UNIVERSITÀ DI PARMA
CORSO DI LAUREA TRIENNALE IN INFORMATICA

Specifica in notazione Z di una parte semplificata di un piccolo areoporto

Author:
Lorenzo Galafassi

Dipartimento di Scienze Matematiche, Fische ed Informatiche

March 11, 2021

Breve introduzione

cos'è una specifica?

Con il termine specifica, molto spesso si intende quel documento che, appunto, specifica in modo chiaro, completo e preciso quali sono le caratteristiche che il software/sistema che dobbiamo produrre e sviluppare deve avere.

cos'è una specifica formale?

Una specifica formale è una specifica che utilizza il linguaggio matematico per definire tutte le caratteristiche del nostro software/sistema. Definire questo documento usando "la matematica" non solo ci consente di eliminare tutte quelle ambiguità che possono essere (purtroppo) presenti in specifiche non formali, ma ci permette anche di dimostrare e quindi verificare alcune proprietà (ad esempio di correttezza) sul nostro software/sistema.

cos'è la notazione Z?

La notazione Z è un linguaggio di specifica formale, basato principalmente sulla teoria degli insiemi e sulla logica del primo ordine. Attraverso questa notazione possiamo definire quelli che vengono chimati "schemi", cioè descrizioni dello stato di alcuni "oggetti" ed operazioni. Questi schemi sono la parte fondante e fondamentale della notazione Z, attraverso i quali andremo a specificare tutte le caratteristiche del nostro sistema.

Requisiti

Cosa sono i requisiti?

I requisiti di un software/sistema sono quei vincoli e quelle caratteristiche che il software/sistema dovrà avere e rispettare. In sostanza non sono altro che "quelle cose" che andremo a scrivere sulla nostra specifica (traducendole, naturalmente, in base al "linguaggio" utilizzato nella specifica); infatti, seguendo sia il buon senso che "le regole" dell'ingegneria del software, prima raccogliamo i requisiti del nostro sistema (raccolti dal destinatario del software/sistema, ad esempio un cliente) e poi con questi ultimi scriviamo la nostra specifica.

requisiti del sistema areoportuale

Come da titolo, ho voluto descrivere una piccola parte (e semplificata) del funzionamento di un areporto. Quest'ultimo dobbiamo immaginarlo come un piccolo aereporto che avrà un solo aereo con pochi posti e che offre il viaggio per poche destinazioni.

Avremo un registro dei passeggeri che ci permetterà di tenere traccia dei passeggeri dell'aereporto che:

- potranno **comprare un biglietto** tra quelli disponibili
- potranno **fare il checkin** per consegnare le valigie che verranno messe in un deposito
- nel caso lo vorranno potranno **cancellare la prenotazione** del volo
- potranno subire controlli ed **essere identificati** dalle forze di polizia dell'areoporto

* sottolineato = schema di stato * **grassetto** = operazione

Tipi, assiomi ed enumerazioni

Passaporto

Il tipo PASSPORT indica, appunto, l'insieme dei passaporti. Per semplicità (e non necessità) non ne indicheremo la concreta struttura.

$$[PASSPORT]$$

Orario

Il tipo TIME indica l'orario di un volo. Un elemento di questo insieme, per semplicità, indica solo l'ora (intesa come le 16, le 21, ecc...) di un volo e va dall'una di notte fino a mezzanotte.

$$TIME == \{hour : \mathbb{Z} \mid 1 \leq hour \leq 24\}$$

Valigia più piccola

Questo assioma indica la valigia più piccola esistente, ed è utilizzato per definire correttamente un tipo (per maggiori dettagli leggere il punto 3.1.2).

$$\begin{array}{|l} suitcaseSmallerSize : \mathbb{Z} \times \mathbb{Z} \\ \hline suitcaseSmallerSize = 50 \mapsto 80 \end{array}$$

Valigia

Questo tipo indica le informazioni riguardanti una valigia. Questa è l'informazione principale di una valigia e quest'ultima viene definita proprio come un elemento di questo insieme. Questa coppia di numeri interi deve essere maggiore di una costante per il semplice motivo che una valigia, per essere identificata tale, non può essere troppo piccola da un punto di vista logico (altri motivi non ce ne sono, una questione solo "filosofica" quindi). All'interno di questo insieme viene aggiunto anche la coppia (-1,-1) dato che, come si vedrà più avanti, sarà un valore "flag" per le nostre valigie che indicherà una situazione ben precisa.

$$\begin{array}{l} SUITCASESIZE == \{pair : \mathbb{Z} \times \mathbb{Z} \mid \\ \quad pair.1 \geq suitcaseSmallerSize.1 \land pair.2 \geq suitcaseSmallerSize.2 \land pair = (0-1) \mapsto (0-1) \\ \} \end{array}$$

Numero di posti sull'aereo

Questo assioma indica il numero massimo di posti a sedere sull'aereo, che equivale anche al numero di biglietti prodotti per ogni destinazione. Come detto in precendenza esiste un solo aereo che, quindi, effettua un volo e poi torna indietro per effettuarne un altro

$$planeSeatsNumber : \mathbb{Z}$$
$$\hline$$
$$planeSeatsNumber = 50$$

Posto sull'aereo

Il tipo PLANESEATID indica l'insieme degli identificativi dei posti presenti sull'unico aereo presente in areoporto. In questo inseme viene aggiunto anche il valore -1, che, come nel caso del tipo SUITCASESIZE e come vedremo più avanti indicherà una situazione e uno stato per il passeggero ben preciso.

$$PLANESEATID == \{ID : \mathbb{Z} \mid 1 \leq ID \leq planeSeatsNumber\} \cup \{(0 - 1)\}$$

Destinazione

Questa enumerazione o "free type" indica l'insieme delle destinazioni. Una destinazione è una parte delle informazioni che compongono un biglietto aereo. Viene anche specificata la costante "NULL" sempre per indicare uno stato preciso, come vedremo, del passeggero.

$$DESTINATION ::= Milano \mid Roma \mid Firenze \mid Torino \mid Bologna \mid NULL$$

Biglietti aerei

Ogni variabile di questo tipo/insieme, indica i biglietti aerei disponibili in quel momento nell'areoporto. Dato che ho deciso che i biglietti disponibili sarebbero stati tutte le coppie ordinate (posto in aereo, destinazione) allora ho utilizzato una relazione binaria in modo tale che i valori che una variabile di questo tipo può assumere sono i sottoinsiemi dell'insieme di queste coppie ordinate (dato che questa variabile nel tempo "muta", come vedremo, perdendo o guadagnando biglietti).

$$FLIGHTS == PLANESEATID \leftrightarrow DESTINATION$$

Biglietto aereo "vuoto"

Questo assioma indica un biglietto vuoto/bianco e viene "assegnato", inteso come valore, ad un passeggero nel momento in cui cancella il volo.

$$blankTicket : PLANESEATID \times DESTINATION$$
$$\hline$$
$$blankTicket = (0 - 1) \mapsto NULL$$

Messaggi di output delle operazioni

Questa enumerazione o "free type" indica l'insieme dei messaggi di output delle operazioni. utili per indicare lo stato finale queste ultime.

$$OPERATIONMESSAGE ::= BuyTicketOk \mid FlightTicketsSoldOut \mid WantedTicketNotAvalaible \mid GiveSuitcasesOk \mid MustBuyTicket \mid TooManyCheckin \mid TooBigSuitcases \mid NoMoreSpace \mid DescribedPassengersStopped \mid FlightCanceled$$

Informazioni sul passeggero

Questo schema rappresenta tutte le informazioni riguardanti un passeggero. Se si nota bene, il tipo PLANESEATID può avere il valore -1, il tipo DESTINATION il valore "NULL" e il tipo SUITCASESIZE il valore (-1,-1), ma tutto ciò è previsto dalla specifica, dato che questo registro deve essere visto come un qualcosa di virtuale (come ad esempio un Database) che raccoglie tutte le informazioni possibili di un passeggero (al contrario del prossimo schema che invece ha una natura più materiale). Per quanto riguarda la scelta di mettere una sequenza di valigie, è necessario per il fatto che un passeggero possa avere valigie di dimensioni uguali.
Questo schema è inserito in questa categoria, dato che deve essere usato solo come tipo e non come schema "globale" sul quale fare delle operazioni.

PassengerInfo

$$flightTicket : PLANESEATID \times DESTINATION$$
$$suitcases : \text{seq}\, SUITCASESIZE$$

Deposito delle valigie

Questo schema rappresenta il deposito nel quale verranno depositate le eventuali valigie dei passeggeri. Raccoglie varie informazioni utilizzate per la gestione stessa del deposito e per il controllo sulle valigie che vogliono essere depositate. All'interno di questo deposito c'è un registro per annotare, per ogni passeggero che abbia depositato le valigie, le informazioni riguardanti queste ultime. Dal tipo SUITCASES ho tolto il valore (-1,-1) dato che è solo un valore, come detto in precendenza, flag che non ha nessun senso al di fuori di un contesto ben determinato (non ha senso come "size" delle valigie la coppia (-1,-1) come non ha senso che dentro al deposito ci siano valigie di dimensione (-1,-1)).
Questo schema è inserito in questa categoria per il motivo che deve essere usato solo come tipo e non come schema "globale" sul quale fare delle operazioni, anche se poi, all'interno della specifica ne abbiamo usato "solo una variabile" (e quindi di depositi infatti ce n'è solamente uno) ma desideravo avere una variabile singola all'interno dello schema globale Airport.

PassengersSuitcasesDeposit

$$maxSuitcaseSize : SUITCASESIZE \setminus \{(0-1) \mapsto (0-1)\}$$
$$maxSuitcasesNumber : \mathbb{Z}$$
$$currentSuitcasesNumber : \mathbb{Z}$$
$$suitcasesRegister : PASSPORT \pfun \text{seq}(SUITCASESIZE \setminus \{(0-1) \mapsto (0-1)\})$$

Schema principale e la sua inizializzazione

schema dell'areoporto

Questo è lo schema principale di questa specifica ed il solo ed unico sul quale verranno eseguite delle operazioni. Avendo già spiegato, in precedenza, i componenti di questo schema penso che sia sufficientemente autoesplicativo

Airport

$$
\begin{array}{l}
registeredPassengers : PASSPORT \pfun PassengerInfo \\
avalaibleFlightTickets : FLIGHTS \\
suitcasesDeposit : PassengersSuitcasesDeposit
\end{array}
$$

Inializzazione

Per una questione di preferenza personale trovavo più semplice pensare all'inizializzazione come ad una operazione vera e propria, che prende come valore gli stati successivi all'operazione dei componenti dello schema (dato che quelli precedenti non ci interessano). Naturalmente al momento dell'inizializzazione dell'areoporto non abbiamo nessun passeggero registrato, i biglietti sono tutti disponibili (tranne i biglietti formati da componenti "flag", come si può ben vedere) e settiamo i valori "a piacimento" per quanto riaguarda il deposito.

AirportInit

$$
\begin{array}{l}
(Airport)' \\
\hline
registeredPassengers' = \emptyset \\
\\
avalaibleFlightTickets' = \{seat : PLANESEATID;\ dest : DESTINATION \mid \\
\quad seat \neq (0 - 1) \land dest \neq NULL \bullet seat \mapsto dest \\
\} \\
\\
suitcasesDeposit' = \langle\!| \\
\quad maxSuitcaseSize == 100 \mapsto 50, \\
\quad maxSuitcasesNumber == (\#avalaibleFlightTickets' * 3), \\
\quad currentSuitcasesNumber == 0, \\
\quad suitcasesRegister == \emptyset \\
|\!\rangle
\end{array}
$$

Comprare un biglietto aereo

l'acquisto va a buon fine

Innanzitutto l'input dovrà essere ben formato. Il passeggero può essere in possesso di un solo biglietto alla volta e se ne acquista un altro, quest'ultimo verrà sovrascritto. Nel registro dei passeggeri, per quanto riguarda le valigie del passeggero, viene aggiunta la sequenza formata dalla coppia (-1,-1) per indicare che il passeggero ha acquistato il biglietto ma non ha ancora effettuato il checkin. Per ulteriore sicurezza ho esplicitato che lo stato del deposito non cambia.

BuyAFlightTicketOk

$$
\begin{array}{l}
\Delta Airport \\
passengerId? : PASSPORT \\
passengerSeatId? : PLANESEATID \setminus \{(0-1)\} \\
passengerDestination? : DESTINATION \setminus \{NULL\} \\
operationOutput! : OPERATIONMESSAGE \\
\hline
avalaibleFlightTickets \neq \emptyset \land passengerSeatId? \mapsto passengerDestination? \in avalaibleFlightTickets \\
\\
avalaibleFlightTickets' = avalaibleFlightTickets \setminus \{passengerSeatId? \mapsto passengerDestination?\} \\
\\
(\\
\quad (\\
\qquad passengerId? \notin \mathrm{dom}\, registeredPassengers \land \\
\qquad\quad registeredPassengers' = registeredPassengers \cup \\
\qquad\quad \{ \\
\qquad\qquad passengerId? \mapsto \\
\qquad\qquad (\!| \\
\qquad\qquad\quad flightTicket == passengerSeatId? \mapsto passengerDestination?, \\
\qquad\qquad\quad suitcases == \langle (0-1) \mapsto (0-1) \rangle \\
\qquad\qquad |\!) \\
\qquad\quad \} \\
\quad) \\
\lor \\
\quad (\\
\qquad passengerId? \in \mathrm{dom}\, registeredPassengers \land \\
\qquad\quad registeredPassengers' = registeredPassengers \oplus \\
\qquad\quad \{ \\
\qquad\qquad passengerId? \mapsto \\
\qquad\qquad (\!| \\
\qquad\qquad\quad flightTicket == passengerSeatId? \mapsto passengerDestination?, \\
\qquad\qquad\quad suitcases == \langle (0-1) \mapsto (0-1) \rangle \\
\qquad\qquad |\!) \\
\qquad\quad \} \\
\quad) \\
) \\
\\
suitcasesDeposit' = suitcasesDeposit \\
operationOutput! = BuyTicketOk
\end{array}
$$

casi di fallimento dell'operazione di acquisto

flightTicketSoldOut

$\Xi Airport$
$operationOutput! : OPERATIONMESSAGE$

$avalaibleFlightTickets = \emptyset$

$operationOutput! = FlightTicketsSoldOut$

FlightTicketNotAvalaible

$\Xi Airport$
$passengerSeatId? : PLANESEATID \setminus \{(0-1)\}$
$passengerDestination? : DESTINATION \setminus \{NULL\}$
$operationOutput! : OPERATIONMESSAGE$

$passengerSeatId? \mapsto passengerDestination? \notin avalaibleFlightTickets$

$operationOutput! = WantedTicketNotAvalaible$

operazione totale

$$BuyAFlightTicket == BuyAFlightTicketOk \lor flightTicketSoldOut \lor FlightTicketNotAvalaible$$

Effettuare il Checkin

il checkin va a buon fine

Innanzitutto l'input dovrà essere ben formato. Che il passeggero si presenti con le valigie oppure senza, registreremo questo fatto all'interno del registro dei passeggeri, mentre segneremo e virtualmente metteremo le valigie nel deposito solo se queste ultime sono presenti. Incrementare il numero di valigie nel deposito risulta in ogni caso una operazione corretta. per ulteriore sicurezza ho esplicitato che lo stato dell'insieme dei biglietti disponibili non cambia.

GiveSuitcasesAtCheckinOk

$$
\begin{array}{l}
\Delta Airport \\
passengerId? : PASSPORT \\
passengerSuitcases? : \mathrm{seq}(SUITCASESIZE \setminus \{(0-1) \mapsto (0-1)\}) \\
operationOutput! : OPERATIONMESSAGE \\
\hline
passengerId? \in \mathrm{dom}\, registeredPassengers \land passengerId? \notin \mathrm{dom}\, suitcasesDeposit.suitcasesRegister \\
\\
\forall\, suitcaseSize : \mathrm{ran}\, passengerSuitcases? \bullet \\
\quad suitcaseSize.1 < suitcasesDeposit.maxSuitcaseSize.1 \land \\
\quad suitcaseSize.2 < suitcasesDeposit.maxSuitcaseSize.2 \\
\\
\#passengerSuitcases? + suitcasesDeposit.currentSuitcasesNumber \\
\quad \leq suitcasesDeposit.maxSuitcasesNumber \\
\\
registeredPassengers' = registeredPassengers \oplus \\
\{ \\
\quad passengerId? \mapsto \\
\quad (\!| \\
\quad\quad flightTicket == (registeredPassengers(passengerId?)).flightTicket, \\
\quad\quad suitcases == passengerSuitcases? \\
\quad |\!) \\
\} \\
\\
(\\
\quad (\\
\quad\quad passengerSuitcases? \neq \langle\rangle \land \\
\quad\quad suitcasesDeposit'.suitcasesRegister = suitcasesDeposit.suitcasesRegister \cup \\
\quad\quad\quad \{passengerId? \mapsto passengerSuitcases?\} \\
\quad) \\
\quad \lor \\
\quad passengerSuitcases? = \langle\rangle \\
) \\
\\
suitcasesDeposit'.currentSuitcasesNumber = \\
\quad suitcasesDeposit.currentSuitcasesNumber + \#passengerSuitcases? \\
\\
avalaibleFlightTickets' = avalaibleFlightTickets \\
operationOutput! = GiveSuitcasesOk
\end{array}
$$

casi di fallimento dell'operazione di checkin

Per l'operazione *checkinAlreadyDone* se un passeggero ha delle valigie nel deposito vuol dire che ha già fatto il checkin ed è nella situazione nella quale le sue interazioni con il deposito sono terminate (tranne con l'eventuale caso che le sue valigie siano restituite, a quel punto può rifetturare operazioni con il deposito), per cui non può rifare il checkin.

PassengerDidNotBuyTicket

$\Xi Airport$
$passengerId? : PASSPORT$
$operationOutput! : OPERATIONMESSAGE$

$passengerId? \notin \mathrm{dom}\, registeredPassengers$

$operationOutput! = MustBuyTicket$

checkinAlreadyDone

$\Xi Airport$
$passengerId? : PASSPORT$
$operationOutput! : OPERATIONMESSAGE$

$passengerId? \in \mathrm{dom}\, suitcasesDeposit.suitcasesRegister$

$operationOutput! = TooManyCheckin$

SuitcasesAreTooBig

$\Xi Airport$
$passengerSuitcases? : \mathrm{seq}(SUITCASESIZE \setminus \{(0-1) \mapsto (0-1)\})$
$operationOutput! : OPERATIONMESSAGE$

$\exists\, size : \mathrm{ran}\, passengerSuitcases? \bullet$
$size.1 > suitcasesDeposit.maxSuitcaseSize.1 \lor$
$size.2 > suitcasesDeposit.maxSuitcaseSize.2$

$operationOutput! = TooBigSuitcases$

TooManySuitcases

$\Xi Airport$
$passengerSuitcases? : \mathrm{seq}(SUITCASESIZE \setminus \{(0-1) \mapsto (0-1)\})$
$operationOutput! : OPERATIONMESSAGE$

$\#passengerSuitcases? + suitcasesDeposit.currentSuitcasesNumber$
$> suitcasesDeposit.maxSuitcasesNumber$

$operationOutput! = NoMoreSpace$

operazione totale

$$DoCheckin == GiveSuitcasesAtCheckinOk \lor PassengerDidNotBuyTicket \lor checkinAlreadyDone \lor SuitcasesAreTooBig \lor TooManySuitcases$$

Controllo di polizia dell'areporto

Ho utilizzato i primi due schemi elencati in questa sezione come predicati nell'operazione *AirportPoliceLooksForPassengersOk*. Utilizzare gli schemi come predicati equivale ad aggiungere nella parte dei predicati dello schema nel quale vengono chiamati, i loro stessi predicati; quindi in questo caso stiamo aggiungendo i predicati dei primi due schemi alla parte dei predicati del terzo ed ultimo schema.

Nello schema *airportPoliceGaveTooLittleInfo* ho pensato che dare come identikit del passeggero ricercato dei valori che indicano che il passeggero non ha fatto ancora il checkin (sequenza formata da una coppia (-1,-1) per le valigie), oppure valori uguali al valore del biglietto vuoto o addirittura dare valori non congrui non sia sufficiente per poter ricercare un passeggero.

Con questo modo di organizzare gli schemi in alcuni casi ci sarebbe un fallimento generale senza output, infatti nel caso falliscano i predicati dei due schemi iniziali questo accadrebbe.

airportPoliceGaveTooLittleInfo

$$
\begin{array}{l}
wantedPassengerDescription? : PassengerInfo \\
\hline
wantedPassengerDescription?.flightTicket.1 = (0-1) \lor \\
wantedPassengerDescription?.flightTicket.2 = NULL \lor \\
wantedPassengerDescription?.suitcases = \langle (0-1) \mapsto (0-1) \rangle \lor \\
wantedPassengerDescription?.suitcases = \langle \rangle
\end{array}
$$

AirportPoliceFoundNoOne

$$
\begin{array}{l}
\Xi Airport \\
wantedPassengerDescription? : PassengerInfo \\
\hline
wantedPassengerDescription? \notin \operatorname{ran} registeredPassengers
\end{array}
$$

il controllo della polizia ha dato risultati

AirportPoliceLooksForPassengersOk

$$
\begin{array}{l}
\Xi Airport \\
wantedPassengerDescription? : PassengerInfo \\
wantedPassengers! : \mathbb{P}\, PASSPORT \\
operationOutput! : OPERATIONMESSAGE \\
\hline
(\neg\, airportPoliceGaveTooLittleInfo) \land (\neg\, AirportPoliceFoundNoOne) \\
\\
wantedPassengers! = dom(registeredPassengers \rhd \{wantedPassengerDescription?\}) \\
\\
operationOutput! = DescribedPassengersStopped
\end{array}
$$

Cancellazione della prenotazione del volo

Ancora una volta ho utilizzato i primi 5 schemi presentati in questa sezione come predicati per rappresentare l'operazione espressa nello schema *PassengerCanceledHisFlightOk*.

Nello schema *DeletePassengerFlightTicket* vediamo usare per la prima volta la costante riguardante il "biglietto vuoto", per indicare che il passeggero non ha più un biglietto.

Nello schema *TicketIsAvalaibleAgain* ho deciso che nel caso che la distanza di tempo tra l'orario effettivo e la partenza dell'aereo sia uguale o superiore ad un'ora, allora il biglietto si può riassegnare tra i biglietti disponibili alla vendita. In questo caso ho assunto (senza specificarlo) che questa procedura non possa essere fatta direttamente dal passeggero. Dato che si parla di orari di partenza e orari "correnti" (e quindi devono essere coerenti e soprattutto giusti) non mi sembra corretto che sia il passeggero a darli, quindi il passeggero una volta espressa la sua intenzione di cancellare il biglietto, successivamente una entità più consona (pensiamo ad una procedura automatizzata) può chiamare questa operazione fornendo l'input corretto.

DeletePassengerFlightTicket

$$
\begin{array}{l}
\Delta Airport \\
passengerId? : PASSPORT \\
\hline
registeredPassengers' = registeredPassengers \oplus \\
\{ \\
\quad passengerId? \mapsto \\
\quad (\!| \\
\quad\quad flightTicket == blankTicket, \\
\quad\quad suitcases == (registeredPassengers(passengerId?)).suitcases \\
\quad |\!) \\
\}
\end{array}
$$

ThereIsEnoughTimeBeforeTheFlight

$$
\begin{array}{l}
flightDepartureTime? : TIME \\
currentTime? : TIME \\
\hline
flightDepartureTime? - currentTime? \geq 1
\end{array}
$$

PassengerHasSuitcasesInDeposit

$$
\begin{array}{l}
\Xi Airport \\
passengerId? : PASSPORT \\
\hline
passengerId? \in \mathrm{dom}\, suitcasesDeposit.suitcasesRegister
\end{array}
$$

GiveBackSuictases

$$
\begin{array}{l}
\Delta Airport \\
passengerId? : PASSPORT \\
\hline
suitcasesDeposit'.suitcasesRegister = suitcasesDeposit.suitcasesRegister \setminus \\
\quad \{passengerId? \mapsto suitcasesDeposit.suitcasesRegister(passengerId?)\}
\end{array}
$$

TicketIsAvalaibleAgain

$\Delta Airport$
$passengerId? : PASSPORT$

$avalaibleFlightTickets' = avalaibleFlightTickets \cup \{$
$\quad \{(registeredPassengers(passengerId?)).flightTicket$
$\}$

il volo viene cancellato correttamente

PassengerCanceledHisFlightOk

$\Delta Airport$
$passengerId? : PASSPORT$
$flightDepartureTime? : TIME$
$currentTime? : TIME$
$operationOutput! : OPERATIONMESSAGE$

$passengerId? \in \mathrm{dom}\, registeredPassengers$

$($
$\quad (TicketIsAvalaibleAgain \land ThereIsEnoughTimeBeforeTheFlight)$
$\quad \lor$
$\quad (\neg\, ThereIsEnoughTimeBeforeTheFlight \land avalaibleFlightTickets' = avalaibleFlightTickets)$
$)$

$DeletePassengerFlightTicket$

$($
$\quad (PassengerHasSuitcasesInDeposit \land GiveBackSuictases)$
$\quad \lor$
$\quad (\neg\, PassengerHasSuitcasesInDeposit \land suitcasesDeposit' = suitcasesDeposit)$
$)$

$operationOutput! = FlightCanceled$

operazione totale

$PassengerCanceledHisFlight == PassengerCanceledHisFlightOk \lor PassengerDidNotBuyTicket$

Invarianti di stato

Ho deciso, per questa specifica di aggiungere due invarianti che andranno a comporre l'invariante totale che dovrà essere rispettata prima e dopo ogni operazione.

L'invariante *AvalaibleFlightTicketsInv* ci assicura che i biglietti disponibili sono ben formati.

L'invariante *SuitcasesInv* ci assicura che tutti i passeggeri che risultano aver fatto il checkin e aver depositato le valigie ma non hanno cancellato il volo (e quindi non hanno ripreso le loro valigie) in base alle informazioni del registro dei passeggeri, sono tutti e soli i passeggeri che sono presenti nel deposito, dimostrando così coerenza tra i dati presenti nell'areoporto.

AvalaibleFlightTicketsInv

Airport

$$\neg\ \exists flightTicket : avalaibleFlightTickets \bullet flightTicket.1 = (0-1) \lor flightTicket.2 = NULL$$

SuitcasesInv

Airport

$$\begin{array}{l}
\mathrm{dom}\, suitcasesDeposit.suitcasesRegister = \{ID : \mathrm{dom}\, registeredPassengers \mid \\
\quad (registeredPassengers(ID)).suitcases \neq \langle\rangle \land \\
\quad (registeredPassengers(ID)).suitcases \neq \langle (0-1) \mapsto (0-1) \rangle \land \\
\quad (registeredPassengers(ID)).flightTicket \neq (0-1) \mapsto NULL \\
\}
\end{array}$$

operazione totale

$$AirportInv == AvalaibleFlightTicketsInv \land SuitcasesInv$$

Simulazione basica

*l'output è stato leggermente modificato nella forma per essere adattato a LaTeX. Il numero dei posti e le destinazioni sono state ridotte notevolmente per questa simulazione. Una volta fatta andare la simulazione sul codice si capirà l'irrisoria e non sostanziale differenza.

Codice *{log}*

```
airportInit(A)
&
airportInv(A)
&
buyAFlightTicket(A,lorenzo,roma,1,OutAB,B)
&
airportInv(B)
&
buyAFlightTicket(B,luca,milano,1,OutBC,C)
&
airportInv(C)
&
doCheckin(C,luca,{[1,[49,100]],[2,[57,91]]},OutCD,D)
&
airportInv(D)
&
doCheckin(D,lorenzo,{[1,[70,100]],[2,[57,91]]},OutDE,E)
&
airportInv(E)
&
airportPoliceLooksForPassengers(E,[[1,chicago],{[1,[55,90]]}],WantedPassengers_o,OutEF,F)
&
airportInv(F)
&
passengerCanceledHisFlight(F,lorenzo,23,21,OutFG,G)
&
airportInv(G).
```

Output

start–>airportInit
registro
{}
biglietti
{[1,milano],[1,roma],[2,milano],[2,roma]}
deposito
[[100,150],12,0,{}]
finish–>airportInit

start–>buyAFlightTicket
registro
{[lorenzo,[[1,roma],{[1,[-1,-1]]}]]}
biglietti
{[1,milano],[2,milano],[2,roma]}
deposito
[[100,150],12,0,{}]
finish–>buyAFlightTicket

start–>buyAFlightTicket
registro
{[lorenzo,[[1,roma],{[1,[-1,-1]]}]],[luca,[[1,milano],{[1,[-1,-1]]}]]}
biglietti
{[2,milano],[2,roma]}
deposito
[[100,150],12,0,{}]
finish–>buyAFlightTicket

start–>doCheckin
registro
{[lorenzo,[[1,roma],{[1,[-1,-1]]}]],[luca,[[1,milano],{[1,[-1,-1]]}]]}
biglietti
{[2,milano],[2,roma]}
deposito
[[100,150],12,0,{}]
finish–>doCheckin

start–>doCheckin
registro
{[luca,[[1,milano],{[1,[-1,-1]]}]],[lorenzo,[[1,roma],{[1,[70,100]],[2,[57,91]]}]]}
biglietti
{[2,milano],[2,roma]}
deposito
[[100,150],12,2,{[lorenzo,{[1,[70,100]],[2,[57,91]]}]}]
finish–>doCheckin

start–>airportPoliceLooksForPassengers
airportPoliceFoundNoOne–>donotconsiderfurtheroutput
wantedPassenger–>{}
registro
{[luca,[[1,milano],{[1,[-1,-1]]}]],[lorenzo,[[1,roma],{[1,[70,100]],[2,[57,91]]}]]}
biglietti
{[2,milano],[2,roma]}
deposito
[[100,150],12,2,{[lorenzo,{[1,[70,100]],[2,[57,91]]}]}]
finish–>airportPoliceLooksForPassengers

start–>passengerCanceledHisFlight
registro
{[luca,[[1,milano],{[1,[-1,-1]]}]],[lorenzo,[[-1,null],{[1,[70,100]],[2,[57,91]]}]]}
biglietti
{[2,milano],[2,roma],[1,roma]}
deposito
[[100,150],12,2,{}]
finish–>passengerCanceledHisFlight

Simulazione simbolica

*l'output è stato leggermente modificato nella forma per essere adattato a LaTeX. Il numero dei posti e le destinazioni sono state ridotte notevolmente per questa simulazione. Una volta fatta andare la simulazione sul codice si capirà l'irrisoria e non sostanziale differenza.

Codice *{log}*

```
airportInit(A)
&
airportInv(A)
&
buyAFlightTicket(A,A1,A2,A3,OutAB,B)
&
airportInv(B)
&
buyAFlightTicket(B,B1,B2,B3,OutBC,C)
&
airportInv(C)
&
doCheckin(C,C1,C2,OutCD,D)
&
airportInv(D)
&
doCheckin(D,D1,D2,OutDE,E)
&
airportInv(E)
&
airportPoliceLooksForPassengers(E,E1,WantedPassengers_o,OutEF,F)
&
airportInv(F)
&
passengerCanceledHisFlight(F,F1,F2,F3,OutFG,G)
&
airportInv(G).
```

Un possile output

_N18>=50,
_N17>=80,

subset(_N16,ris([_157752,[_157818,_157860]]in _N16,[],_157818>=50_157860>=80,
[_157752, [_157818, _157860]], *true*)),

$\_N18 =< 100$,
$\_N17 =< 150$,

$subset(\_N16, ris([\_158556, [\_158622, \_158664]]in\_N16, [], \_158622 =< 100\_158664 =< 150,$
$[\_158556, [\_158622, \_158664]], true)),$

_N15,[_N18,_N17
]nin_N16,
$size(\_N16, \_N14)$,
$\_N14 >= 0$,
$\_N13 >= 1$,
$\_N14is\_N13 - 1$,
$\_N13 >= 0$,
$\_N13 + 0 =< 12$,
$\_N12is0 + \_N13$,
$\_N11 >= 50$,
$\_N10 >= 80$,

$subset(\_N9, ris([\_160482, [\_160548, \_160590]]in\_N9, [], \_160548 >= 50\_160590 >= 80,$
$[\_160482, [\_160548, \_160590]], true)),$

$\_N11 =< 100$,
$\_N10 =< 150$,

$subset(\_N9, ris([\_161286, [\_161352, \_161394]]in\_N9, [], \_161352 =< 100\_161394 =< 150,$
$[\_161286, [\_161352, \_161394]], true)),$

_N8,[_N11,_N10
]nin_N9,
$size(\_N9, \_N7)$,
$\_N7 >= 0$,
$\_N6 >= 1$,
$\_N7is\_N6 - 1$,
$\_N6 >= 0$,
$\_N6 + \_N12 =< 12$,
$\_N5is\_N12 + \_N6$,
$set(\_N16)$,
$\_N4 - \_N3 >= 1$,
$\_N2neq\_N1$,
$set(\_N9)$