INGEGNERIA DEL SOFTWARE

ZETA

Avvertenza: gli appunti si basano sul corso di Ingegneria del Software tenuto dal prof. Picco della facoltà di Ingegneria del Politecnico di Milano (che ringrazio per aver acconsentito alla pubblicazione). Essendo stati integrati da me con appunti presi a lezione, il suddetto docente non ha alcuna responsabilità su eventuali errori, che vi sarei grato mi segnalaste in modo da poterli correggere.

e-mail: webmaster@morpheusweb.it
web: http://www.morpheusweb.it

ZETA

Z si ispira al lavoro di Zermelo e di Fraenkel sulla teoria degli insiemi e sulla logica del primo ordine
E' stato sviluppato negli anni '70 da Programming Research Group (PRG) presso Oxford University Computing Laboratory (OUCL)
Z è ora uno standard ISO di pubblico dominio
E' stato utilizzato in ambiente industriale e ne esistono diverse estensioni Object Z, real-time Z, …

Zeta è un linguaggio basato su metodi formali.

- Elementi matematici
- Specifica non ambigua (Le definizioni sono chiare e precise, non si lavora su tutta l'applicazione ma solo su una parte).
- Astrazione del modello

E' composto da:

- Logica del primo ordine
- Concetto di tipo
- Schema (definisco le relazioni tramite logica del primo ordine)
- Linguaggio naturale

Z **specifica** le proprietà di un sistema (descrizione formale)

Z **prova** rigorosamente le proprietà del sistema (non verrà trattato in questo corso)

Z non è adatto per: proprietà non funzionali usability, performance, reliability real-time e concorrenza (si userà ad esempio Trio)

OPERATORI

$\neg$	not
$\wedge$	and
$\vee$	or
$\Rightarrow$	implica
$\Leftrightarrow$	sse

TABELLE DELLA VERITA'

p	q	$p \wedge q$
t	t	t
t	f	f
f	t	f
f	f	f

p	q	$p \vee q$
t	t	t
t	f	t
f	t	t
f	f	f

p	q	$p \Rightarrow q$
t	t	t
t	f	f
f	t	t
f	f	t

p	q	$p \Leftrightarrow q$
t	t	t
t	f	f
f	t	f
f	f	t

RELAZIONI

- Relazione $\leftrightarrow$

$$X \leftrightarrow Y \subseteq \mathbb{P}(X \times Y)$$

- $r1 \cup r2$

$r1 \cup r2 == \{x:X, y:Y \bullet x \mapsto y \in r1 \Rightarrow x \mapsto y \in r1 \cup r2 \wedge x \mapsto y \in r2 \Rightarrow x \mapsto y \in r1 \cup r2\}$

- $A \lhd R$ il dominio di R viene ristretto ad A

$$A \lhd R = \{x:X; y:Y \mid x \mapsto y \in R \wedge x \in A \bullet x \mapsto y\}$$

- $A \lhd\!\!\!- R \mathrel{\widehat{=}} (X \setminus A) \lhd R$
- $R \rhd B$ il range di R viene ristretto a B

$$R \rhd B = \{x:X; y:Y \mid x \mapsto y \in R \wedge y \in B \bullet x \mapsto y\}$$

- $R \rhd\!\!\!- B \mathrel{\widehat{=}} R \rhd (Y \setminus B)$

FUNZIONI

- $\nrightarrow$ Funzione Parziale

$$X \nrightarrow Y == \{f:X \leftrightarrow Y \mid \forall x:X;y1,y2:Y \bullet x \mapsto y1 \in f \land x \mapsto y2 \in f \Rightarrow y1=y2\}$$

- $\rightarrow$ Funzione Totale

$$X \rightarrow Y == \{f:X \nrightarrow Y \mid dom\ f = X\}$$

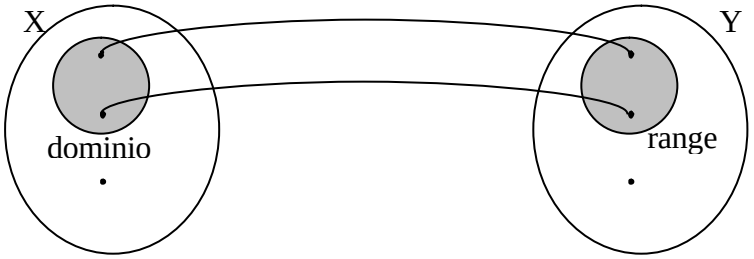
- $\rightarrowtail$ Funzione Parziale ed Iniettiva

$$X \rightarrowtail Y == \{f:X \nrightarrow Y \mid \forall x1,x2:X;\ \forall y1,y2:Y \bullet x1 \neq x2 \land x1 \mapsto y1 \land x2 \mapsto y2 \Rightarrow y1 \neq y2\}$$

- $\twoheadrightarrow$ Funzione Parzile Surriettiva

$$X \twoheadrightarrow Y == \{f:X \nrightarrow Y \mid \forall y:Y \bullet \exists x:X \bullet x \mapsto y\}$$

- $\rightarrowtail$ Funzione Totale Biettiva (= surriettiva + iniettiva)

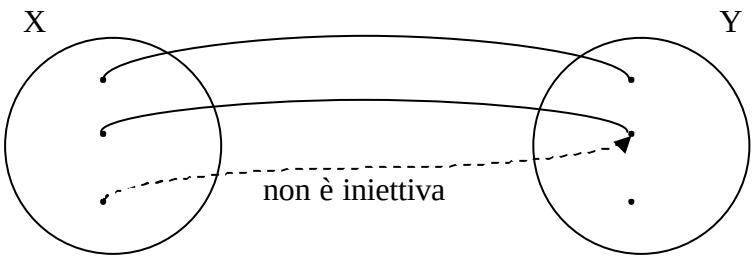


E' una funzione quando da un elemento di X parte un solo ramo verso Y ($x \mapsto y_1 \Leftrightarrow f(x)=y_1$)

$\nrightarrow$ Funzione **parziale**: non è richiesto che sia definita su tutto X.

$\rightarrow$ Funzione **totale**: definita su tutto X.

$\rightarrowtail$ Funzione **iniettiva**: se prendiamo due X diverse e calcoliamo $f(x_1)$ ed $f(x_2)$, otteniamo y_1 e y_2 diverse.



$\twoheadrightarrow$ Funzione **suriettiva**: se è definita su tutto il range Y (è il condominio).

$\rightarrowtail$ Funzione **biettiva**: se è iniettiva e suriettiva.

COME SCRIVERE UNA SPECIFICA

Z è un formalismo basato sugli insiemi Per scrivere una specifica devo:

1. Individuare gli insiemi necessari per descrivere il sistema
2. Individuare i sottoinsiemi (e le relative relazioni/funzioni) attraverso cui descriverò le proprietà del sistema
3. Descrivere le proprietà del sistema descrivendo vincoli sui sottoinsiemi e sulle relazioni/funzioni (invarianti)
4. Descrivere come il sistema può evolvere

ESEMPIO: PRENOTAZIONI

Voglio descrivere l'ufficio di prenotazione di un teatro (per semplicità consideriamo che sia un solo spettacolo)

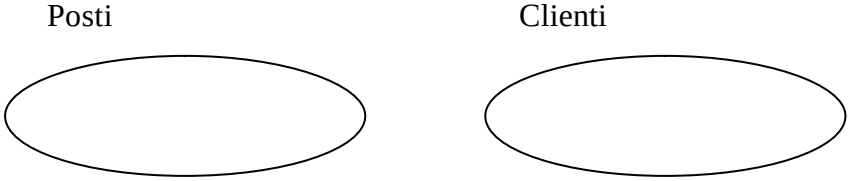
Voglio descrivere:

- Posti del teatro
- Quali posti è possibile prenotare
- Clienti del teatro
- Prenotazioni effettuate

1° PASSO: INSIEMI

Quali sono gli insiemi?

- Posti insieme dei posti del teatro (prenotabili e non prenotabili)
- Clienti insieme di tutti i possibili clienti del teatro (non solo quelli che hanno prenotato)



2° PASSO: SOTTOINSIEMI

I posti prenotabili sono un sottoinsieme dei posti del teatro

La specifica dovrà descrivere delle proprietà relativamente ai posti prenotabili

Non ha senso descrivere i posti prenotabili come un insieme, perché questi possono variare e non sono fissi. Li descriveremo come sottoinsieme dei posti.

- i posti del teatro sono sempre gli stessi
- i posti prenotabili possono variare

PostiPrenotabili⊆*Posti*

2° PASSO: RELAZIONI\FUNZIONI

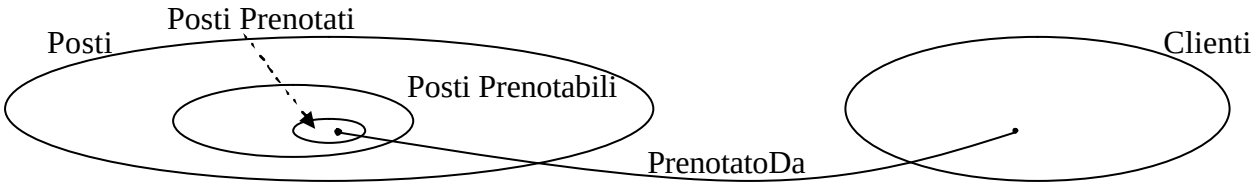
Voglio descrivere le prenotazioni dei posti del teatro
Definisco la funzione ***PrenotatoDa*** che collega un posto al cliente che l'ha prenotato.

E' una funzione perché un posto può essere prenotato da al più un cliente
Non è totale poiché non è vero che per qualsiasi posto c'è un cliente che l'ha prenotato
Non è iniettiva poiché uno stesso cliente può prenotare più posti
Non è suriettiva poiché non è vero che tutti i clienti devono aver prenotato almeno un posto.

PrenotatoDa: Posti ⇸ Clienti

3° PASSO: PROPRIETA' DEL SISTEMA

I posti che i clienti possono prenotare sono solo e soltanto quelli che appartengono all'insieme dei posti prenotabili



dom *PrenotatoDa* ⊆ *PostiPrenotabili*

SPECIFICA IN Z

La specifica in Z è suddivisa in due parti:

- Definizione degli insiemi (in Z sono chiamati **tipi**)
- Definizione delle relazioni/funzioni e delle proprietà attraverso gli **schemi**

DEFINIZIONE DI TIPO

E' un insieme generico, dove non specifichiamo gli elementi che lo compongono

[nome_tipo]

per esempio
[Posti]

E' un insieme definito attraverso l'enumerazione dei suoi elementi

nome_tipo::=a1|a2 |a3 |..

per esempio
Posti::= p_{11}, p_{12}, p_{13}…

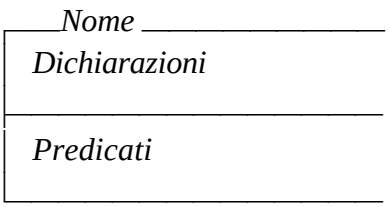
E' necessario definire un insieme per enumerazione solo quando devo far riferimento ad un particolare elemento dell'insieme
Per esempio: nella prenotazione dei posti del teatro non è necessario sapere quale posto ho prenotato, quindi non è necessario enumerare i posti
Se voglio descrivere dettagliatamente i posti del teatro, posso enumerarli aumentando la precisione della specifica, ma al tempo stesso complico eccessivamente il modello.
Generalmente gli insiemi NON si definisco per enumerazione.

SCHEMA

Uno schema descrive il sistema attraverso predicati e dichiarazioni
Una specifica è formata da più schemi, per questo è necessario associare ad ogni schema un **NOME**

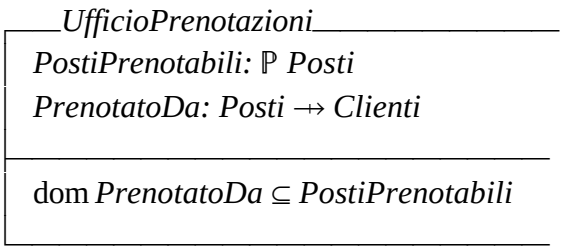
Uno schema è suddiviso in due parti:

- **DICHIARAZIONI**
- **PREDICATI**



ESEMPIO: UFFICIO PRENOTAZIONI IN Z

[*Posti, Clienti*]



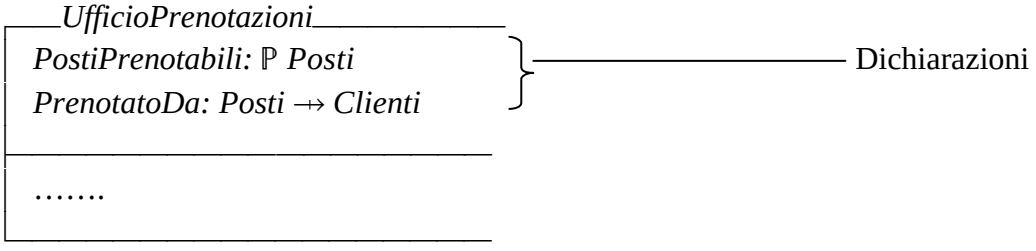
p:Posti
p è un elemento di Posti, quindi descrive un posto del teatro

postiPrenotati: ℙ Posti
postiPrenotati è un elemento dell'insieme delle parti di Posti, quindi è un sottoinsieme dei Posti

prenotazioni:Posti **X** Clienti
prenotazioni è il prodotto cartesiano dei due insiemi (l'insieme di tutte le cpooie ottenute dai due insiemi)

DICHIARAZIONI

Le **dichiarazioni** definiscono il sistema, cioè gli insiemi e le relazioni attraverso cui descriviamo il sistema



PROPRIETA' INVARIANTI

I **predicati** definiscono anche le proprietà invarianti del sistema, cioè quei vincoli che il sistema deve sempre rispettare per funzionare correttamente.

UfficioPrenotazioni

….…

dom *PrenotatoDa* ⊆ *PostiPrenotabili* ——— Predicati

RIUTILIZZO DEGLI SCHEMI

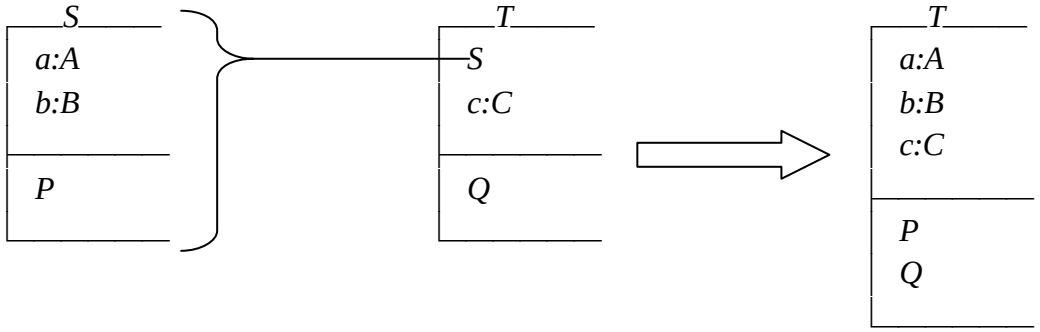
La specifica di un sistema viene descritta attraverso un insieme di schemi
Uno schema può essere descritto attraverso altri schemi

Z prevede diversi modi per poter riutilizzare gli schemi

- Inclusione fra schemi
- Operatori logici ∧ , Ú e ¬

INCLUSIONE

Attraverso l'inclusione estendo uno schema arricchendolo con nuove dichiarazioni e nuovi predicati



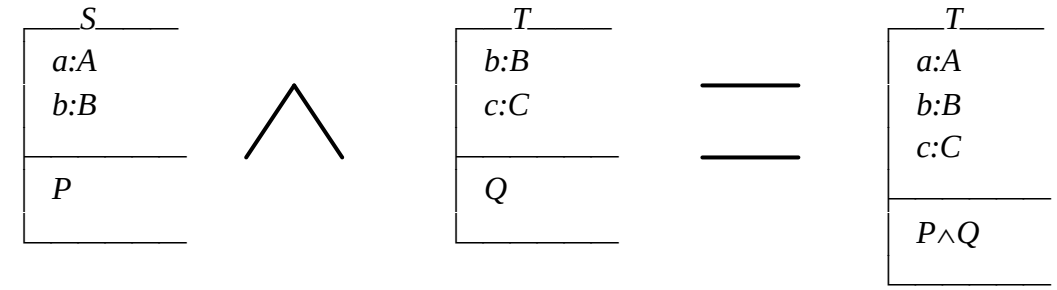
Riporto il nome di S all'interno dello schema T

CONGIUNZIONE

S ∧ *C*

Prendo le parti dichiarative di S e T e le mischio.

Metto i predicati in OR

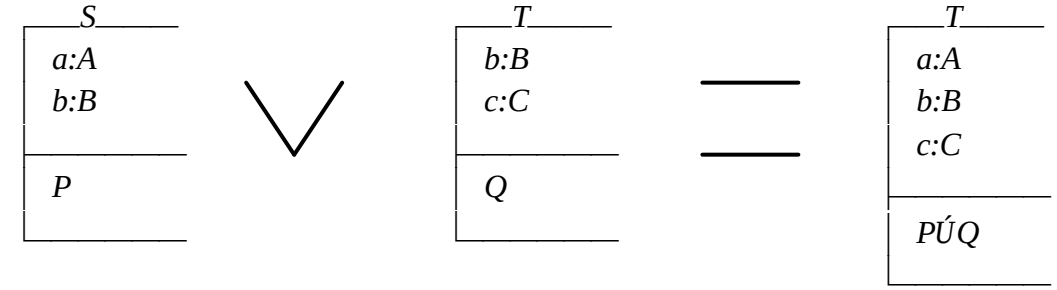


Differenza con l'inclusione: nell'inclusione non posso avere dichiarazioni uguali, qui si (e si scrivono una volta sola.

DISGIUNZIONE

$S \acute{U} T$

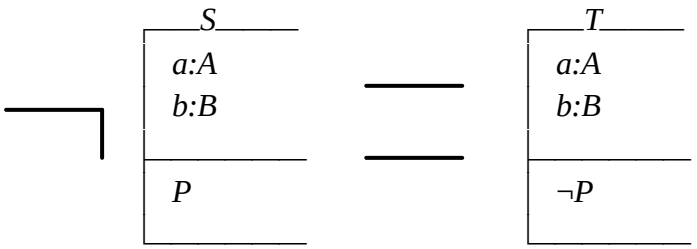
Ricopio le parti omettendo le replicazioni e metto in OR i predicati



NEGAZIONE

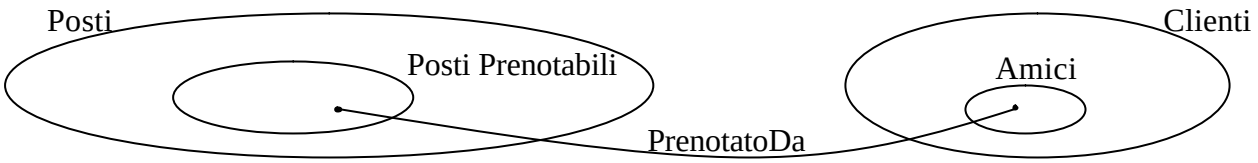
$\neg S$

Ricopio le parti dichiarative e nego i predicati



ESEMPIO: UFFICIO PRENOTAZIONI IN Z

Solo un insieme di clienti, detti **Amici**, può accedere a determinati spettacoli, detti **Prime**



TipoSpettacolo ::= *Standard* | *Prima*

UfficioPrenotazioniPrime

postiPrenotabili: ℙ *Posti*
prenotatoDa: Posti ⇸ *Clienti*
amici: ℙ *Clienti*
tipo: TipoSpettacolo

---

dom *prenotatoDa* ⊆ *postiPrenotabili*
tipo = Prima
ran *prenotatoDa* ⊆ *amici*

Possiamo indicare solo le parti in più rispetto ad UfficioPrenotazioni

TipoSpettacolo ::= *Standard* | *Prima*

UfficioPrenotazioniPrime

UfficioPrenotazioni
amici: ℙ *Clienti*
tipo: TipoSpettacolo

---

dom *prenotatoDa* ⊆ *postiPrenotabili*
tipo = Prima
ran *prenotatoDa* ⊆ *amici*

DESCRIZIONE DELLO STATO

Attraverso gli schemi abbiamo descritto:

- Lo stato del sistema
- I vincoli sullo stato

Cioè le proprietà del sistema, e non cosa succede

E' possibile descrivere:

- l'**evoluzione** dello stato (e le proprietà invarianti, cioè i vincoli sullo stato)
- le proprietà dello schema in funzione dell'evoluzione

STATO FUTURO

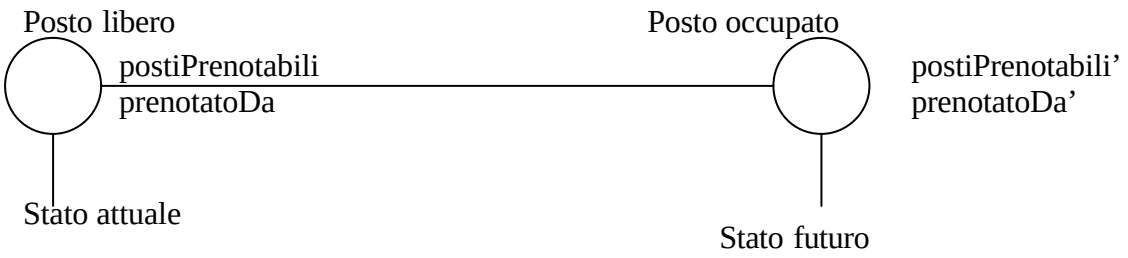
All'interno di uno schema è possibile descrive l'evoluzione di uno stato.
Durante l'evoluzione dello stato i predicati potrebbero diventare falsi (durante la transizione il sistema potrebbe essere inconsistente!)
Le proprietà invarianti devo essere vere solo nello stato corrente e in quello futuro. Non è richiesto che vengano rispettate durante l'evo luzione dello stato.

ESEMPIO: UFFICIO PRENOTAZIONI IN Z

Un cliente prenota un posto Il posto è prenotabile (*adesso*)

La coppia <posto,cliente> verrà aggiunta a venduto

Il posto non è più prenotabile (*dopo*)



OPERAZIONE

Un'operazione descrive come il sistema può evolvere
Un'operazione è descritta attraverso uno schema
Un'operazione può modificare lo stato del sistema o può semplicemente leggere e restituire dei valori
Per descrivere un'operazione si include lo schema che descrive lo stato in cui sono descritte le proprietà invarianti del sistema.

- Se l'operazione modifica lo stato, si antepone al nome dello schema il simbolo Δ
- Se l'operazione NON modifica lo stato del sistema, si antepone al nome dello schema il simbolo Ξ

AggiuntaPostoPrenotabileOk

Δ*UfficioPrenotazioni*

PostiLiberi

Ξ*UfficioPrenotazioni*

MODIFICA DI UN INSIEME

Per modificare un insieme, utilizzo gli operatori per gli insiemi

- **Aggiungere** un elemento: *postiPrenotabili'= postiPrenotabili* $\cup$ *{p?}*
- **Togliere** un elemento: *postiPrenotabili'= postiPrenotabili* $\setminus$ *{p?}*

ESEMPIO: UFFICIO PRENOTAZIONI IN Z

Alcune operazioni per poter essere eseguite possono:

- richiedere dei dati in ingresso (**?**)
- fornire dei dati in uscita (**!**)
- richiedere alcune proprietà aggiuntive sullo stato o sui dati in ingresso (pre-condizioni)

pre-condizioni: sono i vincoli affinché l'operazione possa essere eseguita

post-condizioni: vincoli che voglio siano rispettati nello stato futuro

AggiuntaPostoPrenotablieOk

Δ*UfficioPrenotazioni*
p?: Posti

---

p? ∉ postiPrenotabili — pre-condizione

postiPrenotabili' = postiPrenotabili ∪ {p?}
prenotatoDa' = prenotatoDa — post-condizioni

Nota: devo mettere *"prenotatoDa' = prenotatoDa"* perché devo sempre specificare cosa cambia e cosa non cambia.

Devo poi definire cosa succede quando ci sono errori:

Risposta ::= *OK* | *PostoGiàInserito*

AggiuntaPostoPrenotabileErrore

Ξ*UfficioPrenotazioni*
p?: Posti
r!: Risposta

---

p? ∈ postiPrenotabili
r! = PostoGiàInserito

In modo completo:

Successo

r!: Risposta

---

r! = OK

AggiuntaPostoPrenotabile ≙
AggiuntaPostoPrenotablieOk ∧ *Successo* ∨ *AggiuntaPostoPrenotabileErrore*

E' la creazione di un nuovo schema.

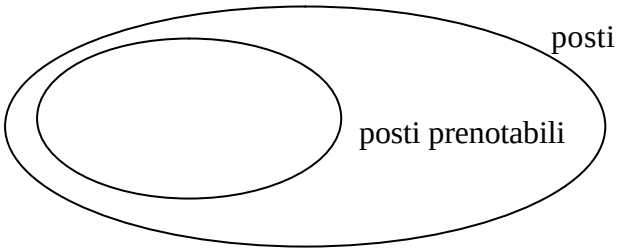
Nota: quando compongo gli schemi con ∧ e ∨, devo stare attento che uno ed un solo schema sia vero (e che non sia possibile averne due che si verificano contemporaneamente).

ESEMPIO: UFFICIO PRENOTAZIONI IN Z; RIMOZIONE DI UN POSTO PRENOTABILE

Quando si toglie un posto prenotabile è necessario aggiornare anche la funzione prenotatoDa eliminando le eventuali prenotazioni

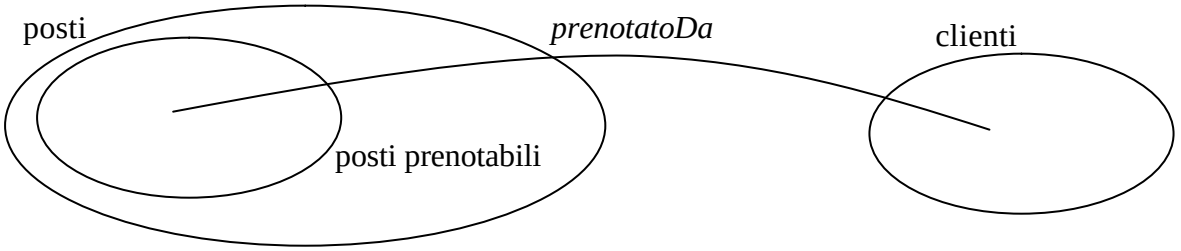
E' necessario modificare:

- l'insieme "*postiPrenotabili*"



postiPrenotabili' = postiPrenotabili \ {p?}

- Abbiamo anche una relazione da postiPrenotabili a Clienti (*prenotatoDa*)



C'è una proprietà invariante.

dom *prenotatoDa* $\acute{I}$ *postiPrenotabili*

$\{p? \mapsto c?\} \in$ *prenotatoDa*

$\{p? \mapsto c?\} \notin$ *prenotatoDa'*

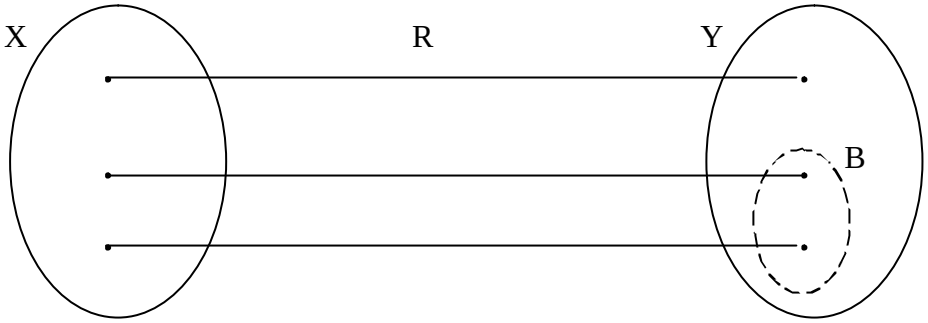
dobbiamo scrivere come varia *prenotatoDa*

prenotatoDa' = prenotatoDa \ $\{p? \mapsto c?\}$

MODIFICA DI UNA RELAZIONE\FUNZIONE

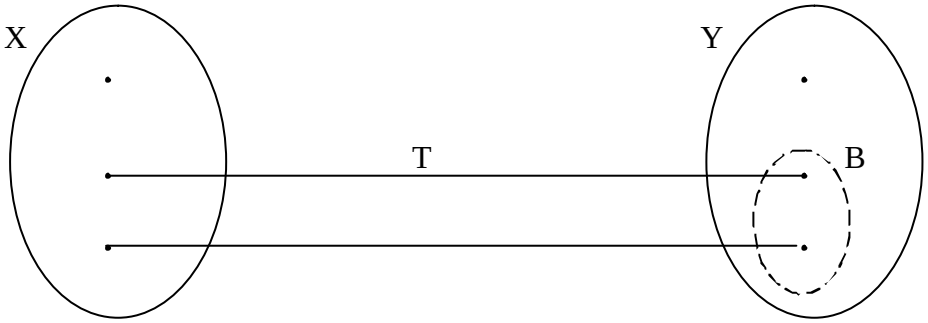
Il problema più grosso è che non sappiamo se c'è il cliente che ha prenotato e in tal caso chi è.

Esistono degli operatori atti a risolvere il problema.



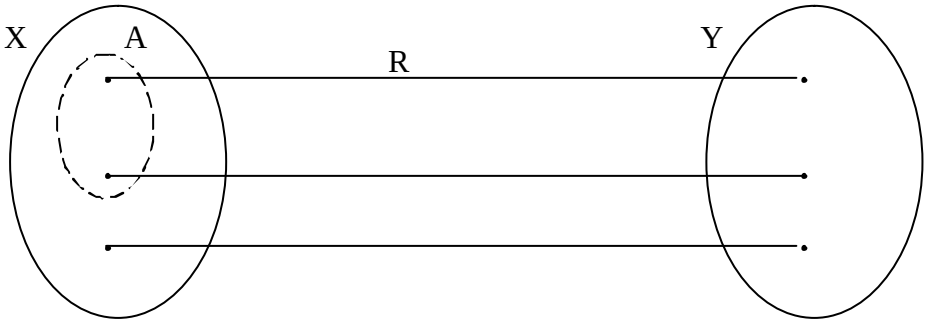
- Se scriviamo R ▷ B

otteniamo una nuova funzione T = R ▷ B

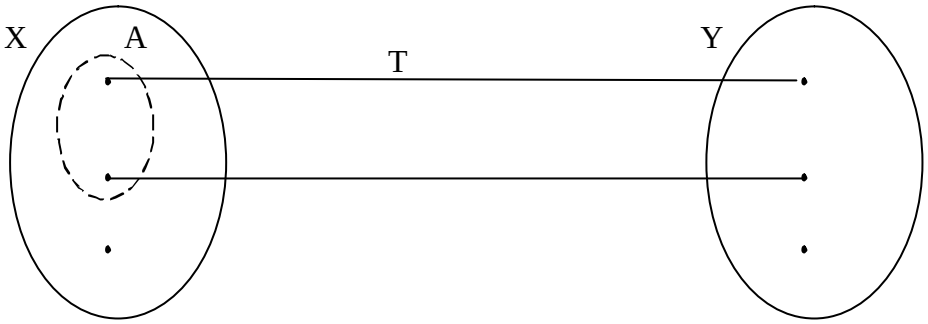


Togliamo elementi ragionando con il condominio

Possiamo anche farlo con il dominio.



- Se scriviamo A ◁ R =T :

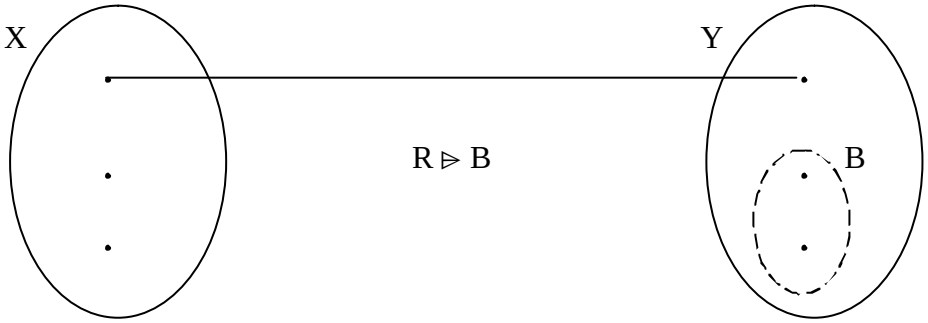


Ci sono poi gli operatori complementari:

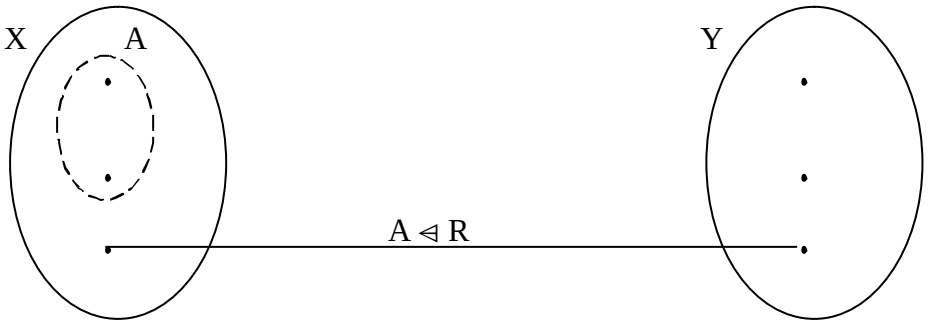
- $R \rhd B$
- $A \lhd R$

Consideriamo la funzione $R : X \rightarrow Y$

- $R \rhd B = R \rhd (Y \setminus B)$



- $A \lhd R = (X \setminus A) \lhd R$



PROPRIETA'

- $(A \lhd R) \cup (A \lhd\!\!\!- R) = R$
- $(R \rhd B) \cup (R \rhd\!\!\!- B)$

Attraverso questi due operatori, posso modificare relazioni e funzioni (non devo per forza conoscere la coppia da togliere)

ESEMPIO: UFFICIO PRENOTAZIONI IN Z

Vediamo come togliere un posto.

prenotatoDa' = {p?} $\lhd\!\!\!-$ *prenotatoDa*

restringo il dominio togliendo l'elemento *p?*

Devo modificare: sottoinsieme e relazione

RimozionePostoOk

Δ*UfficioPrenotazioni*
p?: Posti

---

p? $\in$ *postiPrenotabili*
postiPrenotabili' = postiPrenotabili $\setminus$ {*p?*}
prenotatoDa' = {p?} $\lhd\!\!\!-$ *prenotatoDa*

Completiamo con lo schema errore:

Risposta ::= *OK* | *Errore*

Fallimento

r!: Risposta

---

r! = Errore

Successo

r!: Risposta

---

r! = OK

RimozionePrenotazioneErrore

Ξ*UfficioPrenotazioni*
p?: Posti

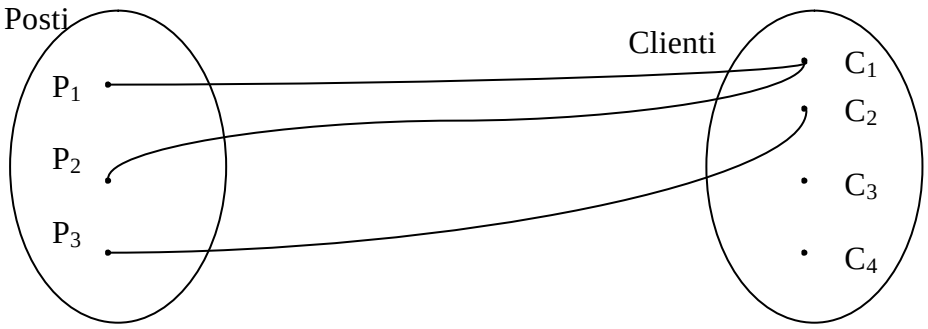
$p? \notin postiPrenotabili$

RimozionePrenotazione $\widehat{=}$
RimozionePrenotazioneOk $\wedge$ *Successo* $\vee$ *RimozionePrenotazioneErrore* $\wedge$ *Fallimento*

AGGIUNTA DI ELEMENTI

Per gli insiemi uso $\cup$, come pure per le relazioni.

Per le funzioni è pericoloso in quanto con quell'operatore ciò che otteniamo potrebbe non essere più una funzione.



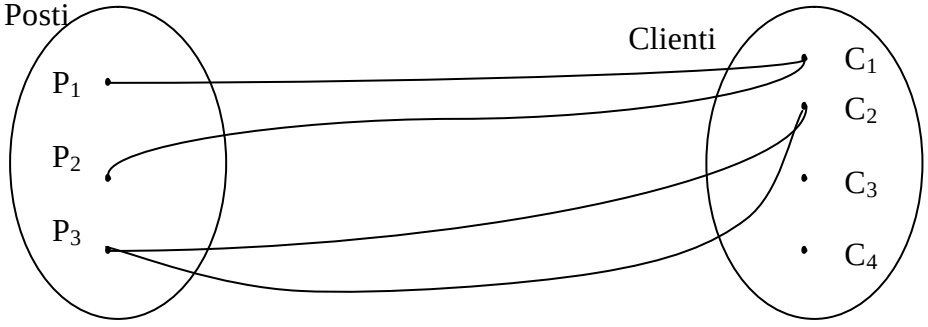
C_4 decide di prenotare P_3.

prenotatoDa è parziale (ed essendo una funzione, da ogni elemento del dominio può partire una sola freccia).

Se scriviamo

prenotatoDa' = *prenotatola* $\cup$ $\{(p_3 \mapsto c_3)\}$

da errore poiché *prenotatoDa'* non è più una funzione.



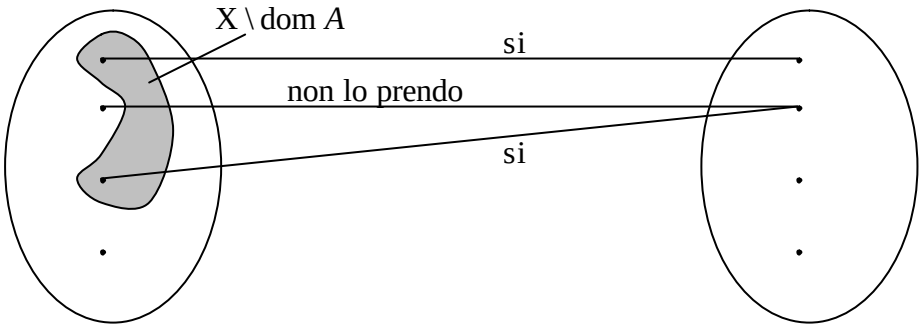
In genere con $\cup$ quello che otteniamo potrebbe non essere una funzione.

Usiamo un nuovo operatore: $\oplus$ che ci garantisce di ottenere una funzione.

Vediamo cosa fa:

$F \oplus A = A \cup \{\text{dom } A \lhd F\}$

Dobbiamo trovare la parte di F che non è definita anche in A
Prendiamo F e togliamo il dominio di A



A parole:

$F \oplus A$ = tutte le coppie di F che non appartengono al dominio di A (cioè che non sono state definite in A)

ESEMPIO: UFFICIO PRENOTAZIONI IN Z

Vediamo come scrivere la prenotazione di un posto.

p? — posto da prenotare

c? — cliente che vuole prenotare il posto

PRECONDIZIONI

il posto deve appartenere ai posti prenotabili: $p? \in postiPrenotabili$

il posto non deve essere già stato prenotato: $p? \notin prenotatoDa$

se sono vere posso prenotare il posto.

POSTCONDIZIONI

$prenotatoDa' = prenotatoDa \cup \{(p? \mapsto c?)\}$

scriviamo lo schema risultante:

PrenotazionePostoOk

Δ*UfficioPrenotazioni*
p?: Posti
c?: Clienti

$p? \in postiPrenotabili$
$p? \notin \text{dom}\ prenotatoDa$
$prenotatoDa' = prenotatoDa \cup \{(p? \mapsto c?)\}$

Descriviamo quando no va a buon fine:

PrenotazionePostoErrore

Ξ*UfficioPrenotazioni*
p?: Posti
r!: Risposta — non ci interessa sapere chi è il cliente e non lo mettiamo

---

$p? \in \text{dom}\ prenotatoDa$
$r! = postoGiàPrenotato$

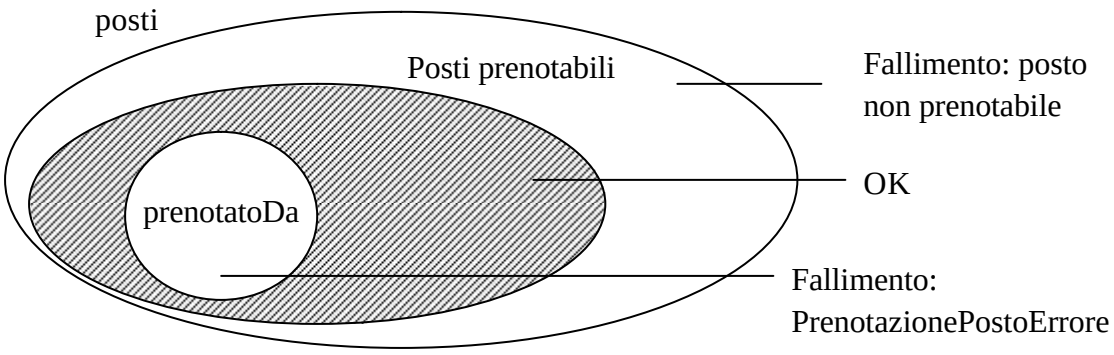
Posso avere anche il caso in cui p? non è un posto prenotabile:

PrenotazionePostoNonPrenotabile

Ξ*UfficioPrenotazioni*
p?: Posti
r!: Risposta

$p? \notin postiPrenotabili$
$r! = postoNonPrenotabile$

Graficamente:

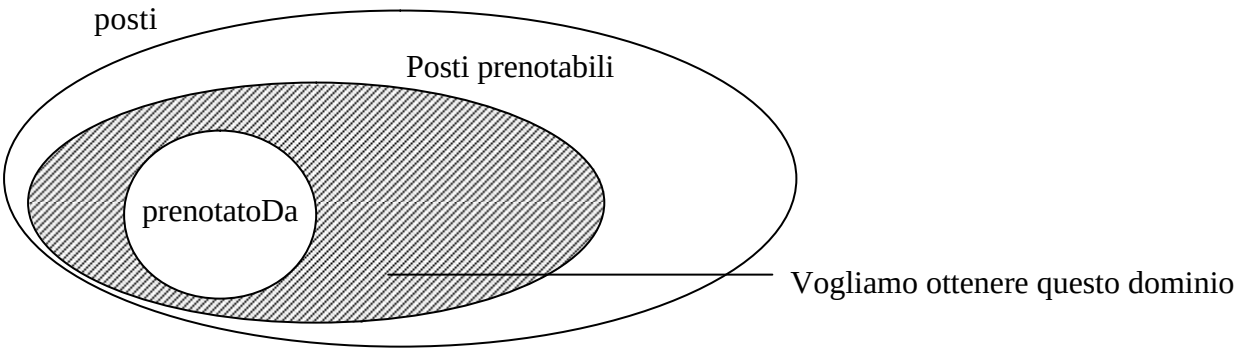


I due schemi di errore potrebbero essere uniti in uno solo (sta a noi decidere se vogliamo diversificare l'errore)

Uniamo i tre schemi definendo PrenotazionePosto.

PrenotazionePosto $\widehat{=}$
PrenotazionePostoOk $\wedge$ *Successo*
$\vee$ *PrenotazionePostoErrore*
$\vee$ *PrenotazionePostoNonPrenotabile*

Un'operazione che va a buon fine potrebbe anche essere di sola lettura.
Ad esempio se vogliamo sapere quali sono i posti ancora disponibili:

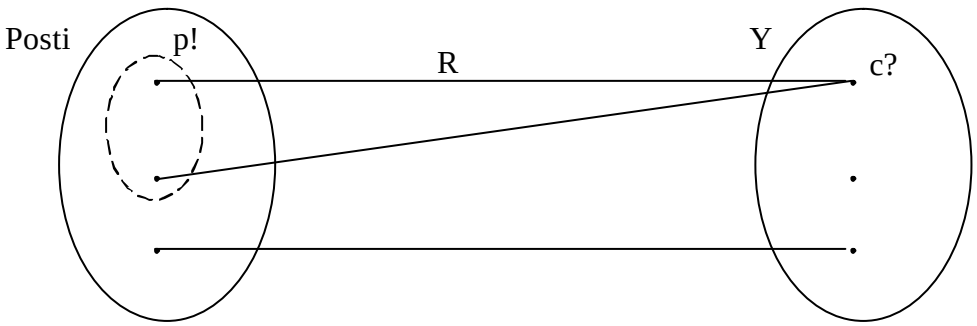


postiPrenotabili \ dom *prenotatoDa*

La definizione dello schema non ha pre-condizioni e non esistono post-condizioni in quanto lo stato non cambia.

PostiLiberi
Ξ*UfficioPrenotazioni*
p!: $\mathbb{P}$ *Posti*
---
p! = *postiPrenotabili* \ dom *prenotatoDa*

Vogliamo ora sapere quali sono i posti prenotati da un cliente



Partiamo dalla funzione *prenotatoDa*; da questa ne estraiamo una il cui range è solo c?

$prenotatoDa \rhd \{ c? \}$

di questa funzione ci interessa il dominio

$p! = dom (prenotatoDa \rhd \{ c? \})$

vediamo quando l'operazione va a buon fine: il cliente c? Deve almeno aver prenotato un posto.

$c? \in \text{ran}\, prenotatoDa$ (c'è almeno una freccia che va da p? a c?)

Definiamo quindi lo schema

PostiPrenotatiDaUnClienteOk

$\Xi UfficioPrenotazioni$
$c?: Clienti$
$p!: \mathbb{P}\ Posti$

$c? \in \text{ran}\, prenotatoDa$
$p! = \text{dom}\,(prenotatoDa \rhd \{c?\})$

PostiPrenotatiDaUnClienteErrore

$\Xi UfficioPrenotazioni$
$c?: Clienti$
$r!: Risposta$

$c? \notin \text{ran}\, prenotatoDa$
$r! = Errore$

La funzione completa è:

$PostiPrenotatiDaUnCliente \mathrel{\widehat=}$
$PostiPrenotatiDaUnClienteOk \land Successo \lor PostiPrenotatiDaUnClienteErrore$

Altra operazione: rimozione dei posti prenotati da un cliente.

In ingresso c?

Identifichiamo i posti prenotati da un cliente:

$prenotatoDa \rhd \{ c? \}$

siamo interessati ai posti che non sono stati prenotati da c?

$prenotatoDa \rhd \{ c? \}$

e vogliamo modificare *prenotatoDa*.

prenotatoDa ‘ = *prenotatoDa* ⩤ { *c?*} (è la post-condizione)

La pre-condizione è che il cliente abbia effettuato almeno una prenotazione

RimozionePostiClientiOk

Δ*UfficioPrenotazioni*
c?: Clienti

---

c? $\in$ ran *prenotatoDa*
prenotatoDa' = *prenotatoDa* ⩤ {*c?*}

RimozionePostiClientiErrore

Ξ*UfficioPrenotazioni*
c?: Clienti
r!: Risposta

---

c? $\notin$ ran *prenotatoDa*
r! = *Errore*

RimozionePostiClienti $\widehat{=}$
RimozionePostiClientiOk $\wedge$ *Successo* $\vee$ *RimozionePostiClientiErrore*

ESEMPIO COMPLETO: UFFICIO PRENOTAZIONI IN Z

[*Posti, Clienti*]

UfficioPrenotazioni

$$
\begin{array}{l}
postiPrenotabili: \mathbb{P}\ Posti \\
prenotatoDa: Posti \pfun Clienti \\
\hline
\mathrm{dom}\ prenotatoDa \subseteq postiPrenotabili
\end{array}
$$

TipoSpettacolo ::= *Standard* | *Prima*

UfficioPrenotazioniPrime

$$
\begin{array}{l}
UfficioPrenotazioni \\
amici: \mathbb{P}\ Clienti \\
tipo: TipoSpettacolo \\
\hline
\mathrm{dom}\ prenotatoDa \subseteq postiPrenotabili \\
tipo = Prima \\
\mathrm{ran}\ prenotatoDa \subseteq amici
\end{array}
$$

AggiuntaPostoPrenotablieOk

$$
\begin{array}{l}
\Delta UfficioPrenotazioni \\
p?: Posti \\
\hline
p? \notin postiPrenotabili \\
postiPrenotabili' = postiPrenotabili \cup \{p?\} \\
prenotatoDa' = prenotatoDa
\end{array}
$$

Risposta ::= *OK*
| *PostoGiàInserito*
| *Errore*
| *postoGiàPrenotato*
| *postoNonPrenotabile*

AggiuntaPostoPrenotabileErrore

Ξ*UfficioPrenotazioni*
p?: Posti
r!: Risposta

$p? \in postiPrenotabili$
$r! = PostoGiàInserito$

Fallimento

r!: Risposta

$r! = Errore$

Successo

r!: Risposta

$r! = OK$

$$AggiuntaPostoPrenotabile \mathrel{\widehat=} AggiuntaPostoPrenotablieOk \land Successo \lor AggiuntaPostoPrenotabileErrore$$

RimozionePrenotazioneOk

Δ*UfficioPrenotazioni*
p?: Posti

$p? \in postiPrenotabili$
$postiPrenotabili' = postiPrenotabili \setminus \{p?\}$
$prenotatoDa' = \{p?\} \lhd prenotatoDa$

RimozionePrenotazioneErrore

Ξ*UfficioPrenotazioni*
p?: Posti

$p? \notin postiPrenotabili$

$$RimozionePrenotazione \mathrel{\widehat=} RimozionePrenotazioneOk \land Successo \lor RimozionePrenotazioneErrore \land Fallimento$$

PrenotazionePostoOk

Δ*UfficioPrenotazioni*
p?: Posti
c?: Clienti

---

p? $\in$ *postiPrenotabili*
p? $\notin$ dom *prenotatoDa*
prenotatoDa' = *prenotatoDa* $\cup$ {(*p?* $\mapsto$ *c?*)}

PrenotazionePostoErrore

Ξ*UfficioPrenotazioni*
p?: Posti
r!: Risposta

---

p? $\in$ dom *prenotatoDa*
r! = *postoGiàPrenotato*

PrenotazionePostoNonPrenotabile

Ξ*UfficioPrenotazioni*
p?: Posti
r!: Risposta

---

p? $\notin$ *postiPrenotabili*
r! = *postoNonPrenotabile*

PrenotazionePosto $\widehat{=}$
PrenotazionePostoOk $\wedge$ *Successo*
$\vee$ *PrenotazionePostoErrore*
$\vee$ *PrenotazionePostoNonPrenotabile*

PostiLiberi

Ξ*UfficioPrenotazioni*
p!: $\mathbb{P}$ *Posti*

---

p! = *postiPrenotabili* \ dom *prenotatoDa*

PostiPrenotatiDaUnClienteOk

Ξ*UfficioPrenotazioni*
c?: Clienti
p!: $\mathbb{P}$ *Posti*

$c? \in \operatorname{ran} prenotatoDa$
$p! = \operatorname{dom}(prenotatoDa \rhd \{c?\})$

PostiPrenotatiDaUnClienteErrore

Ξ*UfficioPrenotazioni*
c?: Clienti
r!: Risposta

$c? \notin \operatorname{ran} prenotatoDa$
$r! = Errore$

$PostiPrenotatiDaUnCliente \mathrel{\widehat=}$
$PostiPrenotatiDaUnClienteOk \land Successo \lor PostiPrenotatiDaUnClienteErrore$

RimozionePostiClientiOk

Δ*UfficioPrenotazioni*
c?: Clienti

$c? \in \operatorname{ran} prenotatoDa$
$prenotatoDa' = prenotatoDa \rhd \{c?\}$

RimozionePostiClientiErrore

Ξ*UfficioPrenotazioni*
c?: Clienti
r!: Risposta

$c? \notin \operatorname{ran} prenotatoDa$
$r! = Errore$

$RimozionePostiClienti \mathrel{\widehat=}$
$RimozionePostiClientiOk \land Successo \lor RimozionePostiClientiErrore$

ESEMPIO: ESTENZIONE UFFICIO PRENOTAZIONI IN Z

Consideriamo l'ufficio prenotazioni per più spettacoli.

Dobbiamo:

- gestire le prenotazioni per ogni spettacolo
- gestire i posti prenotabili per ogni spettacolo

Spettacoli diversi ⇔ posti prenotabili diversi.

Partiamo dalla definizione dei tipi utilizzati

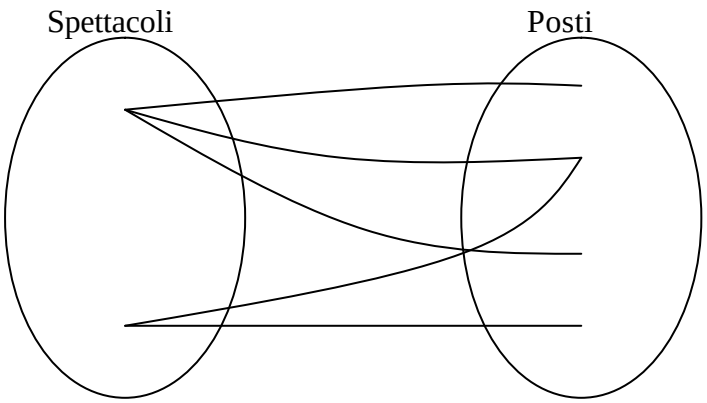
[Posti, Clienti, Spettacoli]

Spettacoli contiene tutti i possibili spettacoli del teatro.

1° PASSO: Definire tutti i posti prenotabili del teatro (varia a seconda dello spettacolo), non è più un puro sottoinsieme di Posti.

Definiamo una relazione tra postiPrenotabili e Spettacolo:

postiPrenotabili : ℙ (Spettacoli X Posti)



Non è una funzione ma una relazione poiché ad uno spettacolo corrispondono più posti prenotabili.

E' un insieme di coppie che collegano lo spettacolo al sottoinsieme di posti prenotabili per quello spettacolo.

2° PASSO: Definiamo prenotatoDa

Un posto può essere prenotato da più clienti a patto che appartenga a spettacoli diversi.

prenotatoDa : (Spettacoli X Posti) → Clienti

Dato lo spettacolo ed il posto, ottengo il cliente che ha prenotato quel posto.

Fissato lo spettacolo è una funzione poiché c'è solo un cliente che può prenotare quel posto.

E' parziale poiché non è definita per tutti gli spettacoli e posso avere posti non prenotabili

Non è iniettiva in quanto un cliente può prenotare più posti

Non è suriettiva in quanto non è definita per tutti i clienti (posso avere clienti che non hanno prenotato)

3° PASSO: Definisco la proprietà invariante.
Voglio che un cliente possa prenotare un posto solo se questo risulta prenotabile per quel determinato spettacolo. (il vincolo è che sia definita solo per i posti prenotabli)

dom *prenotatoDa* $\in$ *postiPrenotabili*

Scrivo lo schema dell'ufficio prenotazioni:

UfficioPrenotazioniPlus

$postiPrenotabili: \mathbb{P}(Spettacoli \times Posti)$
$prenotatoDa: Spettacoli \times Posti \rightarrow Clienti$

$\text{dom}\ prenotatoDa \subseteq postiPrenotabili$

Aggiunta di un posto prenotabile:

Risposta ::= *OK* | *Errore*

Successo

$r!: Risposta$

$r! = OK$

AggiungiPostoOk

$\Delta UfficioPrenotazioniPlus$
$p?: Posti$
$sp?: Spettacoli$

$sp? \mapsto p? \notin postiPrenotabili$
$postiPrenotabili' = postiPrenotabili \cup \{(sp? \mapsto p?)\}$
$prenotatoDa' = prenotatoDa$

Per inserire un posto tra quelli prenotabili, non deve già esserlo

AggiungiPostoErrore

Ξ*UfficioPrenotazioniPlus*
p?: Posti
sp?: Spettacoli
r!: Risposta

$sp? \mapsto p? \in postiPrenotabili$
$r! = Errore$

$AggiungiPosto \mathrel{\widehat=} AggiungiPostoOk \land Successo \lor AggiungiPostoErrore$

Vediamo la rimozione di un posto:

RimozionePostoOk

Δ*UfficioPrenotazioniPlus*
p?: Posti
sp?: Spettacoli

$sp? \mapsto p? \in postiPrenotabili$
$postiPrenotabili' = postiPrenotabili \setminus \{(sp? \mapsto p?)\}$
$prenotatoDa' = \{(sp? \mapsto p?)\} \lhd prenotatoDa$

può accadere che togliamo dai post iprenotabili un posto già prenotato da qualcuno, in quel caso occorre togliere l'elemento dal dominio.

RimozionePostoErrore

Ξ*UfficioPrenotazioniPlus*
p?: Posti
sp?: Spettacoli
r!: Risposta

$sp? \mapsto p? \notin postiPrenotabili$
$r! = Errore$

$RimozionePosto \mathrel{\widehat=} RimozionePostoOk \land Successo \lor RimozionePostoErrore$

Rimozione di uno spettacolo

Occorre:

- togliere tutti i posti prenotabili per quello spettacolo sp?
- Togliere tutte le prenotazioni

RimozioneSpettacoloOk

Δ*UfficioPrenotazioniPlus*
sp?: Spettacoli

$sp? \in \mathrm{dom}\, postiPrenotabili$
$postiPrenotabili' = \{sp?\} \lhd postiPrenotabili$
$prenotatoDa' = (\{sp?\} \times Posti) \lhd prenotatoDa$

Togliamo i posti relativi allo spettacolo

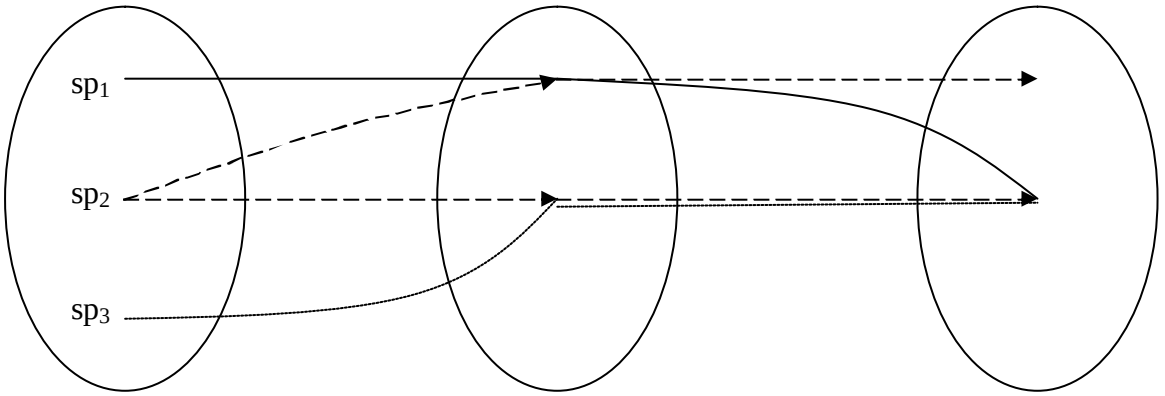
Vogliamo eliminare tutte le possibili coppie da quello spettacolo e i posti del teatro

RimozioneSpettacoloErrore

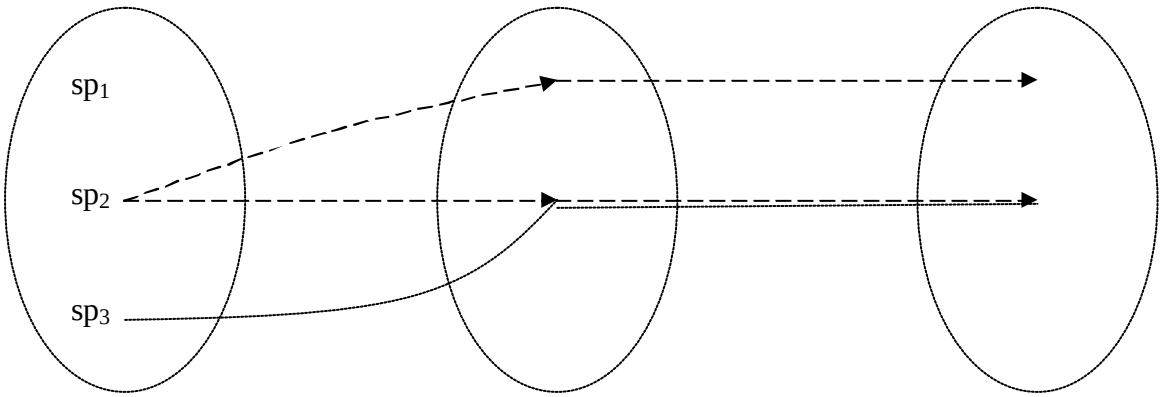
Ξ*UfficioPrenotazioniPlus*
sp?: Spettacoli
r!: Risposta

$sp? \notin \mathrm{dom}\, postiPrenotabili$
r! = Errore

RimozioneSpettacolo $\widehat{=}$
RimozioneSpettacoloOk $\wedge$ *Successo* $\vee$ *RimozioneSpettacoloErrore*



Se vogliamo eliminare sp_1:



Ed infine la prenotazione di un posto:

PrenotazionePostoOk

Δ*UfficioPrenotazioniPlus*
c?: Clienti
p?: Posti
sp?: Spettacoli

$sp? \mapsto p? \in postiPrenotabili$
$sp? \mapsto p? \notin \mathrm{dom}\, prenotatoDa$
$prenotatoDa' = prenotatoDa \oplus \{(sp? \mapsto p? \mapsto c?)\}$

PrenotazionePostoErrore

Ξ*UfficioPrenotazioniPlus*
c?: Clienti
p?: Posti
sp?: Spettacoli
r!: Risposta

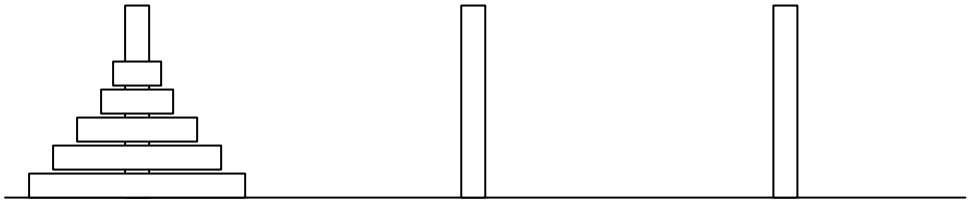
$sp? \mapsto p? \notin postiPrenotabili \lor sp? \mapsto p? \notin \mathrm{dom}\, prenotatoDa$
$r! = Errore$

$PrenotazionePosto \mathrel{\widehat=} PrenotazionePostoOk \land Successo \lor PrenotazionePostoErrore$

ESEMPIO: TORRI DI HANOI

Modellare in Zeta le torri di Hanoi.

- Costituita da tre pioli.
- Su ogni piolo ho tre dischi
- Scopo: portare i dischi dal piolo A al piolo C



Regole:

- posso spostare un disco se è in cima al piolo
- posso metterlo su un piolo vuoto oppure su un disco più grande

MODELLARE:

- lo stato del sistema
    - tre pioli
    - come i dischi sono distinti sui pioli
    - (visione statica, proprietà invarianti)
- spostamento di un disco da un piolo all'altro
    - (è un operazione che produce un cambiamento di stato)

Stato del sistema

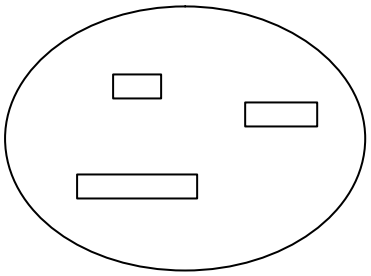
Abbiamo dei dischi impilati su dei pioli.

Descriviamo:

- tre pioli
- come i dischi sono distinti sui pioli

Non descriviamo l'ordine dei dischi.

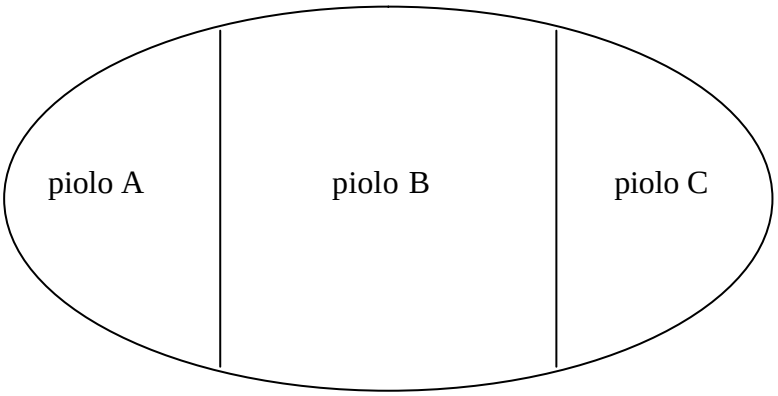
Semplifichiamo il modello



Daremo solo il diametro del disco e non l'ordine in cui sono impilati

Parte dichiarativa

- insiemi
    - dischi
    - pioli (li descriviamo come relazioni di appartenenza)



vediamo come i dischi sono distribuiti sui pioli

- funzioni
    - dobbiamo descrivere le dimensioni del disco
      DiametroDisco : Dischi $\rightarrow \mathbb{N}$
      è una funzione totale perché è definita per tutti i dischi

Definiamo lo stato

[*Dischi*]

Pioli

$PioloA: \mathbb{P}\ Dischi$
$PioloB: \mathbb{P}\ Dischi$
$PioloC: \mathbb{P}\ Dischi$
$DiametroDisco: Dischi \rightarrow \mathbb{N}$

$PioloA \cap PioloB = \varnothing$
$PioloA \cap PioloB = \varnothing$
$PioloA \cap PioloB = \varnothing$
} un disco può stare su un solo piolo

$PioloA \cup PioloB \cup PioloC = Dischi$ — i dischi non vengono persi

$\forall d1: Dischi \bullet \forall d2: Dischi \bullet DiametroDisco\ d1 = DiametroDisco\ d2 \Rightarrow d1 = d2$

se due dischi hanno lo stesso diametro sono uguali. ($\bullet$ = “tale che”)

Spostamento di un disco da A a B

Precondizioni:

- Su A:
  - il disco deve stare su A
  - il disco deve essere più piccolo sul piolo A
- Su B:
  - Non deve esistere un disco più piccolo

ImpilaDaAaB_OK

$\Delta Pioli$
$d?: Dischi$

$d? \in PioloA$
$\forall d1: Dischi \bullet d1 \in PioloA \land DiametroDisco\ d1 \geqslant DiametroDisco\ d?$
$\forall d2: Dischi \bullet d2 \in PioloB \land DiametroDisco\ d2 \geqslant DiametroDisco\ d?$
$PioloA' = PioloA \setminus \{d?\}$
$PioloB' = PioloB \cup \{d?\}$

ESEMPIO: FILE SHARING

Identifichiamo:

- insiemi
- relazioni
- proprietà invarianti

che descrivono lo stato del sistema

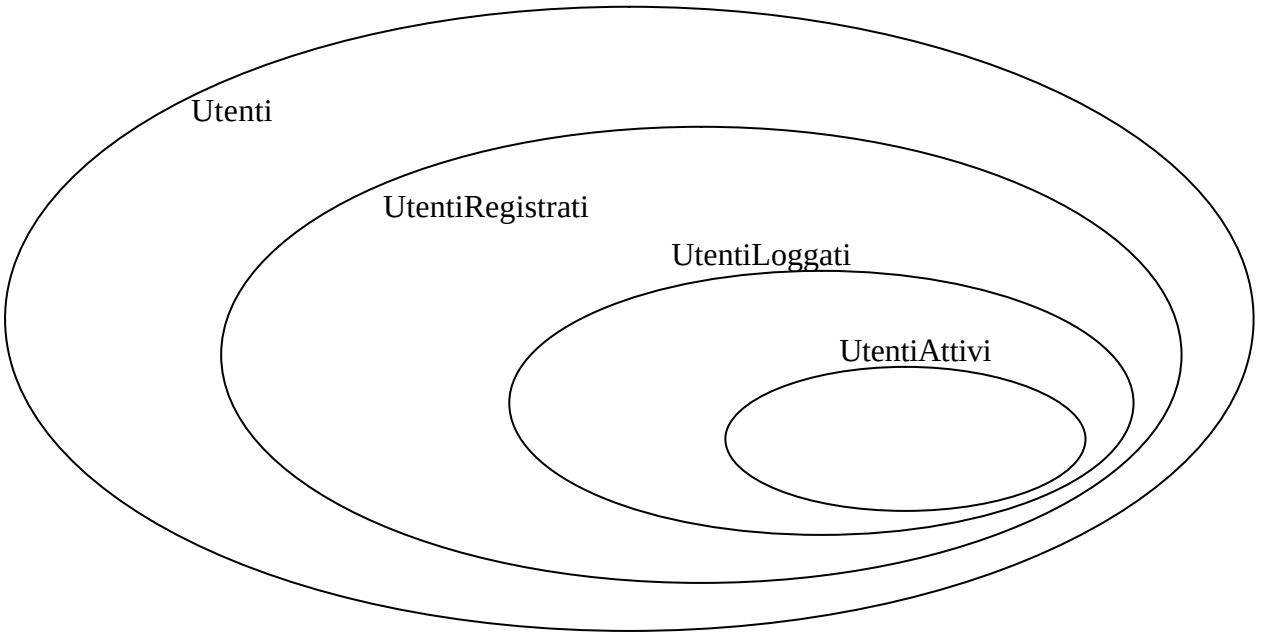
- operazioni
  - registrazione
  - login
  - ricerca

che descrivono come il sistema può evolvere

INSIEMI

- Utenti tutti i possibili utenti (non solo quelli registrati)
- Canzoni insieme di tutte le possibili canzoni condivisibili
- Password insieme di tutte le possibili password

Vediamo gli utenti:



FUNZIONI

- Funzione per controllare la password

  Check : Utenti ⇸ Password

  (non è totale perché solo gli utenti registrati hanno la password)

- Per le canzoni

  Share : Utenti ⇸ ℙ Canzoni

  dice quali canzoni sono condivise da un utente (⇸ ℙ Canzoni, perché un utente può condividere un sottoinsieme di tutte le canzoni)

STATO DEL SISTEMA

[*Utenti, Canzoni, Password*]

FileSharing

UtentiRegistrati: ℙ *Utenti* *UtentiLoggati:* ℙ *Utenti* *UtentiAttivi:* ℙ *Utenti*	descrivo gli utenti
Check: Utenti ⇸ Password *Share: Utenti ⇸* ℙ *Canzoni*	
UtentiLoggati ⊆ *UtentiRegistrati*	un utente può loggarsi solo se registrato
UtentiAttivi ⊆ *UtentiLoggati*	
dom *Check* = *UtentiRegistrati*	Check è definita solo per i registrati
dom *Share* = *UtentiLoggati*	Share solo per i Loggati

REGISTRAZIONE

Un utente può registrarsi solo se non è registrato.

Risposta ::= *OK* | *UtenteGiàRegistrato*

Successo

R!: Risposta
R! = OK

Registrazione_Ok

$\Delta FileSharing$
$u?: Utenti$
$pw?: Password$

$u? \notin UtentiRegistrati$
$UtentiRegistrati' = UtentiRegistrati \cup \{u?\}$
$Check' = Check \oplus \{(u? \mapsto pw?)\}$
$UtentiLoggati' = UtentiLoggati$
$UtentiAttivi' = UtentiAttivi$
$Share' = Share$

Registrazione_Fallimento

$\Xi FileSharing$
$u?: Utenti$
$r!: Risposta$

$u? \in UtentiRegistrati$
$r! = UtenteGiàRegistrato$

$Registrazione \mathrel{\widehat{=}} Registrazione\_Ok \wedge Successo \vee Registrazione\_Fallimento$

DE-REGISTRAZIONE

Precondizione: Utente? ∈ UtentiRegistrati (l'utente deve essere registrato)

De_Registrazione_Ok

$\Delta FileSharing$
$u?: Utenti$

$u? \in UtentiRegistrati$
$UtentiRegistrati' = UtentiRegistrati \setminus \{u?\}$
$Check' = \{u?\} \lhd Check$
$UtentiLoggati' = UtentiLoggati \setminus \{u?\}$
$UtentiAttivi' = UtentiAttivi \setminus \{u?\}$
$Share' = \{u?\} \lhd Share$

De_Registrazione_Fallimento

$\Xi FileSharing$
$u?: Utenti$
$r!: Risposta$

$u? \notin UtentiRegistrati$
$r! = UtenteNonRegistrato$

De_Registrazione $\widehat{=}$ *De_Registrazione_Ok* $\wedge$ *Successo* $\vee$ *De_Registrazione_Fallimento*

LOGIN

Definiamo due schemi:

- uno per il cambiamento di stato per le canzoni e per gli insiemi
- uno per il controllo della password

Più schemi per le precondizioni poiché possono avere login con successo con precondizioni diverse.

- Login Normale
    - utente? $\in$ UtentiRegistrati
    - utente? $\notin$ UtentiLoggati
- Recovery Login
    - utente? $\in$ UtentiRegistrati
    - utente? $\in$ UtentiLoggati
    - utente? $\notin$ UtentiAttivi

Vediamo prima com'è gestito il cambiamento di stato.

LoginUtente

$\Delta FileSharing$
$u?: Utenti$
$c?: \mathbb{P}\ Canzoni$

$UtentiLoggati' = UtentiLoggati \cup \{u?\}$
$UtentiAttivi = UtentiAttivi \cup \{u?\}$
$UtentiRegistrati' = UtentiRegistrati$
$Check' = Check$
$Share' = Share \oplus \{(u? \mapsto c?)\}$

ControllaPassword

FileSharing
u?: Utenti
pw?: Password

Check u? = pw?

LoginUtente descrive il cambiamento di stato

ControllaPassword descrive una precondizione di login.

A partire dai due schemi possiamo definire le operazioni di login per l'utente.

LoginNormale_Ok

LoginUtente
ControllaPassword

$u? \in UtentiRegistrati$
$u? \notin UtentiLoggati$

LoginRipristino_Ok

LoginUtente
ControllaPassword

$u? \in UtentiRegistrati$
$u? \in UtentiLoggati$
$u? \notin UtentiAttivi$

Vediamo quando il Login può fallire.

- password errata
- $u? \notin UtentiRegistrati$
- $u? \in UtentiLoggati \wedge u? \in UtentiAttivi$

LoginPasswordErrata

Ξ*FileSharing*
u?: Utenti
pw?: Password
r!: Risposta

---

u? $\in$ *UtentiRegistrati*
Check u? $\neq$ *pw?*
r! = *PasswordErrata*

LoginUtenteNonRegistrato

Ξ*FileSharing*
u?: Utenti
r!: Risposta

---

u? $\notin$ *UtentiRegistrati*
r! = *UtenteNonRegistrato*

LoginUtenteGiaConnesso

Ξ*FileSharing*
u?: Utenti
r!: Risposta

---

u? $\notin$ *UtentiAttivi*
r! = *UtenteGiaConnesso*

Login $\widehat{=}$
$(LoginNormale\_Ok \lor LoginRipristino\_Ok) \land Successo$
$\lor$ *LoginPasswordErrata*
$\lor$ *LoginUtenteNonRegistrato*
$\lor$ *LoginUtenteGiaConnesso*

LOGOUT

Logout_Ok

$\Delta FileSharing$
$u?: Utenti$

$u? \in UtentiLoggati$
$UtentiLoggati' = UtentiLoggati \setminus \{u?\}$
$UtentiAttivi = UtentiAttivi \setminus \{u?\}$
$Share' = \{u?\} \lhd Share$
$UtentiRegistrati' = UtentiRegistrati$
$Check' = Check$

LogoutNonLoggato

$\Xi FileSharing$
$u?: Utenti$
$r!: Risposta$

$u? \notin UtentiLoggati$
$r! = UtenteNonLoggato$

$Logout \triangleq Logout\_Ok \wedge Successo \vee LogoutNonLoggato$

NOTA: Alla fine Risposta risulta essere:

Risposta ::= *OK*
| *UtenteGiàRegistrato*
| *UtenteNonRegistrato*
| *PasswordErrata*
| *UtenteGiaConnesso*
| *UtenteNonLoggato*

RICERCA

Vediamo adesso le operazione di search e download; hanno delle precondizioni in comune.

u? ∈ UtentiRegistrati; u? ∈ UtentiLoggati; u? ∈ UtentiAttivi

Definiamo gli schemi per le precondizioni.

UtenteOnLine

FileSharing
u?: Utenti

---

u? ∈ UtentiAttivi

UtenteOffLine

FileSharing
u?: Utenti
r!: Risposta

---

u? ∉ UtentiAttivi
u? ∈ UtentiLoggati
r! = Utente_Off_Line

UtenteNonLoggato

FileSharing
u?: Utenti
r!: Risposta

---

u? ∉ UtentiAttivi
u? ∉ UtentiLoggati
u? ∈ UtentiRegistrati
r! = Utente_Non_Loggato

UtenteNonRegistrato

FileSharing
u?: Utenti
r!: Risposta

---

u? ∉ UtentiRegistrati
r! = Utente_Non_Registrato

Quindi l’operazione di Search

OperazioneSearch

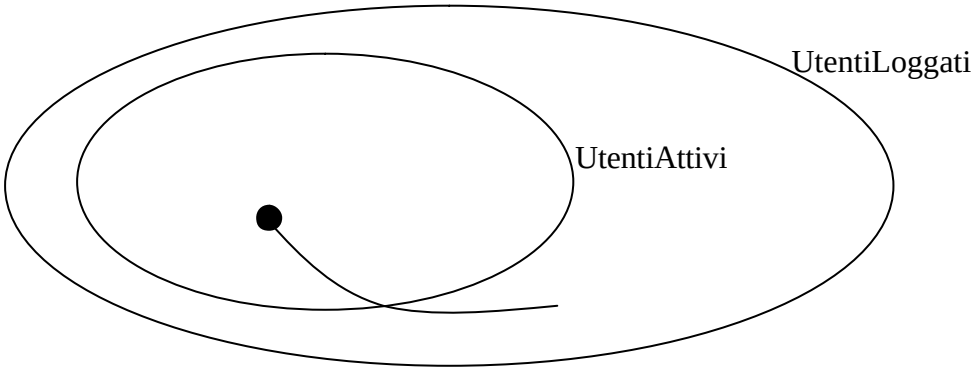
$\Xi FileSharing$
c?: Canzoni
u?: Utenti
lista!: $\mathbb{P}$ *Utenti*

$lista! = \{\ ui: Utenti \mid ui \in UtentiLoggati \land c? \in Share\ ui \land ui \neq u?\ \}$

In maniera completa:

$Search \mathrel{\widehat{=}}$
$OperazioneSearch \land UtenteOnLine \land Successo$
$\lor\ UtenteOffLine$
$\lor\ UtenteNonLoggato$
$\lor\ UtenteNonRegistrato$

CADUTA DI CONNESSIONE COL SERVER



DisconnessioneUtenteOk

$\Delta FileSharing$
u?: Utenti

$UtentiRegistrati' = UtentiRegistrati$
$UtentiLoggati' = UtentiLoggati$
$UtentiAttivi' = UtentiAttivi \setminus \{u?\}$
$Check' = Check$
$Share' = Share$

L’utente quando si disconnette non è in grado di avvisare il server

$DisconnessioneUtente \triangleq$
$DisconnessioneUtenteOk \land UtenteOnLine \land Successo$
$\lor UtenteOffLine$
$\lor UtenteNonLoggato$
$\lor UtenteNonRegistrato$

AGGIORNA LISTA CANZONI

Una volta loggiato, l'utente può aggiornare la lista delle canzoni che sta condividendo

AggiornaCanzoniOk

$\Delta FileSharing$
$u?: Utenti$
$c?: \mathbb{P}\ Canzoni$

$UtentiRegistrati' = UtentiRegistrati$
$UtentiLoggati' = UtentiLoggati$
$UtentiAttivi' = UtentiAttivi$
$Check' = Check$
$Share' = Share \oplus \{(u? \mapsto c?)\}$

Modifico Share per quel particolare utente che condividerà solo

$AggiornaCanzoni \triangleq$
$AggiornaCanzoniOk \land UtenteOnLine \land Successo$
$\lor UtenteOffLine$
$\lor UtenteNonLoggato$
$\lor UtenteNonRegistrato$

DOWNLOAD

Precondizioni:

u? chi vuole scaricare
ur? da chi scarica (utente remoto)

u?, ru? $\in$ UtentiAttivi
c? $\in$ Share ur?

L'operazione farà si che u? metta in condivisione la canzone appena scaricata.

c? $\in$ Share' u?

DownloadOk

$\Delta FileSharing$
$u?: Utenti$
$ur?: Utenti$
$c?: Canzoni$

$c? \in Share\ ur?$
$Share' = Share \oplus \{(u? \mapsto Share\ u? \cup \{c?\})\}$

UtenteRemotoOffLine

$FileSharing$
$ur?: Utenti$
$r!: Risposta$

$ur? \notin UtentiAttivi$
$r! = Utente\_Remoto\_Off\_Line$

UtenteRemotoOnLine

$FileSharing$
$ur?: Utenti$

$ur? \in UtentiAttivi$

DownloadCanzoneNonDisponibile

$\Xi FileSharing$
$u?: Utenti$
$ur?: Utenti$
$c?: Canzoni$
$r!: Risposta$

$c? \notin Share\ ur?$
$r! = Canzone\_Non\_Disponibile$

DownloadUtenteRemotoOffLine

$\Delta FileSharing$
$u?: Utenti$
$ur?: Utenti$
$c?: Canzoni$
$r!: Risposta$

$u? \in UtentiAttivi$
$ur? \notin UtentiAttivi$
$UtentiLoggati' = UtentiLoggati \setminus \{ur?\}$
$UtentiRegistrati' = UtentiRegistrati$
$UtentiAttivi' = UtentiAttivi$
$Check' = Check$
$Share' = \{ur?\} \lhd Share$
$r! = Utente\_Remoto\_Off\_Line$

Tolgo ur? dal dominio di Share

$Download \ \widehat{=}$
$DownloadOk \land UtenteOnLine \land Successo$
$\lor DownloadCanzoneNonDisponibile \land UtenteOnLine \land UtenteRemotoOnLine$
$\lor UtenteOffLine$
$\lor UtenteNonLoggato$
$\lor UtenteNonRegistrato$
$\lor DownloadUtenteRemotoOffLine$

FUNZIONI CON SCHEMI COME PARAMETRO

Posso passare uno schema come parametro ad una funzione

ESEMPIO: CALCOLARE IL NUMERO DI CANZONI CONDIVISE

$Shared == (\lambda FileSharing \bullet \#\{C: Canzoni \mid u \in Utenti \land S \in Share\ u\})$

Non ci interessa FileSharing come una black box, ma ci interessa il contenuto

Cardinalità

Vediamo come può essere usata:

Definiamo un nuovo schema che restituisce le canzoni condivise dagli utenti

CanzoniCondiviseDagliUtenti

$\Xi FileSharing$

$numeroCanzoni! : \mathbb{N}$

$numeroCanzoni! = Shared\ \theta\ FileSharing$

Passiamo il contenuto di FileSharing

ESEMPIO COMPLETO

[*Utenti, Canzoni, Password*]

FileSharing

$UtentiRegistrati: \mathbb{P}\ Utenti$
$UtentiLoggati: \mathbb{P}\ Utenti$
$UtentiAttivi: \mathbb{P}\ Utenti$
$Check: Utenti \pfun Password$
$Share: Utenti \pfun \mathbb{P}\ Canzoni$

$UtentiLoggati \subseteq UtentiRegistrati$
$UtentiAttivi \subseteq UtentiLoggati$
$\mathrm{dom}\ Check = UtentiRegistrati$
$\mathrm{dom}\ Share = UtentiLoggati$

Risposta ::= *OK*
| *Utente_Già_Registrato*
| *Utente_Non_Registrato*
| *Password_Errata*
| *Utente_Già_Connesso*
| *Utente_Non_Loggato*
| *Utente_Off_Line*
| *Utente_Remoto_Off_Line*
| *Canzone_Non_Disponibile*

Successo

$R!: Risposta$

$R! = OK$

RegistrazioneOk

$\Delta FileSharing$
$u?: Utenti$
$pw?: Password$

$u? \notin UtentiRegistrati$
$UtentiRegistrati' = UtentiRegistrati \cup \{u?\}$
$Check' = Check \oplus \{(u? \mapsto pw?)\}$
$UtentiLoggati' = UtentiLoggati$
$UtentiAttivi' = UtentiAttivi$
$Share' = Share$

RegistrazioneFallimento

$\Xi FileSharing$
$u?: Utenti$
$r!: Risposta$

$u? \in UtentiRegistrati$
$r! = Utente\_Già\_Registrato$

$Registrazione \mathrel{\widehat=} RegistrazioneOk \land Successo \lor RegistrazioneFallimento$

DeRegistrazioneOk

$\Delta FileSharing$
$u?: Utenti$

$u? \in UtentiRegistrati$
$UtentiRegistrati' = UtentiRegistrati \setminus \{u?\}$
$Check' = \{u?\} \lhd Check$
$UtentiLoggati' = UtentiLoggati \setminus \{u?\}$
$UtentiAttivi' = UtentiAttivi \setminus \{u?\}$
$Share' = \{u?\} \lhd Share$

DeRegistrazioneFallimento

$\Xi FileSharing$
$u?: Utenti$
$r!: Risposta$

$u? \notin UtentiRegistrati$
$r! = Utente\_Non\_Registrato$

$DeRegistrazione \mathrel{\widehat=} DeRegistrazioneOk \land Successo \lor DeRegistrazioneFallimento$

LoginUtente

$\Delta FileSharing$
$u?: Utenti$
$c?: \mathbb{P}\ Canzoni$

$UtentiLoggati' = UtentiLoggati \cup \{u?\}$
$UtentiAttivi = UtentiAttivi \cup \{u?\}$
$UtentiRegistrati' = UtentiRegistrati$
$Check' = Check$
$Share' = Share \oplus \{(u? \mapsto c?)\}$

ControllaPassword

$FileSharing$
$u?: Utenti$
$pw?: Password$

$Check\ u? = pw?$

incomplete proof of *ControllaPassword$domainCheck*
invoke

LoginNormaleOk

LoginUtente
ControllaPassword

$u? \in UtentiRegistrati$
$u? \notin UtentiLoggati$

LoginRipristinoOk

LoginUtente
ControllaPassword

$u? \in UtentiRegistrati$
$u? \in UtentiLoggati$
$u? \notin UtentiAttivi$

LoginPasswordErrata

$\Xi FileSharing$
u?: Utenti
pw?: Password
r!: Risposta

$u? \in UtentiRegistrati$
Check $u? \neq pw?$
$r! = Password\_Errata$

incomplete proof of *LoginPasswordErrata$domainCheck*
prove by reduce

LoginUtenteNonRegistrato

$\Xi FileSharing$
u?: Utenti
r!: Risposta

$u? \notin UtentiRegistrati$
$r! = Utente\_Non\_Registrato$

LoginUtenteGiàConnesso

$\Xi FileSharing$
u?: Utenti
r!: Risposta

$u? \notin UtentiAttivi$
$r! = Utente\_Già\_Connesso$

$$Login \mathrel{\widehat=} (LoginNormaleOk \lor LoginRipristinoOk) \land Successo \lor LoginPasswordErrata \lor LoginUtenteNonRegistrato \lor LoginUtenteGiàConnesso$$

LogoutOk

$\Delta FileSharing$
$u?: Utenti$

$u? \in UtentiLoggati$
$UtentiLoggati' = UtentiLoggati \setminus \{u?\}$
$UtentiAttivi = UtentiAttivi \setminus \{u?\}$
$Share' = \{u?\} \lhd Share$
$UtentiRegistrati' = UtentiRegistrati$
$Check' = Check$

LogoutNonLoggato

$\Xi FileSharing$
$u?: Utenti$
$r!: Risposta$

$u? \notin UtentiLoggati$
$r! = Utente\_Non\_Loggato$

$$Logout \mathrel{\widehat=} LogoutOk \land Successo \lor LogoutNonLoggato$$

UtenteOnLine

$FileSharing$
$u?: Utenti$

$u? \in UtentiAttivi$

UtenteOffLine

FileSharing
u?: Utenti
r!: Risposta

$u? \notin UtentiAttivi$
$u? \in UtentiLoggati$
$r! = Utente\_Off\_Line$

UtenteNonLoggato

FileSharing
u?: Utenti
r!: Risposta

$u? \notin UtentiAttivi$
$u? \notin UtentiLoggati$
$u? \in UtentiRegistrati$
$r! = Utente\_Non\_Loggato$

UtenteNonRegistrato

FileSharing
u?: Utenti
r!: Risposta

$u? \notin UtentiRegistrati$
$r! = Utente\_Non\_Registrato$

OperazioneSearch

$\Xi FileSharing$
c?: Canzoni
u?: Utenti
lista!: $\mathbb{P}$ *Utenti*

$lista! = \{ ui: Utenti \mid ui \in UtentiLoggati \land c? \in Share\ ui \land ui \neq u? \}$

$Search \mathrel{\widehat=}$
$OperazioneSearch \land UtenteOnLine \land Successo$
$\lor\ UtenteOffLine$
$\lor\ UtenteNonLoggato$
$\lor\ UtenteNonRegistrato$

DisconnessioneUtenteOk

$\Delta FileSharing$
$u?: Utenti$

$UtentiRegistrati' = UtentiRegistrati$
$UtentiLoggati' = UtentiLoggati$
$UtentiAttivi' = UtentiAttivi \setminus \{u?\}$
$Check' = Check$
$Share' = Share$

$DisconnessioneUtente \mathrel{\widehat=}$
$DisconnessioneUtenteOk \land UtenteOnLine \land Successo$
$\lor UtenteOffLine$
$\lor UtenteNonLoggato$
$\lor UtenteNonRegistrato$

AggiornaCanzoniOk

$\Delta FileSharing$
$u?: Utenti$
$c?: \mathbb{P}\ Canzoni$

$UtentiRegistrati' = UtentiRegistrati$
$UtentiLoggati' = UtentiLoggati$
$UtentiAttivi' = UtentiAttivi$
$Check' = Check$
$Share' = Share \oplus \{(u? \mapsto c?)\}$

$AggiornaCanzoni \mathrel{\widehat=}$
$AggiornaCanzoniOk \land UtenteOnLine \land Successo$
$\lor UtenteOffLine$
$\lor UtenteNonLoggato$
$\lor UtenteNonRegistrato$

DownloadOk

$\Delta FileSharing$
$u?: Utenti$
$ur?: Utenti$
$c?: Canzoni$

$c? \in Share\ ur?$
$Share' = Share \oplus \{(u? \mapsto Share\ u? \cup \{c?\})\}$

UtenteRemotoOffLine

$FileSharing$
$ur?: Utenti$
$r!: Risposta$

$ur? \notin UtentiAttivi$
$r! = Utente\_Remoto\_Off\_Line$

UtenteRemotoOnLine

$FileSharing$
$ur?: Utenti$

$ur? \in UtentiAttivi$

DownloadCanzoneNonDisponibile

$\Xi FileSharing$
$u?: Utenti$
$ur?: Utenti$
$c?: Canzoni$
$r!: Risposta$

$c? \notin Share\ ur?$
$r! = Canzone\_Non\_Disponibile$

DownloadUtenteRemotoOffLine

$\Delta FileSharing$
$u?: Utenti$
$ur?: Utenti$
$c?: Canzoni$
$r!: Risposta$

$u? \in UtentiAttivi$
$ur? \notin UtentiAttivi$
$UtentiLoggati' = UtentiLoggati \setminus \{ur?\}$
$UtentiRegistrati' = UtentiRegistrati$
$UtentiAttivi' = UtentiAttivi$
$Check' = Check$
$Share' = \{ur?\} \lhd Share$
$r! = Utente\_Remoto\_Off\_Line$

$$
\begin{aligned}
Download \;\widehat{=}\; & \\
& DownloadOk \land UtenteOnLine \land Successo \\
& \lor DownloadCanzoneNonDisponibile \land UtenteOnLine \land UtenteRemotoOnLine \\
& \lor UtenteOffLine \\
& \lor UtenteNonLoggato \\
& \lor UtenteNonRegistrato \\
& \lor DownloadUtenteRemotoOffLine
\end{aligned}
$$

ESEMPIO: AGENDA DEI COMPLEANNI

Tipi base [name, date]

Spazio di stato:

- persone “known”
  - known: $\mathbb{P}$ Name
- funzione “birthday”
  - birthday: NAME ⇸ DATE (⇸ è una funzione parziale)

PROPRIETA’ DELLO SPAZIO DI STATO

Il libro delle date deve contenere date di persone che conosciamo.

Questa proprietà è invariante.

BirthdayBook
Known: $\mathbb{P}$ Name birthday: NAME ⇸ DATE
dom(birthady) = Known

NOTA:

funzione iniettiva: 2 valori diversi nel dominio si mappano su due valori diversi del codominio

Totale Parziale

funzione suriettiva: il codominio della funzione è tutto y

Totale Parziale

AGGIUNGERE LE OPERAZIONI

Descriviamo come cambia lo stato

- AddBirthady (aggiunge nuovi elementi nel book; modifica lo stato)
- FindBirthday (a partire dal nome trovo il compleanno; non modifica lo stato)
- Remind (trova i compleanni ad una certa data; non modifica lo stato)

OPERAZIONI CHE CAMBIANO LO STATO

- X → Valore di x prima dell'operazione
- X' → Valore dopo l'operazione

AddBirthday

Δ Birthday Book	//E' una modifica
Name?: NAME	//Il nome (? = parametro di input)
Date?: DATE	//la data (? = parametro di input)
Name? $\notin$ Known	//pre condizione
birthday' = birthday $\cup$ {Name? $\rightarrowtail$ Date? }	//post condizione

Δ è un operazione che modifica le variabili di stato

? dato in ingresso (input)

$\rightarrowtail$ per descrivere le coppie di una funzione (si aggiunge a birthday la tupla [name, date])

COMMENTI

La specifica non dice cosa succede a Known

Known' = Known $\cup$ (Name?)
Lo deduco a partire da Known' = dom(birthady')

OPERAZIONI DI QUERY

Non cambiano lo stato.

Devono specificare lo schema su cui fanno le query e le precondizioni (se ci sono)

Danno il risultato della query

OPERAZIONE FIND

FindBirthday	
Ξ Birthday Book	//E' una query
Name?: NAME	//Il nome (? input)
Date!: DATE	//La data (! output)
Name? ∉ Known	//pre condizione
Date! = birthday(Name?)	//post condizione

Ξ → Definizione di una query

! → Parametro in uscita

OPERAZIONE REMIND

Remind	
Ξ Birthday Book	//E' una query
Today?: DATE	//La data (? = parametro di input)
Cards!: P Name	//i nomi (! = parametro di output)
Cards! = {n ∈ Konwn | birthday(n) = Today?}	//post condizione

Non c'è pre-condizione,
La post condizione è l'insieme dei Known che fanno il compleanno oggi.

STATO INIZIALE

InitBirthdayBook
Birthday Book
Success

INCREMENTO

Abbiamo dei casi critici

- Inserisco due volte la stessa persona
- Compleanno di una persona che non esiste

Z è un linguaggio di specifica COMPOSIZIONALE, posso scrivere i pezzi di specifica separatamente e poi comporle, posso quidi dare delle specifiche in modo incrementale.

INCREMENTO

Introduco il tipo che mi verifica l'esito di un operazione

REPORT :: OK | already_known | not_known

è un nuovo tipo enumerato

Success

result! : REPORT

---

result! = ok

ATRI INCREMENTI

AlreadyKnown

Ξ Birthday Book
Name?: NAME
result!: REPORT

Name? ∉ Known
Result! = alerady_known

NotKnown

Ξ Birthday Book
Name?: NAME
result!: REPORT

Name? ∉ Known
Result! = not_known

COMPOSIZIONE DELLO SCHEMA

Versioni robuste

RAddBirthday $\widehat{=}$ (AddBirthday $\wedge$ Success) $\vee$ AlreadyKnown

RFindBirthday $\widehat{=}$ (FindBirthday $\wedge$ Success) $\vee$ NotKnown

RRemind $\widehat{=}$ Remind $\wedge$ Success

SCHEMA RISULTANTE

RAddBirthday

Δ Birthday Book
Name?: NAME
Date?: DATE
Result!:REPORT

Name? $\notin$ Known $\wedge$ birthday' = birthday $\cup$ {Name? $\rightarrow$ Date?} $\wedge$ result! = ok) $\vee$

Name? $\in$ Known $\wedge$ birthday' = birthday $\wedge$ result! = already_known)

ESERCIZI

ESERCIZIO

Specificare in Zeta le modalit`a d'esame di un corso universitario. Il corso pu`o essere sostenuto dai soli studenti iscritti al corso ed `e composto da una prova scritta. Dopo aver sostenuto la prova scritta, lo studente ha la possibilit`a di accettare il voto e di procedere alla registrazione oppure di ripetere lo scritto; se ripete lo scritto, automaticamente perde il voto precedente. Uno studente pu`o registrare il voto soltanto se questo `e superiore o uguale a 18; dopo aver registrato il voto, lo studente non pu`o pi`u sostenere l'esame. Il superamento dello scritto e la registrazione possono avvenire in tempi di_erenti.
Modellare in Zeta:
• il corso, evidenziando:
– gli studenti iscritti;
– il voto degli studenti che hanno superato la prova scritta;
– il voto degli studenti che hanno superato l'esame.
• il superamento della prova scritta;
• la registrazione del voto.
Le operazioni (superamento della prova scritta e registrazione del voto) devono essere descritte in modo completo tenendo conto di tutti i possibili errori.

SOLUZIONE

[*Studenti*]

$Risposta ::= ok \mid studente\_non\_iscritto \mid voto\_non\_corretto \mid esame\_gia\_sostenuto \mid voto\_insufficiente \mid scritto\_non\_sostenuto$

Corso

$iscritti: \mathbb{P}\ Studenti$
$voto: Studenti \pfun \mathbb{N}$
$superato: \mathbb{P}\ Studenti$

$\mathrm{dom}\ voto \subseteq iscritti$
$superato \subseteq \mathrm{dom}\ voto$

SuperamentoScrittoOK

$\Delta Corso$
$studente?: Studenti$
$votoStudente?: \mathbb{N}$

$studente? \in iscritti$
$studente? \notin superato$
$votoStudente? \leqslant 31$
$votoStudente? \geqslant 18$
$voto' = voto \oplus \{(studente? \mapsto votoStudente?)\}$

StudenteNonIscritto

$\Xi Corso$
$studente?: Studenti$
$r!: Risposta$

$studente? \notin iscritti$
$r! = studente\_non\_iscritto$

EsameGiaSostenuto

$\Xi Corso$
$studente?: Studenti$
$r!: Risposta$

$studente? \in superato$
$r! = esame\_gia\_sostenuto$

VotoErrato

$\Xi Corso$
$votoStudente?: \mathbb{N}$
$r!: Risposta$

votoStudente? > 31
r! = voto_non_corretto

Successo

r!: Risposta

---

r! = ok

VotoInsufficiente

ΞCorso
votoStudente?: ℕ
r!: Risposta
studente?: Studenti

---

votoStudente? < 18
voto' = {studente?} ⩤ voto
r! = voto_insufficiente

SuperamentoScritto ≙
SuperamentoScrittoOK ∧ Successo
∨ StudenteNonIscritto
∨ EsameGiaSostenuto
∨ VotoInsufficiente
∨ VotoErrato

RegistrazioneOK

ΔCorso
studente?: Studenti

---

studente? ∈ dom voto
studente? ∉ superato
superato' = superato ∪ {studente?}

ScrittoNonSuperato

ΞCorso
studente?: Studenti
r!: Risposta

---

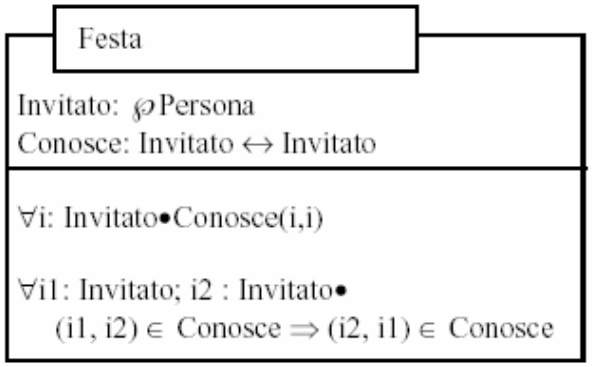
studente? ∉ dom voto
r! = scritto_non_sostenuto

Registrazione ≙
RegistrazioneOK ∧ Successo
∨ StudenteNonIscritto
∨ ScrittoNonSuperato

∨ EsameGiaSostenuto

ESERCIZIO

A una festa c'e un gruppo di invitati. Non tutti gli invitati, pero', si conoscono personalmente fra loro, anche se per ogni persona e' noto esattamente chi sono i presenti che gia' conosce. Il seguente schema Z descrive l'insiemi degli invitati alla festa e il fatto che la relazione di conoscenza é riflessiva e simmetrica.



Specificare opportunamente in Z una operazione che verifica se esiste una *cricca* di dimensione data n>0. Una cricca é un sottogruppo di almeno n invitati in cui ciascun invitato conosce personalmente ogni altro membro del sottogruppo.

ESERCIZIO

Specificare in Zeta una parte di un'applicazione che si occupa dell'autenticazione degli utenti. Gli utenti si devo registrare specificando la propria user-id e password. Una volta registrati, gli utente si connettono attraverso le proprie user-id e password all'applicazione, che verifichera' se questi dati sono effettivamente corretti. Per distinguere quali utenti sono connessi e quali no, l'applicazione tiene traccia dello stato dell'utente (connesso/sconnesso). Un utente puo' connettersi al sistema se e soltanto se esso risulta essere sconnesso.

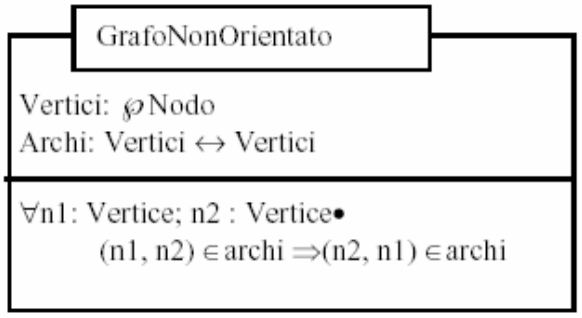
Specificare in Zeta le seguenti parti dell'applicazione:

(a) lo stato dell'applicazione (utenti registrati, utenti connessi, etc. etc.);

(b) la connessione di un utente; questa azione deve essere descritta in modo completo tenendo conto di eventuali errori (password errata, etc. etc.).

ESERCIZIO

Il seguente schema Z specifica un grafo non orientato, costituito da un insieme di nodi (i vertici del grafo) e da una relazione binaria sui vertici (gli archi). Definire opportunamente in Z una operazione che aggiunge un arco al grafo, data una coppia di nodi.



ESERCIZIO

Si consideri il seguente frammento di specifica in Z che descrive i pulsanti del ben noto sistema di ascensori:

[Button]

Button_State

floor_buttons, elevator_buttons: **P** Button
buttons: **P** Button
pushed: **P** Button

---

floor_buttons ∩ elevator_buttons = ∅
floor_buttons ∪ elevator_buttons = buttons

Init_Button

Button_State

---

pushed = ∅

Si specifichi come l'operazione Push_Button (che ha per parametro un Button) modifica Button_State.

SOLUZIONE

Push_Button

button?: Button
ΔButton_State

---

button? ∈ buttons ∧
pushed' = pushed ∪ {button?} ∧
floor_button' = floor_buttons ∧
elevator_buttons' = elevator_buttons