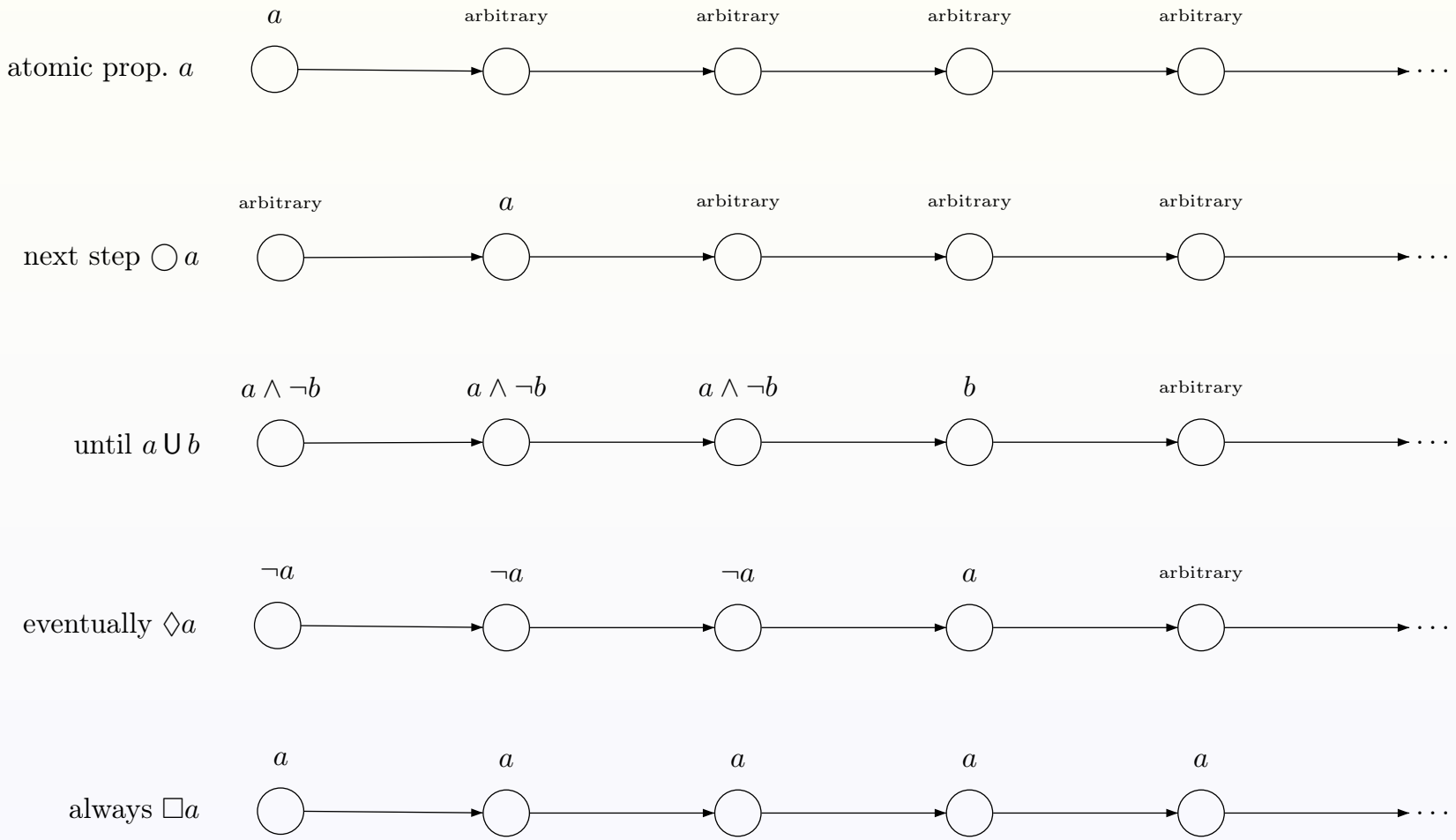
Derived operators

$$
\begin{aligned}
\varphi \vee \psi &\equiv \neg(\neg\varphi \wedge \neg\psi) \\
\varphi \Rightarrow \psi &\equiv \neg\varphi \vee \psi \\
\varphi \Leftrightarrow \psi &\equiv (\varphi \Rightarrow \psi) \wedge (\psi \Rightarrow \varphi) \\
\varphi \oplus \psi &\equiv (\varphi \wedge \neg\psi) \vee (\neg\varphi \wedge \psi) \\
\text{true} &\equiv \varphi \vee \neg\varphi \\
\text{false} &\equiv \neg\,\text{true} \\
\Diamond\varphi &\equiv \text{true}\,\mathsf{U}\,\varphi \qquad \text{“sometimes in the future”} \\
\Box\varphi &\equiv \neg\Diamond\neg\varphi \qquad \text{“from now on for ever”}
\end{aligned}
$$

precedence order: the unary operators bind stronger than the binary ones.
$\neg$ and $\bigcirc$ bind equally strong. U takes precedence over $\wedge$, $\vee$, and $\Rightarrow$

RWTH AACHEN UNIVERSITY

Intuitive semantics



Quantification

- Temporal expression $\exists x \in \tau.\, \Phi$ expresses that Φ holds for at least one alive instance (object or method) referred to by x of type τ.
- For type $\tau = \text{void}, \text{nat}$ or bool, instance $x \in \tau$ is always alive
- For type $\tau = C$ ref, object $x \in \tau$ is alive if it has been created and not yet deallocated (= garbage collected)
- For type $\tau = C.M$ ref, method instance $x \in \tau$ is alive if method M has been invoked and not yet terminated
- We have $\forall x \in \tau.\, \Phi \;\equiv\; \neg\exists x \in \tau.\, \neg\Phi$

RWTH AACHEN UNIVERSITY

Examples

- Along the computation of hotel h, eventually at least one guest will check in:

$$\Diamond\ (\exists m \in h.checkIn \text{ ref}.\, m\ \texttt{alive})$$

  or shortly, $\Diamond\ (\exists m \in h.checkIn \text{ ref})$

- A guest cannot stay forever in hotel h:

$$\Box\ (\forall x \in Guest.\, includes(h.guests, x)\ \Rightarrow\ \Diamond\,(\neg includes(h.guests, x)))$$

- A guest cannot be hosted in more than one room:

$$\Box\,(\forall x \in Guest.\, \exists y, z \in Room \text{ ref}.$$
$$includes(y.guests, x)\ \wedge\ includes(z.guests, x)\ \Rightarrow\ y = z)$$

RWTH AACHEN UNIVERSITY

Configuration automata

Definition (Configuration automaton)

A configuration automaton is a triple $(Cnf, \rightarrow, I)$ where:

1. Cnf is a set of configurations
2. $\rightarrow \subseteq Cnf \times Cnf$, a transition relation
3. $I \subseteq Cnf$ a set of initial configurations with $I \neq \varnothing$

Intuition

- A configuration denotes the state (current objects, method invocations, state of each method and object) of the UML model
- Relation $\rightarrow$ models the evolution from configuration to configuration such that if an active method invocation becomes inactive then it has a well-defined value (i.e., different from $\perp$).

Configration paths

Temporal expressions are interpreted over infinite paths in a configuration automaton $(Cnf, \rightarrow, I)$.

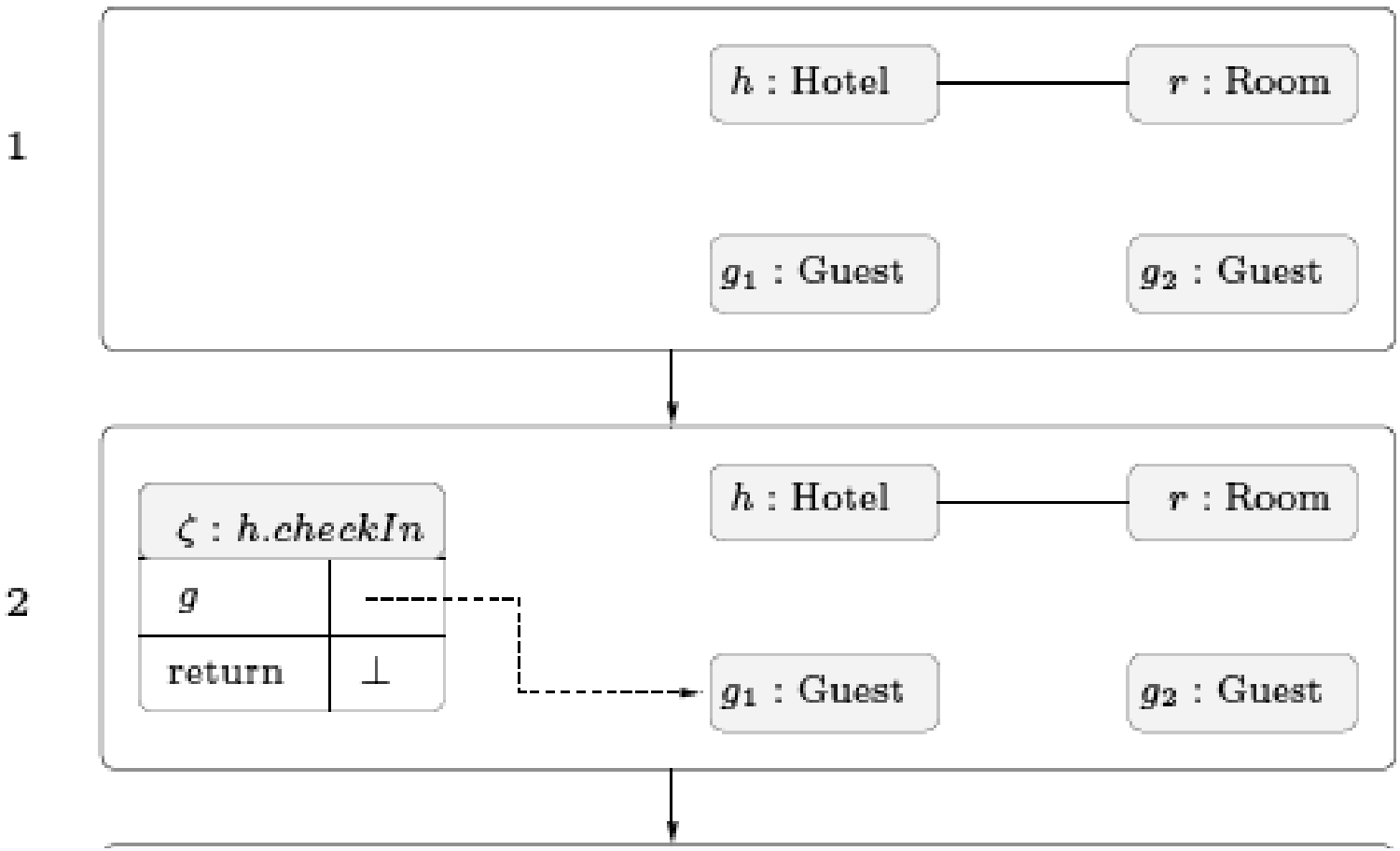
Definition (Path)

An infinite path in $(Cnf, \rightarrow, I)$ is an infinite sequence $\pi = c_0 c_1 c_2 \ldots$ with $c_i \in Cnf$ and $c_i \rightarrow c_{i+1}$ for all $i \in \mathbb{N}$.

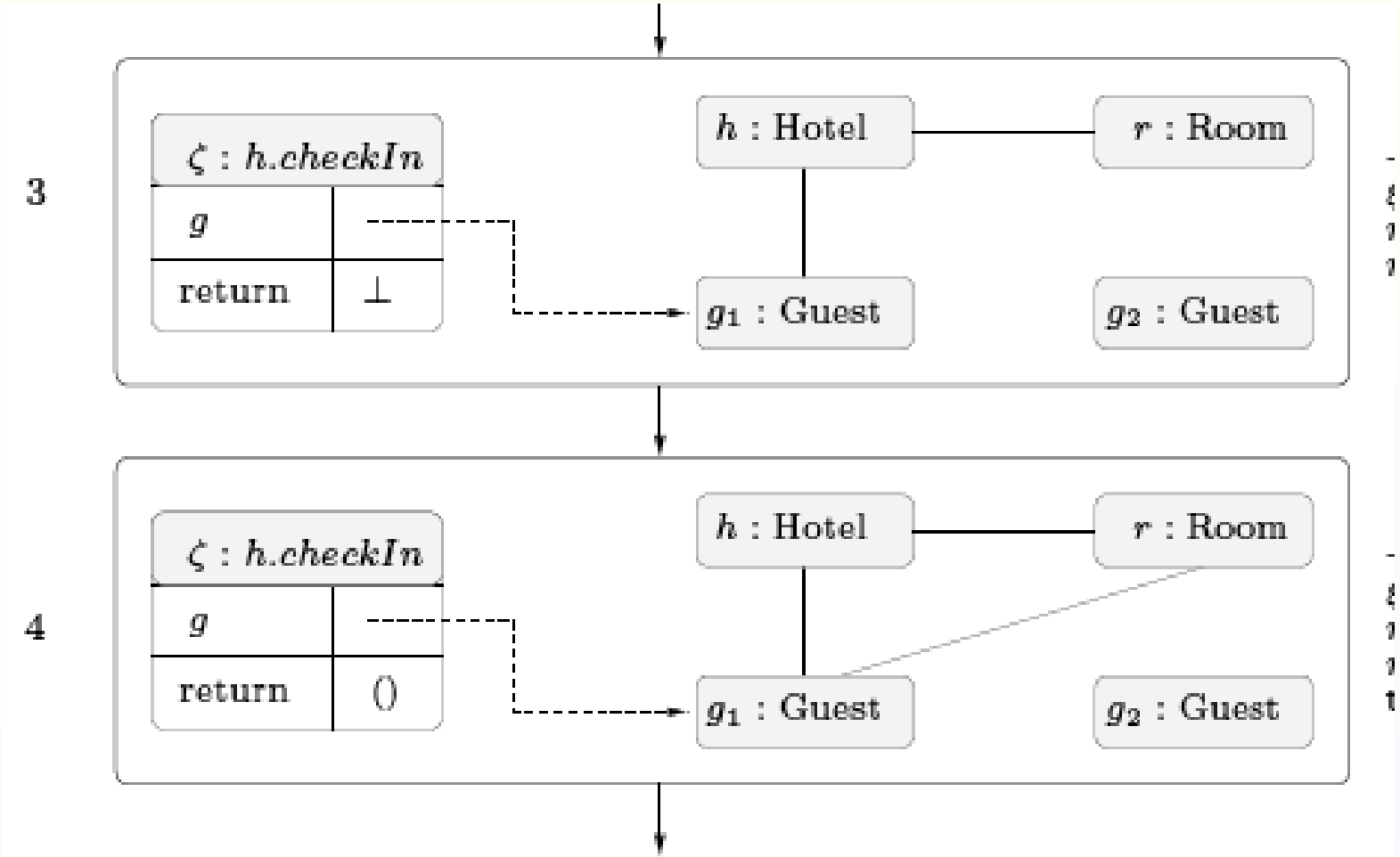
Notation: $\pi[i]$ denotes configuration c_i, and π^k denotes the suffix of π starting from index k, i.e., $\pi^k = \pi[k]\,\pi[k{+}1]\,\ldots$.

RWTH AACHEN UNIVERSITY

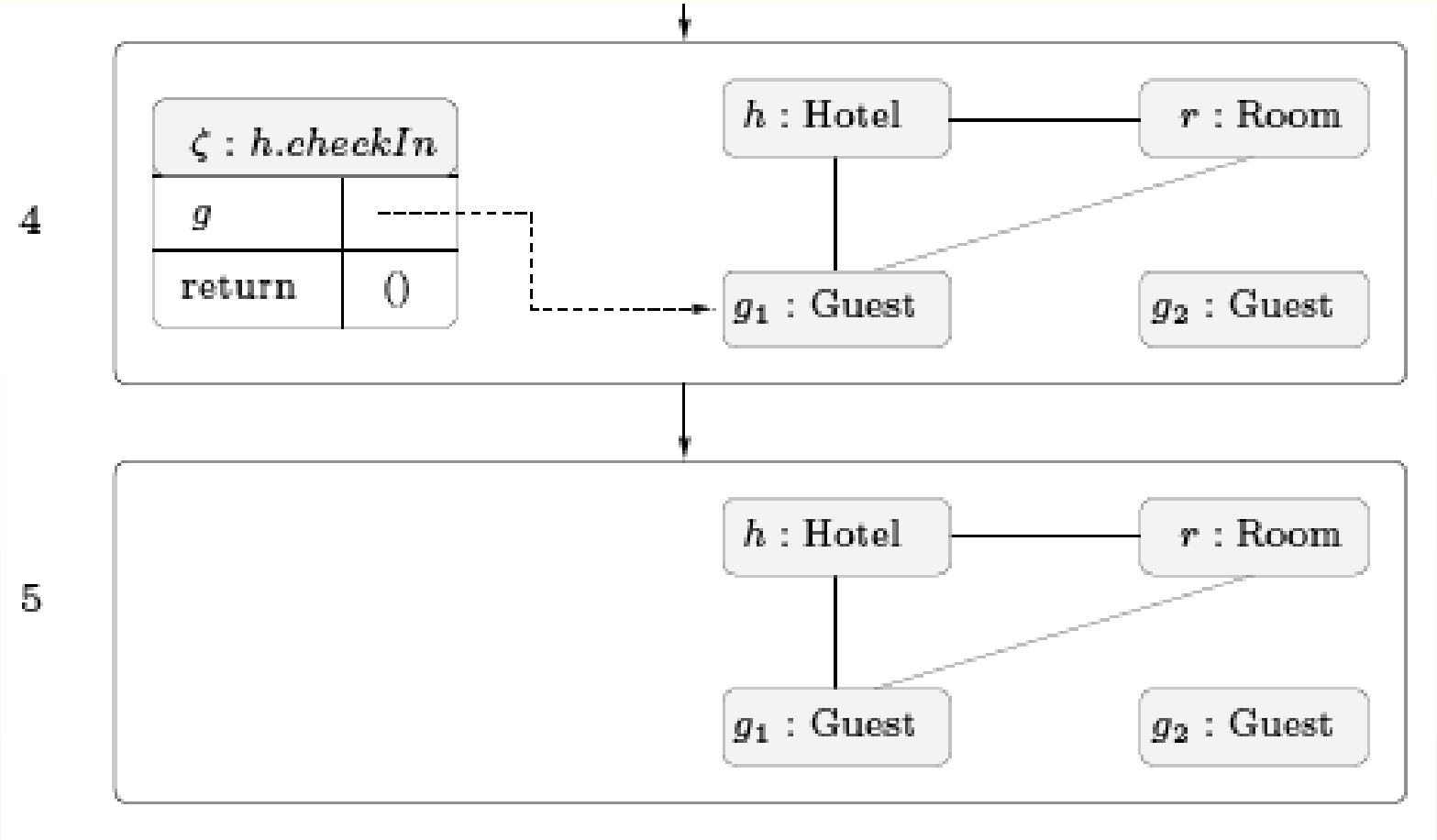
Example (1)


1
h : Hotel
r : Room
g1 : Guest
g2 : Guest
2
ζ : h.checkIn
g
return
⊥
h : Hotel
r : Room
g1 : Guest
g2 : Guest

Example (2)


3
ζ : h.checkIn
g
return
⊥
h : Hotel
r : Room
g1 : Guest
g2 : Guest
4
ζ : h.checkIn
g
return
()
h : Hotel
r : Room
g1 : Guest
g2 : Guest

Example (3)


4
ζ : h.checkIn
g
return
()
h : Hotel
r : Room
g1 : Guest
g2 : Guest
5
h : Hotel
r : Room
g1 : Guest
g2 : Guest

Auxiliary definitions on paths

Definition (New objects and variable valuation along a path)

For path $\pi = c_0 c_1 c_2 \ldots$ let:

1. $N_0 = N \subseteq O_0 \cup E_0$, the objects and events in c_0
2. $N_{i+1} = (O_{i+1} \setminus O_i) \cup (E_{i+1} \setminus E_i)$, i.e., the objects created in $c_i \rightarrow c_{i+1}$ and all events generated in this transition
3. $\theta_i(x) = \begin{cases} \theta(x) & \text{if } \forall k \geqslant i.\, \theta(x) \in O_k \cup E_k \\ \bot & \text{otherwise} \end{cases}$

RWTH AACHEN UNIVERSITY

Semantics of temporal expressions

$(\pi, N, \theta) \models \Phi$ means that for path π, initial set N of new objects (and method invovations), logical valuation θ, the temporal formula Φ holds. It is defined inductively as follows:

$$
\begin{array}{rcl}
\pi, N, \theta \models \xi & \text{iff} & [\![\xi]\!]_{\pi[0], N, \theta} = \mathtt{tt} \\
\pi, N, \theta \models \neg\Phi & \text{iff} & \pi, N, \theta \not\models \Phi \\
\pi, N, \theta \models \Phi \vee \Psi & \text{iff} & \pi, N, \theta \models \Phi \text{ or } \pi, N, \theta \models \Psi \\
\pi, N, \theta \models \bigcirc \Phi & \text{iff} & \pi^1, N, \theta \models \Phi \\
\pi, N, \theta \models \Phi \,\mathsf{U}\, \Psi & \text{iff} & \exists j \in \mathbb{N}.\, \pi^j, N, \theta \models \Psi \text{ and } \forall k < j.\, \pi^k, N, \theta \models \Phi \\
\pi, N, \theta \models \exists x \in \tau.\, \Phi & \text{iff} & \exists v \in \textsc{Val}^\tau \restriction (O_0, E_0).\, \pi, N, \theta[x := v] \models \Phi
\end{array}
$$

where $\textsc{Val}^\tau \restriction (O, E)$ is the subset of $\textsc{Val}^\tau$ alive in (O, E).

RWTH AACHEN UNIVERSITY

OCL invariants

- 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
```

RWTH AACHEN UNIVERSITY

Semantics of OCL invariants

Definition (Semantics of OCL invariants)

The semantics of OCL invariant $\texttt{context}\ C\ \texttt{inv}\ \xi$ is given by a translation Δ onto a temporal expression. Let $y \in \text{Lvar}$ and $dom(C.\text{meths}) = \{M_1, \ldots, M_k\}$. Then Δ is defined by:

$$\Delta(\texttt{context}\ C\ \texttt{inv}\ \xi)$$

$$=$$

$$\Box\ \big(\forall x \in C\ \text{ref}.\ (\neg\exists m_1 \in x.M_1\ \text{ref} \wedge \ldots \wedge \neg\exists m_k \in x.M_k\ \text{ref})\ \Rightarrow\ \delta_{x,y,[\,]}(\xi)\big)$$

Intuition

The condition ξ must hold in any state where no method in $dom(C.\text{meths})$ is active. During the execution of such method, some configuration may violate the condition ξ.

Example

Example 3.5.3. Consider again the OCL invariant in Example 3.4.2 assuming that the collection *guests* is a bag.

$$\texttt{context Hotel inv}$$
$$rooms.guests = guests$$

Recalling the considerations of Example 3.5.2 on set equality, we have:

$$\begin{aligned}\delta(rooms.guests = guests) &= EqList(sort(\delta(rooms.guests)), sort(\delta(guests)))\\ &= EqList(sort(flat(x.rooms.guests)), sort(x.guests)).\end{aligned}$$

We can embed the resulting BOTL expression in the invariant template taking into account that the class Hotel has two methods, i.e., *checkIn* and *checkOut*:

$$\begin{aligned}\mathsf{G}[\forall x \in \text{Hotel ref} : (&\neg\exists m \in x.checkIn\ \text{ref} : \mathsf{tt} \wedge \neg\exists m' \in x.checkOut\ \text{ref} : \mathsf{tt})\\ &\Rightarrow\ EqList(sort(flat(x.rooms.guests)), sort(x.guests))].\end{aligned}$$

Note that apart from the use of the more natural equal sign instead of *EqList*, the resulting invariant coincides with the invariant in Example 3.4.2. □

RWTH AACHEN UNIVERSITY

OCL pre- and postconditions

- 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 evaluating 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

Extended OCL preconditions

Main complication

For any ξ@`pre` expression in the postcondition, we have to remember its value on evaluating the precondition. This is established using auxiliary variables: for each expression $\xi@_i\texttt{pre}$ we use the auxiliary logical variable $u_i \in \textsc{Lvar}$.

Extended precondition

Extended precondition = precondition + auxiliary variables $\{u_1, \ldots, u_n\}$. Formally,

$$\xi_{pre}^{ext} \;=\; \delta(\xi_{pre}) \wedge \bigwedge_{\xi@_i\texttt{pre}\in\xi_{post}} u_i = \delta(\xi)$$

Example

RWTH AACHEN UNIVERSITY

Semantics of OCL pre- and postconditions

Definition (Semantics of pre- and postconditions)

The semantics of a pre- and postcondition is given by a mapping Δ onto a temporal expression, and is defined by:

$$\Delta(\texttt{context}\ C :: M(\vec{p})\ \texttt{pre}\ \xi_{pre}\ \texttt{post}\ \xi_{post})$$

$$=$$

$$\forall u_1 \in \tau_1, \ldots, u_n \in \tau_n . \forall z \in C\ \text{ref} . \forall m \in z . M\ \text{ref} .$$
$$\Box\left(m\ \texttt{new} \wedge \xi_{pre}^{ext} \Rightarrow m\ \texttt{alive}\,\mathsf{U}\,(m\ \texttt{alive} \wedge \bigcirc \neg(m\ \texttt{alive}) \wedge \delta(\xi_{post}))\right)$$

Example

Example 3.5.3. Consider again the OCL invariant in Example 3.4.2 assuming that the collection *guests* is a bag.

$$\texttt{context Hotel inv}$$
$$rooms.guests = guests$$

Recalling the considerations of Example 3.5.2 on set equality, we have:

$$\begin{aligned}\delta(rooms.guests = guests) &= EqList(sort(\delta(rooms.guests)), sort(\delta(guests)))\\ &= EqList(sort(flat(x.rooms.guests)), sort(x.guests)).\end{aligned}$$

We can embed the resulting BOTL expression in the invariant template taking into account that the class Hotel has two methods, i.e., *checkIn* and *checkOut*:

$$\begin{aligned}\mathsf{G}[\forall x \in \text{Hotel ref} : (&\neg\exists m \in x.checkIn\ \text{ref} : \mathsf{tt} \wedge \neg\exists m' \in x.checkOut\ \text{ref} : \mathsf{tt})\\ &\Rightarrow\ EqList(sort(flat(x.rooms.guests)), sort(x.guests))].\end{aligned}$$

Note that apart from the use of the more natural equal sign instead of *EqList*, the resulting invariant coincides with the invariant in Example 3.4.2. □

RWTH AACHEN UNIVERSITY

RWTH AACHEN UNIVERSITY