INGEGNERIA DEL SOFTWARE

TRIO

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

LOGICHE TEMPORALI

Descrivono l'evoluzione nel tempo di un sistema. La verità di una formula dipende dall'istante in cui la valuto.

Possono avere diverse proprietà:

- tempo finito o infinito
- descrizione del futuro e/o del passato
- tempo discreto o continuo
- logiche lineari o a diramazione (branching)

TRIO

E' una logica temporale:

- Tempo infinito
- Descrivo sia il passato che il futuro
- Tempo discreto e tempo continuo
- Logica lineare

E' una logica tipizzata, abbiamo:

- VARIABILI: ad ogni variabile è associato un tipo
- FUNZIONI: a cui è associato un dominio ed un condominio
- PREDICATI: a cui associamo un tipo agli argomenti

Quando scriviamo una specifica dobbiamo dire se è tempo discreto o continuo, ed usiamo solo **unità di tempo** e non minuti, secondi…

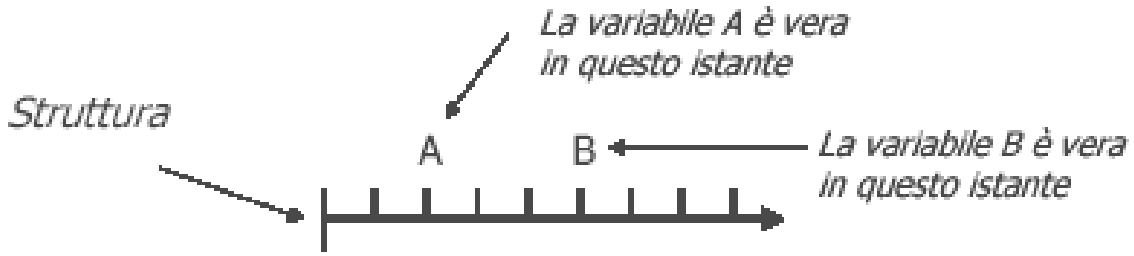
OPERATORI TEMPORALI

Futr(A,t) A è vera fra t istanti di tempo

Past(A,t) A è stata vera t instanti di tempo fa

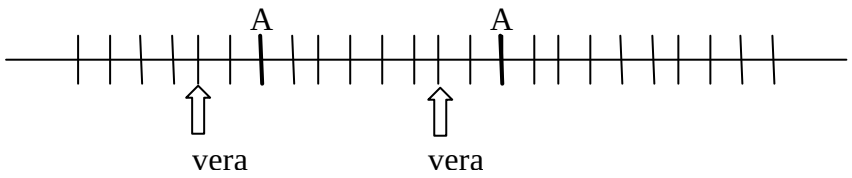
STRUTTURA DI INTERPRETAZIONE

Associamo ad ogni istante di tempo un valore ad ogni variabile, una relazione ad ogni predicato e una funzione ad ogni nome di funzione.



Esempio:

Se scriviamo F = Futr(A, 2)



FORMULE VS STRUTTURE

Temporalmente soddisfacibile

F è temporalmente soddisfacibile in S se esiste almeno un istante di tempo t in cui F è vera

Temporalmente valida

F è temporalmente valida in S se è vera in ogni istante

Valida

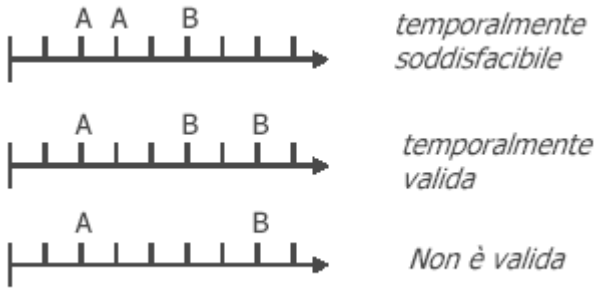
F è valida se è temporalmente valida in qualsiasi struttura rispetto alla quale può essere valuta

Modello

S è un modello per F, se F è temporalmente soddisfacibile in S

Esempio:

Futr(A,3) $\Rightarrow$ B



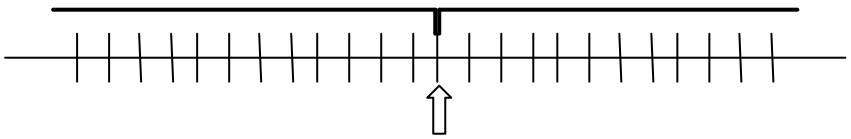
OPERATORI DERIVATI

ALWAYS

Asserisce una proprietà invariante della specifica

La proprietà deve essere sempre vera

Alw(F) = F è vera in ogni istante

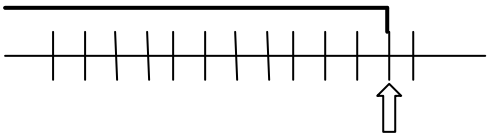


Vediamo come è definito:

$$\boxed{Alw(F) = F \wedge \forall t(t > 0 \Rightarrow (Futr(F,t) \wedge Past(F,t)))}$$

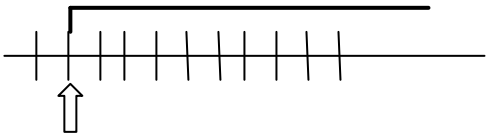
Posso estendere il concetto di Always con “Always in the past” ed “Always in the future”

AlwP(F)



$$\boxed{AlwP(F) = F \wedge \forall t(t > 0 \wedge Past(F,t))}$$

AlwF(F)



$$\boxed{Alw(F) = F \wedge \forall t(t > 0 \wedge Futr(F,t))}$$

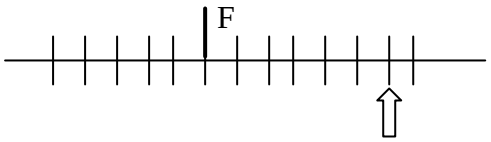
Posso scrivere:

$$\boxed{Alw(F) = F \wedge AlwP(F) \wedge AlwF(F)}$$

SOMETIMES

E' l'operatore duale di Always: prima o poi F sarà vera (passato o futuro)

Som(F)



Abbiamo anche:

SomF(F) = F sarà vera in un istante di tempo nel futuro

SomP(F) = F è stata vera in un istante t nel passato

$$SomF(F) = \exists t(t > 0 \wedge Futr(F,t))$$

$$SomP(F) = \exists t(t > 0 \wedge Past(F,t))$$

$$Som(F) = F \wedge SomF(F) \wedge SomP(F)$$

Esempio

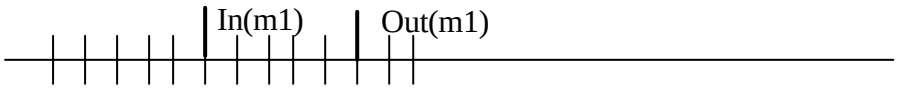
Abbiamo un canale di comunicazione con ritardo fisso
Ogni segnale in ingresso viene ritardato di 5 sec

Dobbiamo definire i **predicati** che possono essere tempodipendenti oppure no.

Ci servono due predicati:

In(m) ed Out(m) ricezione ed invio messaggio.

La struttura del sistema è la seguente:



Definisco l'unità di tempo: 1 UT = 1s

Oltre al ritardo vogliamo descrivere che il canale non perde messaggi e che il canale non genera messaggi (per semplicità imponiamo che non ci siano due messaggi uguali).

Unicità dei messaggi: $In(m) \Rightarrow AlwF(\neg In(m))$
Se ho ricevuto m, non lo riceverò più.

Perdita: tutti i messaggi prima o poi devono essere spediti.

$In(m) \Rightarrow SomF(Out(m))$

Generazione messaggi: se genero un messaggio, devo averlo ricevuto in passato:
$Out(m) \Rightarrow SomP(In(m))$

Nota: Mai usare $\Leftrightarrow$!!!

Ritardo:
$In(m) \Rightarrow Futr(Out(m),5)$

CONVENZIONI

Tutte le variabili libere sono implicitamente quantificate universalmente al livello esterno

$In(m) \Rightarrow Futr(Out(m),5)$ equivale a: $\forall m(In(m) \Rightarrow Futr(Out(m),5))$

Tutte le formule devo essere sempre vere (operatore alw sottointeso)

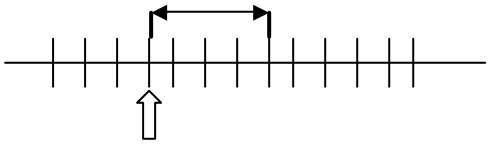
$\forall m(In(m) \Rightarrow Futr(Out(m),5))$ equivale a: $Alw(\forall m(In(m) \Rightarrow Futr(Out(m),5)))$

LASTS E LASTED

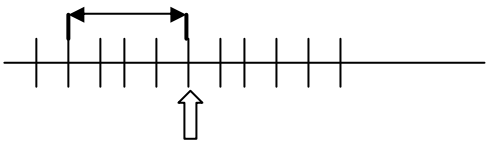
Lasts(A, t) = qualcosa durerà nel futuro

Laed(A, t) = qualcosa è durato nel passato

Lasts(A, 5)



Laed(A, 5)



Gli estremi sono esclusi.

$\boxed{Lasts(A,t) \cong \forall x(0 < x < t \Rightarrow Futr(A,x))}$

$\boxed{Lasted(A,t) \cong \forall x(0 < x < t \Rightarrow Past(A,x))}$

Possiamo includere gli estremi con due pedici:

$Lasts_{ie}$ i = incluso, e = escluso

istante corrente istante futuro

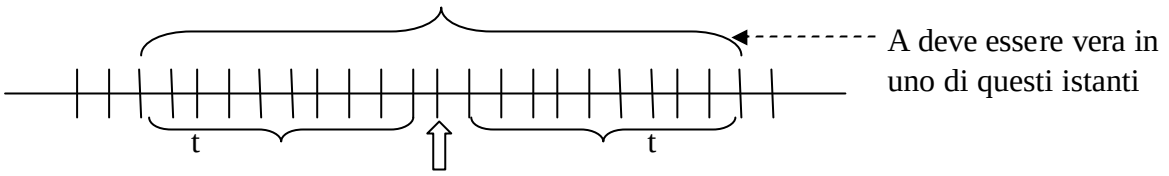
$Lasted_{ie}$ con lasted, gli estremi si invertono

istante corrente istante passato

$\boxed{Lasts(A,t)_{ie} \cong \forall x(0 \leq x < t \Rightarrow Futr(A,x))}$

WITHIN

Within(A,t) = Il predicato deve essere vero in un istante nell'intervallo futuro/passato di tempo t



WithinF(A,t) = A sarà vera nei prossimi t istanti

WithinP(A,t) = A è stata vera in t istanti passati

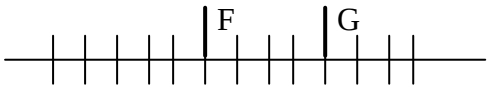
$$WithinF(A,t) \cong \exists x(0 < x < t \wedge Futr(A,x))$$

$$WithinP(A,t) \cong \exists x(0 < x < t \wedge Past(A,x))$$

$$Within(A,t) \cong \forall x(0 < x < t \Rightarrow Futr(A,x))$$

BEFORE

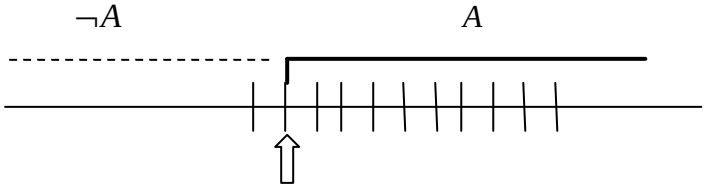
Before(F,G) = F è capitata prima di G



$$Before(F,G) \cong Som(F \wedge SomF(G))$$

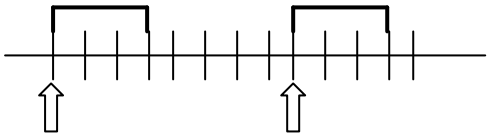
BECOME

Become(A) = Se vogliamo sapere quando A passa da falsa a vera.

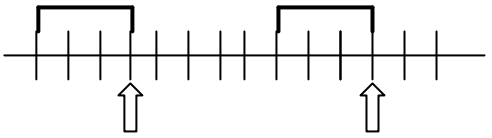


Operatore duale: **Become(** $\emptyset A$ **)**

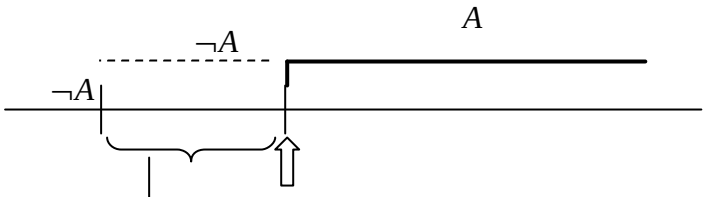
Become(A)



Become($\emptyset A$)



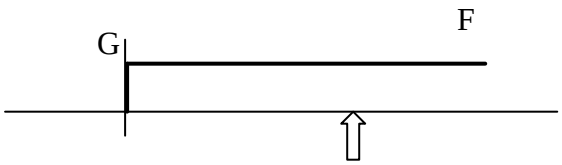
$$Become(A) \cong A \wedge \exists x(x > 0 \wedge Past(\neg A, x) \wedge Lasted(\neg A, x))$$



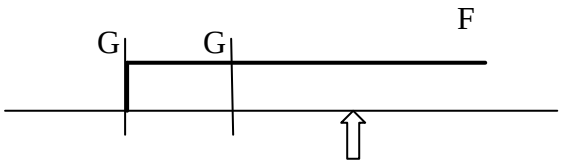
In questo intervallo deve essere $\neg A$

SINCE

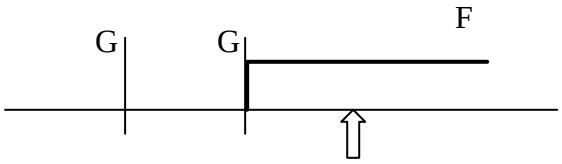
Since(F,G) = F è sempre vera dall'ultimo punto in cui è stata vera G



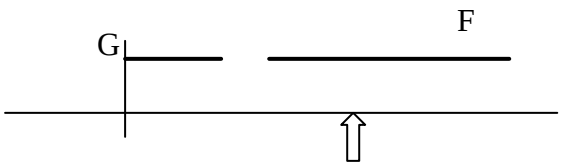
Esempi:



vera



vera

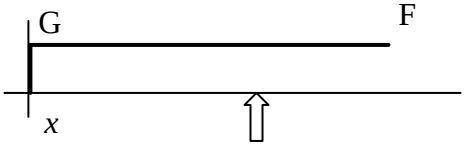


falsa

$$Since(F, G) \cong Past(G, x) \wedge Lasted(F, x)$$

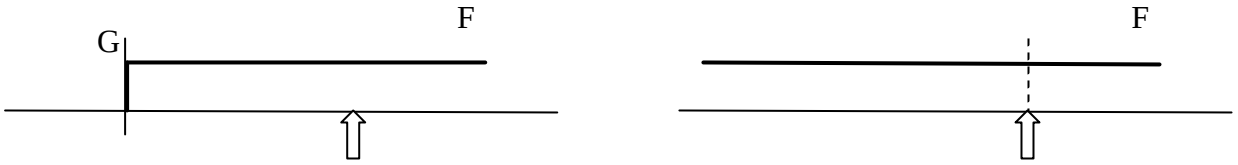
identifico l'istante di tempo in cui G è vera

da quel punto in poi F deve essere vera



Since(F,G) richiede che G sia stata vera almeno in un istante di tempo passato.

Se G è stata sempre falsa, SinceW si comporta come Since se G è stata vera oppure se G nel passato è sempre stata False, F nel passato deve essere sempre stata vera.



oppure

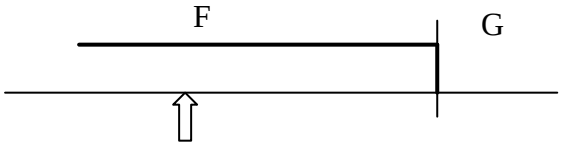
$$SinceW(F,G) \cong Since(F,G) \vee (AlwP(\neg G) \wedge AlwP(F))$$

oppure

$$SinceW(F,G) \cong Since(F,G) \vee AlwP(F)$$

UNTIL

F deve rimanere vera fino a quando capita il primo G



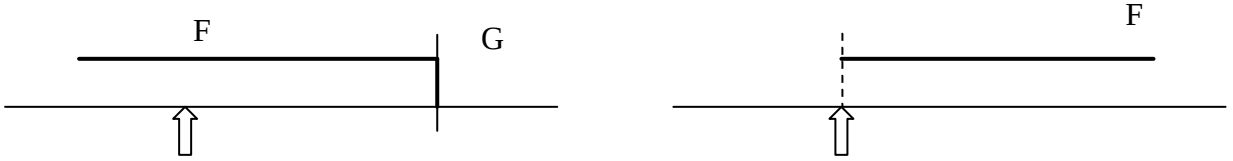
Dopo G, F può essere vera o falsa, non interessa

$$Until(F,G) \cong \exists t(t > 0 \wedge Futr(G,t) \wedge Last(F,t))$$

Con Since predichiamo sul passato, con Until sul futuro

Anche per Until c'è una forma debole:

UntilW(F,G)

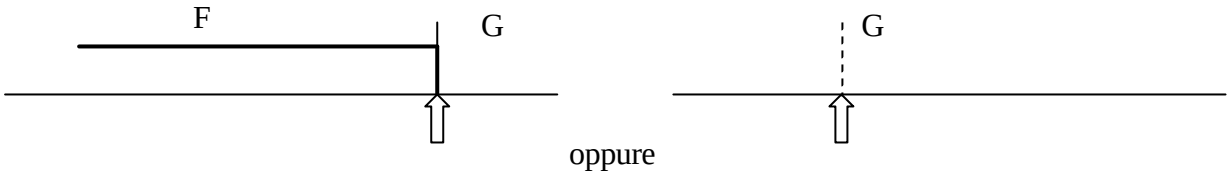


oppure

$$UntilW(F,G) \cong Until(F,G) \vee AlwF(F)$$

Altra variante:

UntilP(F,G) = se G è vera adesso allora la formula è vera.



non predichiamo sulla F

$$UntilP(F,G) \cong Until(F,G) \vee G$$

LASTTIME E NEXTTIME

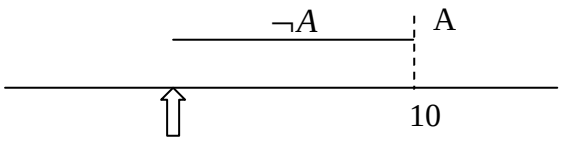
Until, Since e Become ci permettono di ragionare su ***intervalli temporali***

Per ***eventi istantanei*** un sono usati e ne vengono utilizzati altri due.

NextTime(A,t) = la prossima volta A sarà vera all'istante di tempo t e tra adesso e t sarà sempre falsa (la prossima volta in cui un evento si verificherà.

Esempio:

NextTime(A,10)



$$NextTime(A,t) \cong \exists t(t > o \wedge Futr(A,t) \wedge Lasts(\neg A,t))$$

LastTime(A,t) = l'ultima volta che un evento si è verificato

Esempio:

LastTime(A,10)

$$LastTime(A,t) \cong \exists t(t > o \wedge Past(A,t) \wedge Lasted(\neg A,t))$$

ESERCIZI E TEMI D'ESAME

CANALE DI COMUNICAZIONE

Ritardo variabile Δt

Il canale può essere acceso o spento: se è spento i messaggi ricevuti non sono più spediti, ma quelli ricevuti prima dello spegnimento sono spediti.

$\Delta t \leqslant 10s$

L'ordine dei messaggi è rispettato

predicati

Sono tempodipendenti

In(m) = quando ho ricevuto il messaggio (istantaneo)
Out(m) = quando spedisco il messaggio (istantaneo)
Acceso = il canale è acceso (identifica un intervallo)
Spento $\Leftrightarrow \neg$ Acceso

Specifica

Se ricevo un messaggio prima o poi lo spedirò:
$In(m) \wedge Acceso \Rightarrow SomF(Out(m))$

Se ricevo un messaggio ed il canale è spento, non lo devo spedire
$In(m) \wedge Spento \Rightarrow AlwF(\neg Out(m))$

Out(m) può accadere solo se in passato ho spedito un messaggio
$Out(m) \Rightarrow SomP(In(m))$

Unicità dei messaggi
$In(m) \Rightarrow AlwF(\neg In(m))$
$Out(m) \Rightarrow AlwF(\neg Out(m))$

Ritardo:
$In(m) \wedge Acceso \Rightarrow WithinF(Out(m),10)$

Ordine dei messaggi:
$Before(In(m_1) \wedge Acceso, In(m_2)) \Rightarrow Before(Out(m_1), Out(m_2))$

LAMPADA

La lampada viene gestita da un telecomando
La lampada si accende e si spegne tramite segnali inviati dal telecomando
In assenza di segnali, la lampada rimane accesa o spenta

Predicati TD
- Turn(On), Turn (Off)
- Light

Turn(On)→ Until (Light, Turn(Off))
Until richiede che prima o poi la lampada venga spenta!

Turn(Off)→ UntilW (¬Light, Turn(On))
La lampada potrebbe rimanere spenta per sempre!

¬(Turn(On)∧Turn(Off))
Il telecomando può emettere solo un segnale per volta

SinceW(¬(Turn(On),Turn(Off))→¬Light
La lampada inizialmente è spenta

SARACINESCA

Modelliamo una saracinesca che viene aperta o chiusa con un interruttore

Durante il movimento i segnali vengono ignorati

Per aprirsi o chiudersi impiega D ut

TD
- pos: posizione della sarac. (OP,CL,MV)
- com: comando (com(OP),com(CL))

Costante D: tempo di chiusura

Stato iniziale (sarac. aperta)

$$AlwP(\neg\exists m: com(m)) \rightarrow pos=OP$$

Apertura saracinesca
- Inizia subito a muoversi
- Si muove per D ut
- Dopo D ut diventa aperta
- Rimane aperta fino al prossimo comando di chiusura

$$com(OP)\wedge pos=CL \Rightarrow (Lasts(pos=MV,D)\wedge Futr(pos=OP\wedge UntilWP(pos=OP,com(CL)),D)$$

Chiusura saracinesca
- Inizia subito a muoversi
- Si muove per D ut
- Dopo D ut diventa chiusa
- Rimane chiusa fino al prossimo comando di apertura

$$com(CL)\wedge pos=OP \Rightarrow (Lasts(pos=MV,D)\wedge Futr(pos=CL\wedge UntilWP(pos=CL,com(OP)),D)$$

Descriviamo quando la saracinesca è:
- In movimento
- Aperta
- Chiusa

Saracinesca in movimento

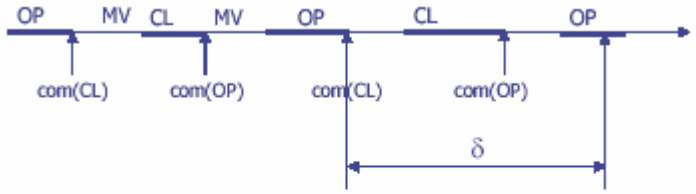
$pos=MV \Leftrightarrow WithinP(\exists m(com(m) \wedge pos \neq MV \wedge pos \neq m), D)$

Saracinesca aperta

- Non ha mai ricevuto alcun comando
- Ultimo comando valido ricevuto è stato OP e sono passati almeno D ut

$$pos=OP \Leftrightarrow (AlwP(\neg\exists m: com(m)) \vee \forall t \forall m (LastTime(com(m) \wedge pos \neq MV \wedge pos \neq m, t) \Rightarrow m=OP \wedge t \geq D))$$

Contro esempio



$t= \delta$ e $m=CL$

$LastTime(com(m) \wedge pos \neq MV \wedge pos \neq m, t)$

$m=OP \wedge t \geq D$ ← FALSA

Soluzione corretta

$$pos=OP \Leftrightarrow (AlwP(\neg\exists m: com(m)) \vee \forall t \forall m (LastTime(com(m) \wedge pos \neq MV \wedge pos \neq m, t) \Rightarrow m=OP \wedge t \geq D \wedge Past(com(OP), t)))$$

Saracinesca chiusa

$pos= CL \Leftrightarrow \neg(pos= OP \vee pos=MV)$

Descriviamo la velocità della saracinesca

- In 5 ut deve essersi mossa di 1/10 rispetto alla corsa totale

Nuova variabile reale **position:[0,1]**

ASCENSORE

Un ascensore serve N piano.

All'interno ci sono N bottoni

Ad ogni piano ci sono due bottini

Si trascurano i meccanismi di apertura e chiusura delle porte

Strategia: mantiene la stessa direzione fino a quando non occorre più prosegurie

Inverte la direzione se:

- Occorre visitare piani nella direzione opposta
- Non si deve espletare alcun servizio

Rimane in sosta nell'ultimo piano richiesto fino alla prossima richiesta

Si ferma ad un piano sse

- il piano è stato richiesto dall'interno
- Il piano è stato richiesto dall'esterno nella stessa direzione di marcia dell'ascensore
- L'ascensore è stato richiesto al piano e non ci sono altre richieste per i piani superiori

Alfabeto – Predicati DT

Floor:[1..N] way:{up,down} s:{on,off}

pushLiftButton(floor) il pulsante all'interno dell'ascensore è stato premuto

pushFloorButton(floor,way) il pulsante al piano è stato premuto

arrives(floor) l'ascensore è arrivato al piano

stops(floor) l'ascensore si è fermato al piano

leaves(floor) l'ascensore è partito dal piano

floorButton(floor,way,s) stato del pulsante al piano

liftButton(floor,s) stato del pulsante nell'ascensore

Alfabeto – Variabili TD

liftPosition:[1..N] l'ultimo piano toccato dall'ascensore

dir:{up,down,stnd} senso di marcia dell'ascensore (non coincide con il moto dell'ascensore!)

Requested Floor

Un piano è richiesto se è stato schiacciato in passato il bottone all'interno dell'ascensore e da allora l'ascensore non si è mai fermato a quel piano

$$requestedFloor(floor) \Leftrightarrow \exists t(Past(pushLiftButton(floor),t) \wedge Lasted(liftPosition \neq floor,t))$$

Called From

- L'ascensore è chiamato da un certo piano in una direzione se il bottone al piano è stato schiacciato e l'ascensore non si è mai fermato a quel piano con quella direzione

$$calledFrom(floor,way) \Leftrightarrow \exists t(Past(pushFloorButton(floor,way),t) \wedge Lasted(\neg(liftPosition=floor \wedge dir=way),t))$$

Needed Down/Up

L'ascensore è richiesto ai piani inferiori/superiori

$$needDown \Leftrightarrow \exists floor(floor<liftPosition \wedge (requestedFloor(floor) \vee \exists d(calledFrom(floor,d)))$$

$$needDown \Leftrightarrow \exists floor(floor>liftPosition \wedge (requestedFloor(floor) \vee \exists d(calledFrom(floor,d)))$$

Specifica

- L'ascensore arriva ad un piano se è sceso da un piano superiore o è salito da un piano inferiore. Il tempo per passare da un piano all'altro è Δtb

$$leaves(floor) \wedge dir=up \Rightarrow Futr(arrives(floor+1) \wedge dir=up, \Delta tb) \wedge Lasts(dir=up, \Delta tb)$$

$$leaves(floor) \wedge dir=down \Rightarrow Futr(arrives(floor-1) \wedge dir=down, \Delta tb) \wedge Lasts(dir=down, \Delta tb)$$

- L'ascensore si ferma al piano sse:
  - Il piano è stato prenotato dall'interno
  - E' stato chiamato a quel piano nella direzione in cui si sta muovendo
  - L'ascensore non deve proseguire ai piani superiori

$$\begin{aligned}&\text{stops(floor)} \Rightarrow \\ &\quad \text{arrives(fllor)} \wedge \\ &\quad\quad (\text{requested(floor)} \vee \\ &\quad\quad \exists \text{way}(\text{calledFrom(floor,way)} \wedge \text{way=dir}) \vee \\ &\quad\quad (\text{dir=down} \wedge \neg \text{neededDown}) \vee \\ &\quad\quad (\text{dir=up} \wedge \neg \text{neededUp}))\end{aligned}$$

- Se l'ascensore di ferma al piano, vi rimane almeno per Δts e durante questo intervallo non cambia la sua direzione. La scelta della direzione futura verrà presa dopo Δts da quando si è fermato

$$\begin{aligned}&\text{stops(floor)} \wedge \text{dir=up} \Rightarrow \\ &\quad \text{Lasts(liftPosition=floor} \wedge \text{dir=up}, \Delta\text{ts)}\end{aligned}$$

$$\begin{aligned}&\text{stops(floor)} \wedge \text{dir=down} \Rightarrow \\ &\quad \text{Lasts(liftPosition=floor} \wedge \text{dir=down}, \Delta\text{ts)}\end{aligned}$$

- Dopo essersi fermato per Δts, l'ascensore rimane fermo se non è richiesto a nessun piano e cambia la sua direzione in **stnd**. Rimarrà a quel piano fino alla prossima chiamata

$$\begin{aligned}&\text{Becomes(dir=stnd)} \wedge \\ &\text{UntilW(dir=stnd} \wedge \text{liftPosition=floor, neededUp} \vee \text{neededDown)} \\ &\Rightarrow \\ &\text{Past(stops(floor, } \Delta\text{ts)} \wedge \neg(\text{neededUp} \vee \text{neededDown)}\end{aligned}$$

- L'ascensore si ferma al piano per Δts con delle chiamate pendenti.
  L'ascensore partirà verso il basso sse:
  - La sua direzione è verso il basso e ci sono chiamate pendenti verso i piani inferiori
  - Ci sono chiamate pendenti solo verso il basso

.... simile per la direzione "alto"

Leaves(floor) ⇒
Past(stops(floor) ∧ dir=down, Δts) ∧ neeedDown ∧ dir=down
∨
Past(stops(floor) ∧ dir=up, Δts) ∧ neeedUp ∧ dir=up
∨
Past(stops(floor) ∧ dir=down, Δts) ∧ ¬neeedUp ∧ neededDown ∧
dir=down
∨
Past(stops(floor) ∧ dir=down, , Δts) ∧ neeedUp ∧ ¬ neededDown ∧ dir=up
∨
Since(¬(neededUp∨neededDown),Becomes(dir=stnd)) ∧
(
(neededDown ∧ ¬ neededUp ∧ dir=down) ∨
(¬ neededDown ∧ neededUp ∧ dir=up) ∨
(neededDown ∧ neededUp)
)

- Accensione dei bottoni

liftButton(floor,on) ⇔ requestedFloor(floor)
∧
∀d(floorButton(floor,d,on) ⇔ calledFrom(floor,d))
∧
Alw(¬pushFloorButton(1,down) ∧ ¬pushFloorButton(N,up)

Proprietà del sistema

- Qualsiasi richiesta viene prima o poi soddisfatta (liveness)

pushLiftButton(floor) ∧liftPosition≠floor ⇒
SomF(stops(floor))

Vincoli quantitativi

- Un ascensore soddisfa una chiamata o richiesta entro un tempo limite (inferiore a 2*N*(Δtb +Δts))

∀floor: Alw(pushLiftButton(floor) ∧ liftPosition≠floor⇒
∃t(t<2*N*(Δtb +Δts) ∧ Futr(stops(floor),t)))

DISTRIBUTORE

Modellare in Trio un distributore di bevande. Una volta inserito il gettone, la bevanda viene preparata in 10 secondi; i gettoni non possono essere inseriti mentre la macchina sta preparando la bevanda. Modellare in Trio i seguenti aspetti del sistema:

i gettoni possono essere inseriti solo quando il distributore non e in funzione;

una volta inserito il gettone, prepara la bevanda in 10 secondi

la bevanda viene prodotta soltanto se e stato inserito il gettone;

quando il distributore e in funzione

Gettone = viene inserito il gettone nel distributore (istante)
InFunzione = la macchina sta preparando la bevanda
Bevanda = la bevanda è pronta

$$InFunzione \Rightarrow \neg Gettone$$

$$Gettone \wedge \neg InFunzione \Rightarrow Becomes(InFunzione) \wedge Futr(Becomes(\neg InFunzione) \wedge Bevanda, 10)$$

$$Bevanda \Rightarrow Past(Gettone, 10)$$

$$InFunzione \Rightarrow \exists t | LastTime(Gettone, t) \wedge Lasted(\neg Bevanda, t)$$

PERSONE

Dati questi predicati:

- `nasce(p)` = nasce la persona p (è vero solo in un istante di tempo)
- `sposa(p1,p2)` = la persona p1 sposa la persona p2 (è vero solo in un istante di tempo). Se è vero `sposa(p1,p2)` allora è vero anche `sposa(p2,p1)`
- `divorzia(p1,p2)` = la persona p1 divorzia dalla persona p2 (è vero solo in un istante di tempo)
- `figlioDi(f,p)` = `f` è figlio di p

specificare in Trio le seguenti proprieta':

(a) Una persona può nascere solo una volta

(b) Una persona si può sposare solo dopo essere nata

(c) Il figlio deve essere nato dopo i suoi genitori

(d) Una persona può divorziare se e' stata sposata e non ha ancora divorziato

(e) *Facoltativo* Una persona si può sposare se non e' mai stata sposata o se ha gia' divorziato dall'ultima volta che si e' sposata.

(a) Una persona può nascere solo una volta

$$nasce(p) \Rightarrow AlwP(\neg nasce(p)) \wedge AlwF(\neg nasce(x)) \quad (1)$$

(b) Una persona si può sposare solo dopo essere nata

$$sposa(x,y) \Rightarrow SomP(nasce(x)) \wedge SomP(nasce(y)) \quad (2)$$

(c) Il figlio deve essere nato dopo i suoi genitori

$$figlioDi(x,y) \Rightarrow \exists t(t > 0 \wedge Past(nasce(x),t) \wedge \exists k(k > 0 \wedge k < t \wedge Past(nasce(x),k)) \quad (3)$$

(d) Una persona può divorziare se e' stata sposata e non ha ancora divorziato

$$divorzia(x,y) \Rightarrow \exists t(t > 0 \wedge Past(sposa(x,y),t) \wedge Lasted(\neg divorzia(x,y),t)) \quad (4)$$

(e) Una persona si può sposare se non e' mai stata sposata o se ha gia' divorziato dall'ultima volta che si e' sposata.

$$sposa(x,y) \Rightarrow AlwP(\neg sposa(x,y)) \vee \exists(t > 0 \wedge \exists z(Past(divorzia(x,z),t)) \wedge \forall w(Lasted(\neg sposa(x,w),t)) \quad (5)$$

CONTROLLORE

Formalizzare in TRIO le seguenti specifiche informali:

i. Tutte le volte che l'impianto invia al controllore il segnale X, il controllore deve rispondere inviando all'impianto il comando Y entro un tempo compreso fra T1 e T2 dall'invio di X, a meno che nel frattempo l'impianto non emetta il segnale di annullamento Z.

ii. Tutte le volte che il controllore invia all'impianto il segnale A, se questo si trova nello stato B entra immediatamente nello stato C, nel quale resta per un tempo T, per poi ritornare nello stato B. Il ritorno nello stato B può anche avvenire prima che trascorra T se il controllore invia all'impianto il segnale D di annullamento.

Soluzioni

$$Alw(X \rightarrow (\exists t\ (t1 <= t\ <= t2 \wedge Futr(Y, t) \wedge \neg\ \exists\ t'\ (t'<t \wedge Futr(Z, t'))))$$
$$\vee$$
$$(\exists t\ (t<t2 \wedge Futr(Z, t) \wedge \neg\ \exists t'\ (t1<=t'<=t\ Futr(Y, t)))))$$

$$Alw(B \wedge A \rightarrow (Lasts_{ie}(C \wedge \neg D\ ,\ T) \wedge Futr(B \wedge \neg C, T)$$
$$\vee$$
$$(\exists t\ (t<T \wedge Futr(D, t) \wedge Futr(B, t) \wedge Lasts_{ie}(C, t))$$