Specifica – parte IV

Leggere Sez. 5.7.2.3 Ghezzi et al.

università di ferrara

DA SEICENTO ANNI GUARDIAMO AVANTI.

Z

- Linguaggio di specifica formale basato su
  - insiemi
  - relazioni
  - funzioni
  - predicati
- Linguaggio tipizzato: ogni variabile ha un tipo.

Z

- Un sistema è specificato per mezzo del suo *spazio degli stati* (collezione di variabili tipizzate).
- Le proprietà dello spazio degli stati sono definite da predicati *invarianti* che devono essere verificati in qualunque stato.

Z

- Transizioni di stato causate da *operazioni*, definite da:
  - relazione input-output
  - predicati riguardanti lo stato
- *Schema*: costrutto che definisce gli stati e l'effetto delle operazioni sugli stati.

Tipi in Z

- Ogni variabile ha un tipo.
- Tipi predefiniti:
  - $\mathbb{N}$ (naturali), $\mathbb{N}_1$ (interi positivi), $\mathbb{Z}$ (interi)
  - $\mathbb{R}$ (reali)
  - $\mathbb{P}$ X (powerset di X)
- Il tipo di un oggetto si indica con una *dichiarazione* <variabile> : <Tipo>

  Es: x : $\mathbb{N}$

Costruzione di tipi in Z

- Enumerazione. Es:

  {2,3,5,7,11}
- *Free types*. Es:

  on|off

Costruzione di tipi in Z

- *Set comprehension*: firma|predicato• termine. Es:

  evens == {n: ℕ| n≠0 • 2*n}
- Il predicato può essere omesso se sempre vero
- Il termine può essere omesso se uguale all'unica variabile dichiarata nella firma

Quantificatori

- Esistenziale: $\exists x: \mathbb{N} \bullet x = x * x$
- Esistenziale ristretto: $\exists x:\mathbb{N} \mid x<5 \bullet x = x*x$
- Universale: $\forall x: \mathbb{N} \bullet x = x * x$
- Universale ristretto: $\forall x:\mathbb{N} \mid x<5 \bullet x = x*x$
- Semantica dei quantificatori ristretti:
  - $\forall J \mid P \bullet Q \leftrightarrow \forall J \bullet P \rightarrow Q$
  - $\exists J \mid P \bullet Q \leftrightarrow \exists J \bullet P \wedge Q$

Relazioni

- *Relazione* $r \subseteq X \times Y$,
- $X \leftrightarrow Y = \mathbb{P}(X \times Y)$ è il tipo delle relazioni fra X e Y.
- *Coppia ordinata*: (x,y) oppure $x \mapsto y$
- *Dominio* di r: $dom\ r = \{x \in X | (\exists y \in Y \bullet (x,y) \in r)\}$
- *Codominio* (o *range*) di r: $ran\ r = \{y \in Y | (\exists x \in X \bullet (x,y) \in r)\}$

Restrizione e corestrizione del dominio

- Dati U: $\mathbb{P}$ X, F: X $\leftrightarrow$ Y,
  - U $\lhd$ F = {(x,y)$\in$F | x$\in$U}
  - U $\lhd\!\!\!-$ F = {(x,y)$\in$F | x$\notin$U}
- Esempio:
  - ages = {(john,23), (mary,30), (tom,27), (alan,27), (alice,31)}
  - male = {john, tom, alan}
  - male $\lhd$ ages = {(john,23),(tom,27),(alan,27)}
  - male $\lhd\!\!\!-$ ages = {(mary,30), (alice,31)}

Funzioni

- Una *funzione* f da X a Y è una relazione che a ogni elemento di X associa al massimo un elemento di Y.
- Funzione *parziale* $X \nrightarrow Y$: non definita su tutti gli elementi di X.
- Funzione totale $X \rightarrow Y$: definita su tutti gli elementi di X
- *Overriding* $\oplus$: $f \oplus g$ è la funzione che si ottiene sostituendo ogni coppia di g a quella di f avente lo stesso primo elemento. Es: bisestile = normale $\oplus$ {(feb, 29)}

Sequenze

- Tipo seq X: liste di elementi di X (indicati fra parentesi acute). Es. <gen, feb, mar>
- Se m è la lunghezza, si possono rappresentare come funzioni da 1..m a X.
- Operazioni:
  - *Concatenazione*: <gen, feb> ^ <mar> = <gen, feb, mar>
  - *Head*: head <gen, feb, mar> = gen
  - *Tail*: tail <gen, feb, mar> = <feb, mar>

Bag (o multiset o multiinsiemi)

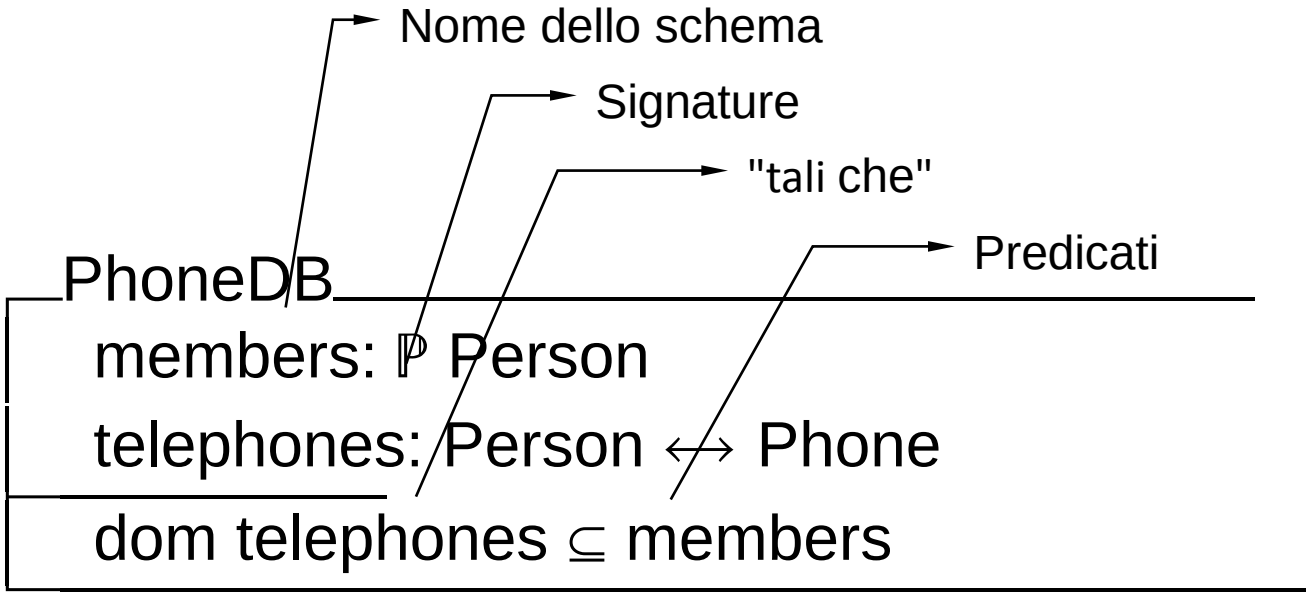
- Differiscono dagli insiemi perchè un elemento può comparire più volte (l'ordine non conta)
- L: bag Person, L = ⟦john,john,fred,tom,tom,tom⟧ = ⟦fred,tom,john,tom,tom,john⟧
- Si può vedere come funzione parziale
  bag Person = Person $\rightarrow\!\!\!\!\!+$ N_1
- Operazioni:
  - count: count L john = 2;
  - unione $\uplus$: {john $\mapsto$ 1} $\uplus$ {john $\mapsto$ 2, tom $\mapsto$ 1} = {john $\mapsto$ 3, tom $\mapsto$ 1}

Schema

- Rappresentazione grafica bidimensionale per rappresentare stati
- Formato da
  - insieme di oggetti
  - predicati veri per gli oggetti
- Usi:
  - specificare uno stato
  - specificare transizioni di stato

Schema: esempio

- Si vogliono scrivere le specifiche in Z di un database di numeri telefonici interni, ad esempio dei numeri interni di una università



Descrizioni assiomatiche

- Introducono variabili, funzioni e relazioni valide in tutta la specifica. Esempio:

la funzione sum calcola il valore totale di un bag di numeri:
sum {2 ⊎ 7, 5 ⊎ 3} = 14 + 15 = 29

Oggetti

$$\text{sum}: \text{bag}\ \mathbb{N} \rightarrow \mathbb{N}$$

$$\forall\ i, j: \mathbb{N}; L: \text{bag}\ \mathbb{N} \bullet$$
$$\text{sum}\ [\![\ ]\!] = 0 \land$$
$$\text{sum}(\{i \mapsto j\} \cup L) = i * j + \text{sum}\ L$$

Predicati che vincolano gli oggetti

Definizioni locali

- La parola chiave let permette di definire variabili locali alla parte dei predicati. Es:
- (let x==i+j+k • x+j>x-j) equivale a i+j+k+j>i+j+k-j
- In generale, (let $x_1==t_1,...,x_n==t_n$ • P) equivale a $P[t_1/x_1,...t_n/x_n]$ (sostituzione)
- Nessun x_i può comparire in nessun t_j, per i,j=1..n

Operazioni su schemi

- Inclusione: uno schema S può essere incluso nella parte di dichiarazione di un altro schema T.
  - Le dichiarazioni di S sono aggiunte a quelle di T
  - I predicati di S sono in congiunzione con quelli di T.

Operazioni su schemi

- Se S e T sono schemi e ♦ è un generico operatore logico, S ♦ T è lo schema che si ottiene
  - Unendo le dichiarazioni di S e T
  - Combinando i loro predicati per mezzo di ♦.

Convenzioni per operazioni

- Negli schemi che definiscono un'operazione:
  - <nome>? indica un parametro di ingresso dell'operazione
  - <nome>! indica un parametro di uscita
  - <nome>' indica il valore dell'oggetto <nome> dopo l'operazione
- I predicati sui parametri di uscita e valori con ' sono *postcondizioni*
- Gli altri sono *precondizioni*

Convenzioni per operazioni

- Se S è uno schema, S’ è ottenuto da S aggiungendo un apice a tutte le variabili, sia nella firma che nei predicati

Convenzioni per operazioni

- In schemi che specificano operazioni
- Convenzione Δ:

  $\Delta S = S \wedge S'$
- Convenzione Ξ (xi):

  ΞS (oppure $\theta S = \theta S'$) indica che l'operazione non cambia lo stato di S

Esempio

ΔPhoneDB

$members, members' : \mathbb{P}\ Person$

$telephones, telephones' : Person \leftrightarrow Phone$

$dom\ telephones \subseteq members$

$dom\ telephones' \subseteq members'$

ΞPhoneDB

ΔPhoneDB

$members = members'$

$telephones = telephones'$

Database di numeri telefonici interni

- Person, il tipo delle persone
- Phone, il tipo di tutti i numeri telefonici interni
- telephones: Person ↔ Phone, relazione fra persone e numeri telefonici
- telephones = {(diller, 4794), (jarrat,4936), (jarrat,5317), (smith, 3174), (jones,3174)}

Schema dello stato

PhoneDB

$$members : \mathbb{P}\ Person$$

$$telephones : Person \leftrightarrow Phone$$

$$dom\ telephones \subseteq members$$

Aggiunta di una entry

AddEntry

$members, members' : \mathbb{P}\ Person$

$telephones, telephones' : Person \leftrightarrow Phone$

$name? : Person$

$newnumber? : Phone$

$dom\ telephones \subseteq members$

$dom\ telephones' \subseteq members'$

$name? \in members$

$name? \mapsto newnumber? \notin telephones$

$telephones' = telephones \cup \{name? \mapsto newnumber?\}$

$members' = members$

Schema più conciso

AddEntry

ΔPhoneDB
name?: Person
newnumber?: Phone

---

name? $\in$ members
name? $\mapsto$ newnumber? $\notin$ telephones
telephones' = telephones $\cup$ {name? $\mapsto$ newnumber?}
members' = members

Condizioni di errore

NotMember

ΞPhoneDB
name?: Person
rep!: Report

---

name? $\notin$ members
rep! = 'Not a member'

EntryAlreadyExist

ΞPhoneDB
name?: Person
newnumber?: Phone
rep!: Report

---

name? $\mapsto$ newnumber? $\in$ telephones
rep! = 'Entry already exists'

Specifica completa

Success

rep!: Report

---

rep! = 'Okay'

$$DoAddEntry \mathrel{\widehat=} AddEntry \land Success$$

$$\lor$$

$$NotMember$$

$$\lor$$

$$EntryAlreadyExists$$

Interrogare per persona

FindPhones

ΞPhoneDB
name?: Person
numbers!: $\mathbb{P}$ Phones

---

name? $\in$ dom telephones
numbers! = telephones $\lhd$ {name?} $\rhd$

Immagine di un insieme I attraverso una relazione R:X$\leftrightarrow$Y:
{j$\in$Y | ($\exists$i$\in$I • (i,j) $\in$ R)}

NoNumber

ΞPhoneDB
name?: Person
rep!: Report

---

name? $\notin$ dom telephones
rep! = 'No number for the person'

Specifica completa:

$$\text{DoFindPhones} \mathrel{\widehat=} \text{FindPhones} \wedge \text{Success} \vee \text{NoNumber}$$

Interrogare per numero

FindNames

ΞPhoneDB
names!: $\mathbb{P}$ Person
number?: Phones

---

number? $\in$ ran telephones
names! = telephones^{-1} ⦇{number?}⦈

UnknownNumber

ΞPhoneDB
number?: Phone
rep!: Report

---

number? $\notin$ ran telephones
rep! = 'Uknown number'

Specifica completa:

$$\text{DoFindNames} \mathrel{\widehat=} \text{FindNames} \land \text{Success} \lor \text{UnknownNumber}$$

Rimuovere una entry

RemoveEntry

ΔPhoneDB
name?: Person
oldnumber?: Phones

---

name? $\mapsto$ oldnumber? $\in$ telephones
telephones' = telephones $\setminus$ {name? $\mapsto$ oldnumber?}
members'=members

UnknownEntry

ΞPhoneDB
oldnumber?: Phone
name?: Person
rep!: Report

---

name? $\mapsto$ oldnumber? $\notin$ telephones
rep! = 'Uknown entry'

$$\text{DoRemoveEntry} \mathrel{\widehat{=}} \text{RemoveEntry} \wedge \text{Success} \vee \text{UnknownEntry}$$

Aggiungere un membro

AddMember

ΔPhoneDB
name?: Person

---

name? $\notin$ members
members'=members $\cup$ {name?}
telephones' = telephones

AlreadyMember

ΞPhoneDB
name?: Person
rep!: Report

---

name? $\in$ members
rep! = 'Already a member'

$$\text{DoAddMember} \mathrel{\widehat=} \text{AddMember} \wedge \text{Success} \vee \text{AlreadyMember}$$

Rimuovere un membro

RemoveMember

ΔPhoneDB
name?: Person

---

name? $\in$ members
members' = members $\setminus$ {name?}
telephones' = {name?} $\lhd$ telephones

DoRemoveMember $\widehat{=}$ RemoveMember $\wedge$ Success

$\vee$

NotMember

Schema renaming e schema hiding

- Schema renaming:

  S[m/x] è lo schema che si ottiene da S sostituendo m a tutte le occorrenze libere (non quantificate) di x.

- Schema hiding:

  S\(x) è lo schema che si ottiene da S, in cui la variabile x è rimossa dalla parte di dichiarazione e quantificata esistenzialmente nella parte dei predicati

Schema renaming e schema hiding

Renaming

S

$x: \mathbb{N}$

$y: \mathbb{P}\ \mathbb{N}$

$x \in y$

$T \mathrel{\widehat=} S[m/x]$

T

$m: \mathbb{N}$

$y: \mathbb{P}\ \mathbb{N}$

$m \in y$

Hiding

$T \mathrel{\widehat=} S \setminus (x)$

T

$y: \mathbb{P}\ \mathbb{N}$

$\exists x: \mathbb{N} \bullet x \in y$

Composizione di due schemi

- Serve a costruire operazioni complesse da operazioni semplici.
- S;T: Consiste nel legare le variabili dello stato successivo di S con le variabili dello stato iniziale di T.
- S;T specifica l'esecuzione successiva delle operazioni specificate da S e T.

Composizione di schemi: procedimento

S

$x?, s, s', y! : \mathbb{N}$

$s' = s - x?$

$y! = s$

T

$x?, s, s' : \mathbb{N}$

$s < x?$

$s' = s$

1. Dare alle variabili di stato futuro di S un nuovo nome ($S[s^+/s']$)
2. Dare alle variabili di stato iniziale di T lo stesso nome ($T[s^+/s]$)
3. Congiunzione dei due schemi ottenuti ($S[s^+/s'] \wedge T[s^+/s]$)
4. Nascondere le variabili introdotte
5. Semplificare

$S \fatsemi T \mathrel{\widehat=} (S[s^+/s'] \wedge T[s^+/s]) \backslash (s^+)$

Composizione di schemi: esempio

Passo 1: renaming

$S[s^+/s']$

$x?, s, s^+, y! : \mathbb{N}$

$s^+ = s - x?$

$y! = s$

Passo 2: renaming

$T[s^+/s]$

$x?, s^+, s' : \mathbb{N}$

$s^+ < x?$

$s' = s^+$

Passo 3: and

$S[s^+/s'] \land T[s^+/s]$

$x?, s, s', s^+, y! : \mathbb{N}$

$s^+ = s - x?$

$y! = s$

$s^+ < x?$

$s' = s^+$

Passo 4: hiding

$S[s^+/s'] \land T[s^+/s] \setminus s^+$

$x?, s, s', y! : \mathbb{N}$

$\exists\ s^+ : \mathbb{N} \bullet$

$(s^+ = s - x?$

$y! = s$

$s^+ < x?$

$s' = s^+)$

Passo 5: semplificazione

$S \fatsemi T$

$x?, s, s', y! : \mathbb{N}$

$s' = s - x?$

$y! = s$

$s' < x?$

Modifica di un numero di telefono

RemoveEntry ⨟ AddEntry

ΔPhoneDB

oldnumber?,newnumber?: Phones

name?: Person

---

name? $\in$ members

name? $\mapsto$ newnumber? $\notin$ telephones

$\setminus$ {name? $\mapsto$ oldnumber?}

name? $\mapsto$ oldnumber? $\in$ telephones

telephones' = telephones

$\setminus$ {name? $\mapsto$ oldnumber?} $\cup$ {name? $\mapsto$ newnumber?}

members'=members

Modifica di un numero di telefono

$$UpdateEntry \mathrel{\widehat{=}} RemoveEntry \fatsemi AddEntry \land Success \lor UnknownEntry \lor NotMember$$

Specifica di un distributore automatico

- Vogliamo specificare un distributore automatico di merendine: l'utente inserisce le monete e riceve l'articolo richiesto più il resto.
- Sia Good l'insieme di tutti gli articoli che possono essere acquistati dalla macchina. Ad ogni istante, solo un sottoinsieme di Good sarà disponibile dalla macchina.

Specifica di un distributore automatico

- Lo stato della macchina è descritto dallo schema VendingMachine

VendingMachine

$coin: \mathbb{P}\ \mathbb{N}_1$
$cost: Good \rightarrow \mathbb{N}$
$stock: bag\ Good$
$float: bag\ \mathbb{N}_1$

$dom\ stock \subseteq dom\ cost$
$dom\ float \subseteq coin$

Specifica di un distributore automatico

- coin: insieme di tutte le monete che la macchina accetta.
- La funzione cost restituisce per ciascun articolo il suo costo in centesimi. Ad es

cost(wispa) = 17

cost(crisps) = 19

cost(kitkat) = 19

Specifica di un distributore automatico

- stock: articoli correntemente nella macchina. Ad es

stock = {wispa ↦ 5, crisps ↦ 1, kitkat ↦ 3}

- float: monete correntemente nella macchina. Ad es

float = {100 ↦ 5, 20 ↦ 3, 10 ↦ 1, 2 ↦ 45, 1 ↦ 13}

Specifica di un distributore automatico

$$\Delta VendingMachine \mathrel{\widehat{=}} VendingMachine \land VendingMachine'$$

$$\Xi VendingMachine \mathrel{\widehat{=}} \Delta VendingMachine \mid$$
$$coin' = coin \land$$
$$cost' = cost \land$$
$$stock' = stock \land$$
$$float' = float$$

Specifica di un distributore automatico

Success

$rep! : Report$

$rep! = \text{'Okay'}$

InitVendingMachine

$VendingMachine'$

$coin' = \{\}$

$cost' = \{\}$

$stock' = [\![\,]\!]$

$float' = [\![\,]\!]$

Assegnare un prezzo ad un articolo

Price

ΔVendingMachine
item?: Good
price?: $\mathbb{N}_1$

---

cost' = cost $\oplus$ {item? $\mapsto$ price?}
coin' = coin
stock' = stock
float' = float

L'operazione ha sempre successo, quindi

DoPrice $\widehat{=}$ Price $\wedge$ Success

Aggiungere una moneta

Aggiungere una moneta all'insieme delle monete accettabili.

Accept

ΔVendingMachine

$c? : \mathbb{N}_1$

$c? \notin coin$

$coin' = coin \cup \{c?\}$

$cost' = cost$

$stock' = stock$

$float' = float$

Caso in cui la moneta sia già accettabile

AlreadyAcceptable

ΞVendingMachine
c?: $\mathbb{N}_1$
rep!: Report

---

c? $\in$ coin
rep! = 'Coin already acceptable'

DoAccept $\widehat{=}$ Accept $\wedge$ Success
$\vee$
AlreadyAcceptable

Rifornimento di articoli

ReStock

ΔVendingMachine

new?: bag Good

---

dom new? $\subseteq$ dom cost

stock' = stock $\uplus$ new?

coin' = coin

cost' = cost

float' = float

Rifornimento di articoli

ReStock fallisce se si tenta di aggiungere degli articoli che non hanno prezzo

GoodsNotPriced

ΞVendingMachine
new?: bag Good
rep!: Report

$\neg$(dom new? $\subseteq$ dom cost)
rep! = 'Some goods not priced'

$$\text{DoReStock} \mathrel{\widehat{=}} \text{ReStock} \wedge \text{Success} \vee \text{GoodsNotPriced}$$

Rifornimento di articoli

Definiamo la funzione sum che calcola il valore totale di un bag di numeri

$$\text{sum } \{2 \mapsto 7, 5 \mapsto 3\} = 14 + 15 = 29$$

$$\text{sum}: \text{bag } \mathbb{N} \rightarrow \mathbb{N}$$

$$\forall\ i, j: \mathbb{N}; L: \text{bag } \mathbb{N} \bullet$$
$$\text{sum } [\![\ ]\!] = 0 \land$$
$$\text{sum}(\{i \mapsto j\} \cup L) = i * j + \text{sum } L$$

Questa notazione è chiamata descrizione assiomatica

Definizioni generiche

- Definiscono una famiglia di variabili, funzioni o relazioni in base ad uno o più tipi generici X che devono poi essere istanziati a tipi specifici quando gli identificatori sono usati.
- Definiamo la relazione "sub-bag" ($\subseteq_{bag}$) analogo della relazione sottoinsieme per i bag

$$\{wispa \mapsto 2, crisps \mapsto 1\} \subseteq_{bag} \{wispa \mapsto 5, crisps \mapsto 1, kitkat \mapsto 3\}$$

Definizioni generiche

$$[X]$$
$$\_ \subseteq_{bag} \_ : \text{bag } X \leftrightarrow \text{bag } X$$
$$\forall\ L, M: \text{bag } X \bullet$$
$$L \subseteq_{bag} M \Leftrightarrow (\forall\ x: X \bullet \text{count } L\ x \leqslant \text{count } M\ x)$$

Il box usato in questa definizione

$$[X]$$
$$x: X$$
$$P$$

introduce una definizione generica in Z. Definisce una intera famiglia di variabili x di tipo generico X che soddisfano un predicato P. Quando usiamo x dobbiamo specificate il tipo effettivo al posto di X scrivendo ad es:

x[ℕ] oppure x[Europe]

Quando X si può desumere dal contesto è omesso.

Acquistare

- Operazione di acquisto di un articolo dalla macchina
- in? monete inserite nella macchina
- item? articolo che si vuole acquistare
- out! resto

Buy

ΔVendingMachine
in?, out!: bag $\mathbb{N}_1$
item?: Good

---

item? $\in$ dom stock
sum(in?) $\geq$ cost(item?)
dom in? $\subseteq$ coin
out! $\subseteq_{bag}$ float
sum(in?) = sum(out!) + cost(item?)
stock' $\uplus$ {item? $\mapsto$1}= stock
float' $\uplus$ out!= float $\uplus$ in?
coin' = coin
cost' = cost

Condizioni di errore sull'acquisto

NotInStock

ΞVendingMachine
item?: Good
rep!: Report

---

item? $\notin$ dom stock
rep! = 'Item not in stock'

TooLittleMoney

ΞVendingMachine
in?: bag $\mathbb{N}$
item?: Good
rep!: Report

---

sum(in?) $<$ cost(item?)
rep! = 'Insert more money'

Condizioni di errore sull'acquisto

ExactChangeUnavailable

ΞVendingMachine
in?: bag $\mathbb{N}$
item?: Good
rep!: Report

$\neg \exists L$: bag $\mathbb{N}$ $\bullet$
$(L \subseteq_{bag} \text{float} \wedge$
sum(in?) = sum(L) + cost(item?))
rep! = 'Correct change unavailable'

ForeignCoin

ΞVendingMachine
in?: bag $\mathbb{N}$
rep!: Report

$\neg$(dom in? $\subseteq$ coin)
rep! = 'Unacceptable coin'

Acquisto

$$DoBuy \mathrel{\widehat{=}} Buy \wedge Success$$
$$\vee$$
$$NotInStock$$
$$\vee$$
$$TooLittleMoney$$
$$\vee$$
$$ExactChangeUnavailable$$
$$\vee$$
$$ForeignCoin$$

Ritirare il guadagno

RemoveMoney

ΔVendingMachine

profit?: bag $\mathbb{N}$

---

float' $\uplus$ profit? = float

stock' = stock

coin' = coin

cost' = cost

Ritirare il guadagno - errori

Si ha errore se si tenta di togliere troppe monete

Profiteering

ΞVendingMachine
profit?: bag $\mathbb{N}$
rep!: Report

$\neg \exists L$: bag $\mathbb{N}$ $\bullet$
$(L \uplus \text{profit?} = \text{float})$
rep! = 'Such profit non existent'

Ritirare il guadagno

$$\text{DoRemoveMoney} \mathrel{\widehat{=}} \text{RemoveMoney} \land \text{Success}$$
$$\lor$$
$$\text{Profiteering}$$

Esempio: ascensore

SWITCH ::= on | off
MOVE::=up | down

FLOORS: $\mathbb{N}$

FLOORS>0

IntButtons

IntReq : 1..FLOORS → SWITCH

FloorsButtons

ExtReq : 1..FLOORS → $\mathbb{P}$ MOVE

---

down $\notin$ ExtReq(1)
up $\notin$ ExtReq(FLOORS)

Scheduler

NextFloorToServe : 0..FLOORS

Elevator

CurFloor : 1..FLOORS
CurDirection: MOVE

Spazio degli stati – versione 1

System

Elevator
IntButtons
FloorButtons
Scheduler

NextFloorToServe $\neq$ 0
$\Rightarrow$ IntReq(NextFloorToServe)=on $\vee$ ExtReq(NextFloorToServe) $\neq \emptyset$

Permette NextFloorToServe = 0 anche in presenza di richieste

Spazio degli stati – versione 2

System

Elevator
IntButtons
FloorButtons
Scheduler

NextFloorToServe $\neq$ 0 $\Rightarrow$
IntReq(NextFloorToServe)=on $\vee$ ExtReq(NextFloorToServe) $\neq \emptyset$
NextFloorToServe=0 $\Rightarrow$
$(\forall$ f : 1..FLOORS • (IntReq(f)=off $\wedge$ ExtReq(f)= $\emptyset$))

Non specifica nessuna politica di scheduling.
Non garantisce che tutte le richieste siano servite.

Spazio degli stati – versione 3

System

Elevator
IntButtons
FloorButtons
Scheduler

$\exists$ Pri1, Pri2, Pri3 : $\mathbb{P}$ $\mathbb{N}$ •
CurDirection = up $\Rightarrow$
(Pri1={f:1..FLOORS | f$\geqslant$CurFloor$\wedge$ (IntReq(f)=on $\vee$ up $\in$ ExtReq(f))} $\wedge$
Pri2={f:1..FLOORS | down $\in$ ExtReq(f) $\vee$ (f<CurFloor $\wedge$ IntReq(f)=on)} $\wedge$
Pri3={f:1..FLOORS | f<CurFloor $\wedge$ up $\in$ ExtReq(f)} $\wedge$
((Pri1 $\neq$ Ø $\wedge$ NextFloorToServe=min(Pri1)) $\vee$
(Pri1 = Ø $\wedge$ Pri2 $\neq$ Ø $\wedge$ NextFloorToServe=max(Pri2)) $\vee$
(Pri1 = Ø $\wedge$ Pri2 = Ø $\wedge$ Pri3 $\neq$ Ø $\wedge$ NextFloorToServe=min(Pri3))
$\vee$ (Pri1 = Ø $\wedge$ Pri2 = Ø $\wedge$ Pri3 = Ø $\wedge$ NextFloorToServe=0))) $\wedge$
CurDirection = down$\Rightarrow$
(Pri1={f:1..FLOORS | f$\leqslant$CurFloor $\wedge$
(IntReq(f) = on $\vee$ down $\in$ ExtReq(f))} $\wedge$
Pri2={f:1..FLOORS | up $\in$ ExtReq(f) $\vee$
f > CurFloor $\wedge$ IntReq(f)=on)} $\wedge$
Pri3 ={f:1..FLOORS | f > CurFloor $\wedge$ down $\in$ ExtReq(f)} $\wedge$
((Pri1 $\neq$ Ø $\wedge$ NextFloorToServe=max(Pri1)) $\vee$
(Pri1 = Ø $\wedge$ Pri2 $\neq$ Ø $\wedge$ NextFloorToServe=min(Pri2)) $\vee$
(Pri1 = Ø $\wedge$ Pri2 = Ø $\wedge$ Pri3 $\neq$ Ø $\wedge$ NextFloorToServe=max(Pri3))
$\vee$ (Pri1 = Ø $\wedge$ Pri2 = Ø $\wedge$ Pri3 = Ø $\wedge$ NextFloorToServe=0)))

Operazioni su ascensore

MoveToNextFloor

ΔSystem

NextFloorToServe $\neq 0$
CurFloor $\neq$ NextFloorToServe
CurFloor > NextFloorToServe $\Rightarrow$
CurFloor' = CurFloor-1 $\wedge$ CurDirection'=down
CurFloor < NextFloorToServe $\Rightarrow$
CurFloor' = CurFloor+1 $\wedge$ CurDirection'=up
ϑIntButtons'= ϑIntButtons
ϑFloorButtons'= ϑFloorButtons

InternalPush

ΔSystem
f? : 1..FLOORS

IntReq' = IntReq $\oplus$ {(f? $\mapsto$on)}
ϑElevator'= ϑElevator
ϑFloorButtons'= ϑFloorButtons

Operazioni su ascensore

ExternalPush

ΔSystem
f? : 1..FLOORS
dir? : MOVE

ExtReq' = ExtReq $\oplus$ {(f? $\mapsto$(ExtReq(f?) $\cup$ {dir?}))}
ϑElevator'= ϑElevator
ϑIntButtons'= ϑIntButtons

ServeIntRequest

ΔSystem

NextFloorToServe = CurFloor
IntReq(CurFloor) = on
IntReq' = IntReq $\oplus$ {(CurFloor $\mapsto$off)}
ExtReq' = ExtReq
CurFloor'= CurFloor
CurDirection'= CurDirection

Operazioni su ascensore

ServeExtRequestSameDir

ΔSystem

NextFloorToServe = CurFloor
IntReq(CurFloor) = off
CurDirection $\in$ ExtReq(CurFloor)
IntReq' = IntReq
ExtReq' = ExtReq $\oplus$ {(CurFloor $\mapsto$(ExtReq(CurFloor) \ {CurDirection}))}
CurFloor'= CurFloor
CurDirection'= CurDirection

Operazioni su ascensore

ServeExtRequestOtherDir

ΔSystem

NextFloorToServe = CurFloor
IntReq(CurFloor) = off
CurDirection $\notin$ ExtReq(CurFloor)
IntReq' = IntReq
ExtReq' = ExtReq $\oplus$ {(CurFloor $\mapsto \emptyset$)}
CurFloor'= CurFloor
CurDirection'= CurDirection

SystemInit

System'

$\forall$ i : 1..FLOORS • IntReq'(i)= off $\wedge$ ExtReq'(i)= $\emptyset$
NextFloorToServe'= 0
CurFloor'= 1
CurDirection'= up