UNIVERSITÀ DI PARMA
CORSO DI INFORMATICA

Sistema di gestione delle prenotazioni di un albergo

Student name: *Singh Karan Preet*

Course: *specifica e verifica dei sistemi software* – Professor: *Maximiliano Cristia'*
Due date: *February 15th, 2020*

Introduzione

La seguente specifica formalizzata in notazione Z, permette di effettuare ed eventualmente gestire le prenotazioni di un albergo.
Le operazioni principali che e' possibile eseguire sono:

- Prenotazione di una camera: il cliente viene aggiunto all'insieme dei clienti, la camera viene aggiunta all'insieme delle camere occupate ed infine viene creata un associazione fra cliente e camera.
- Cancellazione di una prenotazione effettuata: il cliente viene eliminato dall'insieme dei clienti, la camera viene eliminata dall'insieme delle camere occupate ed infine si aggiorna l'insieme delle prenotazioni eliminando la coppia cliente-camera.
- Modifica di una prenotazione: è possibile modificare la camera attuale del cliente con una camera nuova, a patto che quest'ultima non sia attualmente occupata da un altro cliente.
- Stampa delle prenotazioni: fornita una lista di clienti, è possibile visualizzare le camere in cui questi alloggiano.

Tipi dichiarati

Dichiaro un tipo basico che chiamo *CLIENTS*, questo rappresenta l'identificativo del cliente (ad esempio il cognome). Dichiaro un secondo tipo basico chiamato *ROOMS*, questo identifica una camera d'albergo, quindi i valori che può assumere sono: room1, room2, room3 etc. L'ultimo tipo dichiarato invece è *MESSAGE*, il quale mi fornisce una sequenza di messaggi che mi indicano se una determinata operazione è andata a buon fine oppure no.

$[CLIENTS]$

$[ROOMS]$

$MESSAGE ::= Successfull \mid ClientPresent \mid RoomBooked \mid NoClient \mid NoRoom \mid RoomChanged \mid InsertClient \mid WrongClientIsInserted$

Schema: HOTEL

Hotel

$$clients : \mathbb{P}\, CLIENTS$$
$$bookedrooms : \mathbb{P}\, ROOMS$$
$$reserved : CLIENTS \pfun ROOMS$$

- *clients* rappresenta l'insieme dei clienti che alloggiano nell'albergo. E' definito come un insieme del tipo *[CLIENTS]*
- *bookedrooms* rappresenta le camere dell'albergo che sono state prenotate, perciò non disponibili. E' definito come un insieme di tipo *[ROOMS]*
- *reserved* è una funzione parziale che associa ad ogni cliente una camera d'albergo.

Schema: InitHotel

InitHotel

Hotel

$clients = \emptyset$
$bookedrooms = \emptyset$
$reserved = \emptyset$

Lo schema *InitHotel* mi rappresenta lo stato iniziale dell'Albergo in cui *clients*, *bookedroom* e *reserved* sono tutti uguali all'insieme vuoto (dato che all'inizio non ho nessun cliente prenotato).

Schema: InvHotel

InvHotel

Hotel

$clients = \text{dom}\, reserved$
$bookedrooms = \text{ran}\, reserved$

Lo schema *InvHotel* mi rappresenta l'invariante di stato, in cui *clients*, ovvero l'insieme dei clienti, deve appartenere al dominio di *reserved* e *bookedrooms*, ovvero l'insieme delle camere occupate deve appartenere al rango di *reserved*. Questo perchè il cliente è definito tale solo se è associato ad una camera e viceversa una camera è definita occupata solo se associata ad un cliente.

Schema: BookingRoomOK

BookingRoomOK

$\Delta Hotel$
$client? : CLIENTS$
$room? : ROOMS$
$msg! : MESSAGE$

$client? \notin clients$
$room? \notin bookedrooms$
$clients' = clients \cup \{client?\}$
$bookedrooms' = bookedrooms \cup \{room?\}$
$reserved' = reserved \cup \{client? \mapsto room?\}$
$msg! = Successfull$

Lo schema *BookingRoomOK* mi rappresenta l'operazione di prenotazione. Prende in input le variabili *client?* e *room?* che mi rappresentano rispettivamente il cliente che si vuole prenotare e la camera che verrà associata a quel cliente. La variabile *msg!* è una variabile di output per rappresentare un messaggio di avviso.

- La variabile di input *client?* non deve appartenere a *clients*, ovvero non deve appartenere al dominio di *reserved* dato che essendo un nuovo cliente non ha ancora una camera associata.

- La variabile di input *room?* non deve appartenere a *bookedrooms*, dato che non posso assegnare al cliente una camera già prenotata.

- Tutti gli insiemi *clients*, *bookedrooms*, *reserved*, vengono aggiornati con un operazione di unione insiemistica.

Schema: ClientAlreadyPresent

ClientAlreadyPresent

$\Xi Hotel$
$client? : CLIENTS$
$msg! : MESSAGE$

$client? \in clients$
$msg! = ClientPresent$

Negando la prima condizione relativa al predicate part, ottengo lo schema ClientAlreadyPresent che mi definisce il comportamento della specifica nel momento in cui viene effettuata una prenotazione con un cliente già presente nell'insieme dei clienti. Il messaggio di errore mostrato è: ClientPresent.

Schema: RoomAlreadyBooked

RoomAlreadyBooked

$\Xi Hotel$
$room? : ROOMS$
$msg! : MESSAGE$

$room? \in bookedrooms$
$msg! = RoomBooked$

Negando la seconda condizione relativa al predicate part, ottengo lo schema RoomAlreadyBooked che mi definisce il comportamento della specifica nel momento in cui viene effettuata una prenotazione con una camera già presente nell'insieme delle camere occupate. Il messaggio di errore mostrato è: RoomBooked.

possiamo quindi definire l'operazione di Prenotazione nel seguente modo:

BookingRoom == BookingRoomOK $\vee$ *ClientAlreadyPresent* $\vee$ *RoomAlreadyBooked*

Schema: BookingCancelOK

BookingCancelOK

$\Delta Hotel$
$client? : CLIENTS$
$room? : ROOMS$
$msg! : MESSAGE$

$client? \in clients$
$room? \in bookedrooms$
$clients' = clients \setminus \{client?\}$
$bookedrooms' = bookedrooms \setminus \{room?\}$
$reserved' = \{client?\} \lhd reserved$
$msg! = Successfull$

Lo schema BookingCancelOK mi definisce l'operazione di cancellazione di una prenotazione. Prende in input due varibaili: il cliente che si vuole cancellare e la sua rispettiva camera. L'insieme dei clienti e delle camere occupate vengono aggiornati con un operazione di sottrazione insiemistica, mentre l'insieme delle prenotazioni viene aggiornato attraverso un operazione di antirestrizione di dominio; che va ad eliminare dall'insieme *reserved* la coppia che ha come primo valore quello della variabile di input *client?*.

Schema: ClientNotListed

ClientNotListed

$\Xi Hotel$
$client? : CLIENTS$
$msg! : MESSAGE$

$client? \notin clients$
$msg! = NoClient$

Lo schema ClientNotListed mi definisce il comportamento della specifica nel caso si tenti di cancellare un cliente che non è presente nell'insieme dei clienti. Mostra un messaggio di errore: NoClient.

Schema: RoomNotListed

RoomNotListed

$\Xi Hotel$
$room? : ROOMS$
$msg! : MESSAGE$

$room? \notin bookedrooms$
$msg! = NoRoom$

Lo schema RoomNotListed mi definisce il comportamento della specifica nel caso si tenti di cancellare una camera che non è presente nell'insieme delle camere prenotate. Mostra un messaggio di errore: NoRoom.

possiamo quindi definire l'operazione di Cancellazione nel seguente modo:

BookingCancel == BookingCancelOK $\vee$ *ClientNotListed* $\vee$ *RoomNotListed*

Schema: ChangeRoomOK

ChangeRoomOK

$\Delta Hotel$
$client? : CLIENTS$
$room? : ROOMS$
$newroom? : ROOMS$
$msg! : MESSAGE$

$client? \in clients$
$newroom? \notin bookedrooms$
$room? \in bookedrooms$
$clients' = clients$
$bookedrooms' = (bookedrooms \setminus \{room?\}) \cup \{newroom?\}$
$reserved' = reserved \oplus \{client? \mapsto newroom?\}$
$msg! = RoomChanged$

Lo schema ChangeRoomOK mi definisce l'operazione di modifica della prenotazione. E' possibile modificare la camera del cliente con una nuova a patto che la nuova camera non sia già occupata. L'operazione prende in input:

- il cliente al quale si deve effettuare la modifica
- la camera attuale al quale il cliente è associato
- la nuova camera che si vuole associare al cliente

L'insieme dei clienti rimane invariato, dato che non si vuole aggiungere od eliminare nessun cliente. L'insieme delle camere occupate viene aggiornato sottraendo dall'insieme il valore *room?* ed effettuando successivamente un unione insiemistica con il valore di *newroom?*.
L'insieme delle prenotazioni viene aggiornato con l'operatore di overriding relazione, che prima mi esegue un operazione di antirestrizione di dominio per eliminare la coppia corrente e successivamente aggiorna l'insieme con un operazione di unione insiemistica.

Possiamo definire lo schema di modifica della prenotazione nel seguente modo:

ChangeRoom == ChangeRoomOK $\vee$ *ClientNotListed* $\vee$ *RoomAlreadyBooked*

Schema: GetClientRoomOK

GetClientRoomOK

Ξ*Hotel*
client? : $\mathbb{P}$ *CLIENTS*
getRoom! : *CLIENTS* $\pfun$ *ROOMS*
msg! : *MESSAGE*

---

client? $\neq \emptyset$
client? $\subseteq$ *clients*
getRoom! = *client*? $\lhd$ *reserved*
msg! = *Successfull*

L'ultima operazione definita in questa specifica è la GetClientRoom. Preso in input un insieme di clienti, restituisce le rispettive camere in cui questi alloggiano.
L'operazione non va a modificare lo stato dell'Hotel ma si occupa semplicemente di restituire in output un insieme di coppie di tipo client e room.
La variabile di output *getRoom!* è uguale alla restrizione di dominio fra *reserved* e *client?*. Questo implica che a *getRoom!* vengono assegnate tutte le coppie dell'insieme *reserved* che hanno come prima componente gli elementi dell'insieme di *client?* fornito in input.

Schema: WrongClientInserted

WrongClientInserted

Ξ*Hotel*
client? : $\mathbb{P}$ *CLIENTS*
msg! : *MESSAGE*

---

client? $\not\subseteq$ *clients*
msg! = *WrongClientIsInserted*

Lo schema WrongClientInserted definisce il comportamento della specifica nel caso in cui almeno uno dei clienti inseriti non sia un sottoinsieme di *clients*, in tal caso il seguente schema produrrà come output un messaggio di errore:

Possiamo quindi definire l'operazione di getClientRoom nella seguente maniera:

GetClientRoom == GetClientRoomOK $\vee$ *WrongClientInserted*

SIMULAZIONI

dopo aver tradotto la specifica Z in setlog è possibile effettuare le simulazioni per vedere come le operazioni cambiano gli stati degli schemi sopra definiti.

Simulazione BASICA. Le operazioni scelte per effettuare la simulazione basica sono: BookingRoom e BookingCancel.
Gli stati della simulazione saranno i seguenti:

- initHotel prende come stato iniziale una variabile S0.
- bookingRoom prende come stato iniziale la stessa variabile definita in initHotel, cioè S0 e ritorna lo stato finale dell'operazione con la variabile S1.
- bookingCance prende come stato iniziale la variabile finale di bookingRoom, cioè S1 e ritorna lo stato finale con la variabile S2.

La simulazione viene eseguita nel seguente modo (tutti i valori passati sono delle costanti):

- Invoco initHotel passandogli S0.
- Invoco bookingRoom passandogli: S0, il cliente da aggiungere all'insieme dei clienti, la camera da aggiungere all'insieme delle camere occupate, un messaggio di output e lo stato finale S1.
- Invoco bookingCancel passandogli: S1, il cliente da eliminare dall'insieme dei clienti, la camera da eliminare dall'insieme delle camere occupate, un messaggio di output e lo stato finale S2.

```
{log}=> initHotel(S0) & bookingRoom(S0,singh,room1,successfull,S1) &
                bookingCancel(S1,singh,room1,successfull,S2).

S0 = {[clients,{}],[bookedrooms,{}],[reserved,{}]},
S1 = {[clients,{singh}],[bookedrooms,{room1}],[reserved,{[singh,room1]}]},
S2 = {[clients,_N2],[bookedrooms,_N1],[reserved,{}]}
Constraint: set(_N2), subset(_N2,{singh}), singh nin _N2, set(_N1), subset(_N1,{room1}), room1 nin _N1
```

Come si vede dalla simulazione, parto con lo stato iniziale S0 in cui gli insiemi clients, bookedrooms e reserved sono vuoti.
Lo stato S1, va a modificare l'insieme a seguito dell'operazione di bookingRoom aggiungendo il cliente e la camera passati come input. Lo stato S2, va a ad effettuare una cancellazione del cliente e della camera occupata facendo ritornare l'insieme a quello di partenza (dato che avevamo inserito un solo cliente).

Simulazione SIMBOLICA. Nella simulazione simbolica al posto di eseguire le operazioni utilizzando delle costanti, utilizziamo delle variabili. In questo modo setlog sarà in grado di fornire un risultato più generale delle operazioni eseguite.
Per fare la simulazione simbolica, nel nostro caso, utilizziamo l'operazione di bookingRoom e changeRoom.

```
{log}=> initHotel(S0) & bookingRoom(S0,N1,R1,M1,S1) & changeRoom(S1,N2,R2,R3,M2,S2).

S0 = {[clients,{}],[bookedrooms,{}],[reserved,{}]},
M1 = successfull,
S1 = {[clients,{N1}],[bookedrooms,{R1}],[reserved,{[N1,R1]}]},
N2 = N1,
R2 = R1,
M2 = roomChanged,
S2 = {[clients,{N1}],[bookedrooms,{R3/_N1}],[reserved,{[N1,R3]}]}
Constraint: R1 neq R3, set(_N1), subset(_N1,{R1}), R1 nin _N1
```

Quello sopra dimostrato è il primo risultato ottenuto dalla simulazione.
Gli stati della simulazione sono i seguenti:

- initHotel prende come stato iniziale una variabile S0.
- bookingRoom prende come stato iniziale la stessa variabile definita in initHotel, cioè S0 e ritorna lo stato finale dell'operazione con la variabile S1.
- changeRoom prende come stato iniziale la variabile finale di bookingRoom, cioè S1 e ritorna lo stato finale con la variabile S2.

La simulazione viene eseguita nel seguente modo (tutti i valori passati sono delle variabili):

- Invoco initHotel passandogli S0.
- Invoco bookingRoom passandogli: S0, una variabile che rappresenta il cliente da aggiungere all'insieme dei clienti, una variabile che rappresenta la camera da aggiungere all'insieme delle camere occupate, una variabile di output e lo stato finale S1.
- Invoco changeRoom passandogli: S1, una variabile che indica il cliente da modificare, una seconda variabile che indica la camera corrente del cliente, una terza variabile che indica la nuova camera da associare al cliente, una variabile di output e lo stato finale S2.

Dal risultato dell'output possiamo vedere che partendo da initHotel, con l'operazione di bookingRoom l'insieme viene aggiornato con le variabili N1 e R1 che indicano rispettivamente il cliente e la camera associata al cliente.
S2 mostra lo stato finale dopo aver eseguito l'operazione di changeRoom, in cui l'insieme dei clienti rimane invariato, l'insieme delle camere occupate viene aggiornato con la variabile R3, mentre l'insieme delle prenotazioni viene aggiornato con la nuova coppia N1 e R3.

INVARIANTI DI STATO

La specifica di HotelBooking preserva due invarianti di stato definite nello schema invHotel precedentemente descritto.
In setlog posso verificare se uno stato S0 preserva le invarianti di stato nel seguente modo:

```
{log}=> S0 = {[clients,{singh}],[bookedrooms,{room1}],[reserved,{[singh,room1]}]} & invHotel(S0).

S0 = {[clients,{singh}],[bookedrooms,{room1}],[reserved,{[singh,room1]}]}
```

Se l'output dell'operazione restituisce come risultato S0, vuol dire che le invarianti di stato sono preservate.
Supponendo di avere un operazione definita come OP e un invariante di stato definita come INV, per dimostrare che l'operazione preserva l'invariante di stato devo innanzitutto dimostrare che:

$$\text{INV} \wedge OP \Rightarrow INV'(1)$$

In setlog invece dobbiamo dimostrare che la negazione di (1) è insoddisfacibile, ovvero:

$$\text{INV} \wedge OP \neg\, INV'$$

Nel nostro caso vogliamo dimostrare che l'invariante di stato *bookedrooms* = ran *reserved* è valida per tutte le operazioni di HotelBooking:

```
OPERAZIONE DI BOOKING ROOM:

S0={[clients,C],[bookedrooms,B],[reserved,R]} &
S1={[clients,C_],[bookedrooms,B_],[reserved,R_]} &
ran(R,B) &
bookingRoom(S0,C,R,M,S1_) &
nran(R_,B_).

OPERAZIONE DI BOOKING CANCEL:

S0={[clients,C],[bookedrooms,B],[reserved,R]} &
S1={[clients,C_],[bookedrooms,B_],[reserved,R_]} &
ran(R,B) &
bookingCancel(S0,C,R,M,S1_) &
nran(R_,B_).

OPERAZIONE DI CHANGE ROOM:

S0={[clients,C],[bookedrooms,B],[reserved,R]} &
S1={[clients,C_],[bookedrooms,B_],[reserved,R_]} &
ran(R,B) &
changeRoom(S0,C,R,R2,M,S1_) &
nran(R_,B_).

OPERAZIONE DI GET CLIENT ROOM:

S0={[clients,C],[bookedrooms,B],[reserved,R]} &
S1={[clients,C_],[bookedrooms,B_],[reserved,R_]} &
```

```
ran(R,B) &
getClientRoom(S0,C,R,M,S1_) &
nran(R_,B_).
```

Facendo l'esecuzione con setlog di ognuna di queste operazioni, ottengo come risultato 'no'; questo implica che ognuna delle operazioni sopra elencate preserva l'invariante di stato.