Formal methods for concurrent and real-time systems (teoria)

fonte immagine: sconosciuto

Sommario

[1 I sistemi a transizione di stati 3](#_Toc191891496)

[1.1 Il concetto di *non determinismo* nei sistemi a transizione di stati 6](#_Toc191891497)

[1.2 Varianti dei sistemi a transizione di stati 6](#_Toc191891498)

[1.2.1 Input negli stati 6](#_Toc191891499)

[1.2.2 I program graph 7](#_Toc191891500)

[1.3 A modellare la concorrenza 8](#_Toc191891501)

1. I sistemi a transizione di stati

Per modellizzare un sistema concorrente, volendo, è possibile utilizzare anche un automa a stati finiti (o anche altri formalismi come le reti di Petri), ma ci è più comodo, e più esaustivo, utilizzare i sistemi a transizione di stati.

Si definisce *sistema a transizione di stati* (in inglese: *transition system*) la tupla di:

: un insieme di *stati*;

: gli stati iniziali del sistema;

: un insieme di *azioni* (i “simboli di input” del sistema);

: una *relazione di transizione* (data una coppia “stato – azione”, indica in quale stato andare se si riceve quell’azione in quello stato);

: un insieme di *atomi* (variabili booleane in corrispondenza agli stati);

: una *funzione di etichettatura* (assegna agli atomi un grado di verità per ciascuno stato).

Osservazione:

* notiamo che questa definizione ricorda in parte quella dell’automa a stati finiti, le differenze però sono importanti, in quanto nei sistemi a transizione di stati:
  + abbiamo (e, conseguentemente, ). Potremmo, volendo, modellizzare anche in un FSA creando uno stato per ogni possibile “combinazione” dei valori di verità per ciascuno stato, il che può essere molto oneroso!
  + non vi sono stati finali. Questa ci fa intuire come i sistemi a transizione di stati siano sistemi “non terminanti”. In ogni caso, se vogliamo modellare il concetto di “stop” del sistema, possiamo utilizzare uno stato “pozzo” (stato senza transizioni in uscita);
  + tutti gli insiemi enunciati nella definizione (, e ) possono essere infiniti (anche infiniti non numerabili!).

Vediamo qualche esempio di sistemi a transizione di stati, partiamo da un esempio molto semplice:

Immagine che contiene diagramma, testo

Il contenuto generato dall'IA potrebbe non essere corretto.

Questo sistema modelizza un bistabile SR, in particolare:

* , dove è iniziale;

Si comincia da , dove la flag è . Se viene inserito si rimane in , mentre se viene inserito si va , dove è .

Anche se si tratta di un sistema molto semplice possiamo fare qualche osservazione:

* il modello non ci indica cosa accade se ed sono inseriti in contemporanea, questo perché il modello non lo prevede! Il modello prevede che si possa riceve (come negli FSA) un input alla volta;
* in genere, come in questo caso, alle azioni corrispondono i “comandi” inseribili nel sistema (“set” e “reset” in questo caso).

Immagine che contiene diagramma, testo

Il contenuto generato dall'IA potrebbe non essere corretto.Vediamo un esempio leggermente più complesso:

Questo modellizza un distributore automatico di birra e soda. Per prima cosa si inserisce il denaro, dopodiché si seleziona “soda” o “birra” [NON CAPITO IL FATTO DI ].

Rispetto all’esempio precedente:

* viene convenzionalmente utilizzato per rappresentare una transizione di stato che non interessa l’ambiente esterno (in altre parole, il sistema cambia il proprio stato a seguito di una computazione interna) [NON CAPITA QUALE SIA IN QUESTO CASO LA COMPUTAZIONE INTERNA];
* il sistema è non deterministico, in quanto da è possibile andare sia in che in con lo stesso “input” (che in questo caso sarebbe ). Notiamo però che questa definizione di “non determinismo” differisce da quella enunciata per gli FSA. Un FSA riconoscitore deterministico, infatti, percorre tutti i percorsi possibili per un certo input, accettando se almeno uno di questi percorsi porta ad uno stato di accettazione, mentre per i sistemi a transizione di stati non è così! A differenza di un FSA, infatti, il sistema in figura, non procede su entrambi in rami da , ma o in uno o nell’altro in base al risultato della computazione interna (indicata con ).

Un ulteriore esempio che potremmo fare è quello di un circuito RCL, infatti, l’evoluzione di un circuito RCL è modelizzabile per mezzo di un sistema a transizione di stati! Chiaramente si tratta di una scelta assai scomoda (nel caso di un circuito RCL, l’insieme e avrebbero cardinalità del continuo) rispetto all’utilizzo delle equazioni caratteristiche dei componenti, ma solo per fare intendere che si tratta di un modello estremamente generale per rappresentare l’evoluzione temporale dei sistemi.

* 1. Il concetto di *non determinismo* nei sistemi a transizione di stati

Come accennato nell’esempio del distributore automatico, per i sistemi a transizione di stati “non determinismo” ha un’accezione differente rispetto a quella che ha per gli FSA. Più formalmente:

un sistema a transizione di stati è detto *action-deterministico* se la sua relazione mette in relazione qualunque coppia “stato – input” con al più uno stato.

Inoltre, un sistema a transizione di stati è detto *state-label-deterministico* se per ogni stato esiste al più uno stato in cui poter andare da esso e se esiste ci si può andare con un unico input [CHIEDERE SE E’ “AL PIU’]

Osservazione:

* lo state-label-determinismo è dunque un caso particolare del’action-determinismo.
  1. Varianti dei sistemi a transizione di stati

Vediamo ora alcune varianti dei sistemi a transizione di stati, ovvero riscritture/formalismi equivalenti, che torneranno utili più avanti.

* + 1. Input negli stati

Immagine che contiene diagramma, schizzo, disegno, cerchio

Il contenuto generato dall'IA potrebbe non essere corretto.Qualunque sistema a transizione di stati può essere riscritto in modo da eliminare tutte le azioni al di fuori di , vediamolo sull’esempio del bistabile :

Ad esempio, dallo stato possiamo abbiamo due transizioni: con in e con rimaniamo in , possiamo dunque creare due nuovi atomi ed e “sdoppiare” lo stato in due stati:

* : stato in cui è vera;
* : stato in cui è vera.

In ciascuno di questi stati, il comportamento sarà analogo a quello che si avrebbe in se si ricevesse oppure .

* + 1. I program graph

Immagine che contiene diagramma, schizzo, linea, disegno

Il contenuto generato dall'IA potrebbe non essere corretto.Un formalismo del tutto equivalente ai sistemi a transizione di stati ma spesso più espressivo di questi sono i *program graph*:

In questo esempio vediamo (sopra) un sistema a transizione di stati che ha il compito di “contare” fino a in particolare, attende dopodiché si conta fino a e aspetta . Se riceve in conteggio ricomincia.

Anziché rappresentare il “conteggio” con transizioni spontanee è indubbiamente più leggibile e comodo (in quanto molto più facilmente adattabile al caso in cui bisogna conteggiare qualsiasi altro numero) utilizzabile una variabile.

Il concetto di “variabile” non è contemplato nella definizione che abbiamo dato per i sistemi a transizione di stati, si può dimostrare però non portare maggior potere espressivo. Inoltre, nei program graph non si parla di “stati” ma di “posizioni” (“locations” in inglese) in quanto esse non riportano il valore attuale di tutte le variabili del program graph, ma solo le possibili transizioni (“stato”, in informatica, significa “posizione + configurazione attuale”).

Le etichette delle transizioni sono dunque leggermente più complesse nei program graph:

dove:

* : è la lista di “assegnamenti” da effettuare sulle variabili (es. : assegna a );
* : sono le azioni che triggerano la transizione (come per i sistemi a transizione di stati);
* : sono la lista di condizioni (dette “guard”) sulle variabili che devono essere soddisfatte affinché la transizione sia triggerabile dalle azioni (es. : la transizione è effettuata solo in un momento in cui e si riceve ).
  1. A modellare la concorrenza

Siano:

* e due generici sistemi a transizioni di stati:

si definisce interleaving di e (in simboli: ) il sistema a transizione di stati:

dove:

- è la combinazione di tutte le transizioni [VENIVA MOLTO SCOMODA DA SCRIVERE BENE]

-

Immagine che contiene testo, diagramma, schermata, Piano

Il contenuto generato dall'IA potrebbe non essere corretto.Immagine che contiene testo, diagramma, schermata, Piano

Il contenuto generato dall'IA potrebbe non essere corretto.Può sembrare poco intuitivo nella definizione, ma in realtà si tratta di un concetto molto semplice, del tutto analogo a quello di due processi che agiscono in modo concorrente, ad esempio:

Abbiamo due sistemi indipendenti che rappresentano due semafori stradali (per semplicità senza il “giallo”). L’interleaving tra questi due è un sistema atto a rappresentare qualunque possibile esecuzione concorrente, ovvero qualunque esecuzione indipendente dei due dove a ogni “step” solo uno “procede”, ovvero “cambiano colore” uno alla volta (l’esecuzione è concorrente, non parallela!).

L’interleaving si può chiaramente fare anche tra due program graph, l’unica accortezza è il caso in cui i program graph iniziali utilizzano variabili con lo stesso nome: va modellizzato l’accesso concorrente dei due program graph iniziali a queste variabili. L’accesso concorrente può essere gestito in più modi:

* Immagine che contiene diagramma, Piano, testo, Disegno tecnico

  Il contenuto generato dall'IA potrebbe non essere corretto.atomicità: non vi può essere parallelismo per operazioni da program graph diversi sulla stessa variabile:
* Immagine che contiene diagramma, linea, Carattere, testo

  Il contenuto generato dall'IA potrebbe non essere corretto.attraverso variabili condivise:

Qui vediamo un esempio molto semplice con un “producer” e un “consumer” : il “producer” riempie il buffer fino a quando non raggiunge la dimensione massima consentita (10) e il “consumer” preleva dal buffer se questo non è vuoto;

* Immagine che contiene testo, diagramma, Carattere, linea

  Il contenuto generato dall'IA potrebbe non essere corretto.attraverso semafori:

Nell’assunzione che le operazioni di controllo e decremento del semaforo siano atomiche chiaramente!

* + 1. L’handshaking

Per modellare il fatto che due (o più) sistemi a transizione di stati si “sincronizzino” su un insieme di azioni, utilizziamo l’operatore *handshaking*.

Siano:

* , due generici sistemi a transizione di stati (aventi insiemi di azioni, rispettivamente, e );

si definisce “handshaking di e sull’insieme ” (in simboli, ) il sistema a transizione di stati che rappresenta l’evoluzione di e sincronizzati su . [OMESSA DEFINIZIONE FORMALE]

Immagine che contiene diagramma, linea, cerchio, disegno

Il contenuto generato dall'IA potrebbe non essere corretto.In buona sostanza, si comporta come l’interleaving tranne nel caso in cui un’action è compresa in , in quel caso il sistema deve modellizzare il fatto che i due sistemi evolvano in concomitanza su di essa, ad esempio:

Come vediamo, è identico all’interleaving tranne che sulle transizioni che coinvolgono , in cui i due sistemi devono procedere all’unisono.

* + 1. Sistemi a transizione di stati e canali di comunicazione

Immagine che contiene diagramma, testo, schizzo, Piano

Il contenuto generato dall'IA potrebbe non essere corretto.Notiamo che l’handshaking può essere utilizzato per modellare la comunicazione tra sistemi, ad esempio:

Immagine che contiene diagramma, schizzo, disegno, cerchio

Il contenuto generato dall'IA potrebbe non essere corretto.Il sistema complessivo è modellato da : quando il treno è prossimo al cancello un controller “comanda” al cancello di abbassarsi e quando il treno si allontana il controller fa rialzare il cancello. Dunque, con modelliamo la sincronizzazione tra treno e cancello ottenuta, in questo caso, grazie al controller tuttavia, il formalismo non specifica “l’ordine dei messaggi”. In altre parole, dall’esperienza comune, sappiamo che è il controller a “ricevere il messaggio ” dal treno (nel senso che il controller “vede” il treno avvicinarsi), ma il nostro modello non lo specifica (potrebbe anche essere il controller a comunicare al treno di “venire avanti”). Questo concetto di “direzione” può essere modellato attraverso i *canali*, ad esempio:

Qui vediamo modellati tre sistemi:

* un “**buffer** di capacità ;
* un “**producer**” che invia dati a un buffer;
* un “**consumer**” che consuma i dati presenti nel buffer.

In questo esempio abbiamo due canali di comunicazione:

* : canale che utilizza il producer per comunicare al buffer che vuole inviargli dati;
* : canale che utilizza il consumer per comunicare al buffer che vuole consumargli dati.

La notazione è piuttosto intuitiva, restando sull’esempio:

* : invia un messaggio sul canale ;
* : blocca il sistema (il buffer) fino a quando un messaggio non viene inviato sul canale .

Più in generale, è possibile:

* inviare un messaggio sul canale con la notazione ;
* ricevere un messaggio sul canale con la notazione .

I canali sono chiaramente gestiti in FIFO e possono avere una capacità finita (quando viene raggiunta, blocca il processo fino a quando il canale rimane pieno).

[LASCIATO PERDERE ESEMPIO DI NANO PROMELA]

* 1. Proprietà dei sistemi a transizione di stati
     1. *Linear-Time properties*

Sia:

* , un generico set di atomic proposition;

una *proprieta LT* (“Linear-Time”) su è un sottoinsieme di .

In buona sostanza, una proprietà LT è un insieme di possibili trace per un sistema a transizione di stati (il sistema “soddisfa” la proprietà , in simboli , se ogni sua possibile trace è inclusa nella proprietà).

[RICORDARSI CHE NON HO MESSO NEL DOCUMENTO LA DEFINIZIONE DI TRACE]

* + - 1. Definizione di *invariante*

Sia:

* , un generico set di atomic proposition;

una proprietà su è un *invariante* se esiste almeno una formula in logica proposizionale su vera in qualunque elemento di .

Immagine che contiene testo, diagramma, Carattere, linea

Il contenuto generato dall'IA potrebbe non essere corretto.Si tratta di un concetto molto semplice, consideriamo ad esempio l’interleaving tra i due sistemi:

Alcune possibili proprietà LT:

* “i due semafori non possono mai avere lo stesso colore” : è un inviariante? Sì! Possiamo enunciare la formula ;
* “quando diventa verde, non può più tornare rosso” : è un inviariante? No! Non ci è possibile esprimerla con una formula proposizionale!
  + - 1. Definizione di *safety-property*

Sia:

* , un generico set di atomic proposition;

una proprietà su è una *safety-property* se ciascuna trace non appartenente ad essa è costituita da un prefisso, di lunghezza finita, che la viola.

Osservazioni:

* è piuttosto intuitivo notare che tutti gli invarianti sono safety;
* “quando diventa verde, non può più tornare rosso” è safety (tutte le trace in cui non è rispettata hanno un “prefisso” più o meno lungo, ma finito, che la viola).

In buona sostanza, sono tutte le proprietà la cui violazione può essere verificata sui primi elementi (per un certo ).

Immagine che contiene schizzo, diagramma, bianco, linea

Il contenuto generato dall'IA potrebbe non essere corretto.Spesso si utilizzano degli FSA riconoscitori per esprimere le safety-properties ad esempio, per la proprietà “nel semaforo, il rosso deve essere sempre preceduto dal giallo” (l’FSA riconosce i “prefissi” che violano la proprietà):

* + - 1. Definizione di *liveness-property*

Sia:

* , un generico set di atomic proposition;

una proprietà su è una *liveness-property* se ciascuna trace non appartenente ad essa NON è costituita da un prefisso, di lunghezza finita, che la viola.

Dunque, una liveness-propery può essere violata sono “all’infinito”. Ad esempio, la starvation-freedom (“prima o poi ciascun entità del sistema che richiede la risorsa critica, vi accederà”) è chiaramente una liveness-property in quanto l’analisi “del prefisso” (ovvero dei primi elementi per un certo ) non dà alcun elemento per capire se la trace è parte o meno della proprietà (dopo , ciascuna entità potrebbe accedere alla risorsa critica).

* + - 1. Teorema di decomposizione

Sia:

* , un generico set di atomic proposition;

qualunque proprietà LT su può essere scomposta in due proprietà su :

dove:

- è una certa safety-property;

- è una certa liveness-property.

Osservazioni:

* questo teorema fondamentalmente dice che qualunque proprietà può essere verificata controllando “i prefissi” (verifica di ) e verificandola all’infinito (verifica di ), non dice che una proprietà è o safety o liveness (che, chiaramente, non è vero)!
* l’unica proprietà che è sia safety che liveness è quella universale (quella che accetta qualunque trace).
  + - 1. Fairness

Immagine che contiene diagramma, linea, Piano, bianco

Il contenuto generato dall'IA potrebbe non essere corretto.Consideria il seguente interleaving:

toRed2

toGreen2

toRed1

toGreen1

In generale, l’interleaving modella l’evoluzione concorrente di due sistemi indipendenti, incluendo così sistuazioni possibilme irrealistiche. Ad esempio, le seguenti trace:

vogliono ambedue modellare il fatto che i due semafori invertono il proprio colore ma in “i” è fa variare il proprio colore “prima” di mentre in “ii” è l’opposto. Nella pratica, però, i semafori hanno velocità analoghe e dunque entrambe queste trace modellano un comportamento “irrealistico” (il fatto che uno dei due semafori è “più veloce” dell’altro). I fairness constraints sono ciò che aggiungiamo per escludere dal modello questi comportamenti “irrealistici”. Dal punto di vista formale, infatti, escludere questi comportamenti può fare sì che proprietà “a priori” non soddisfatte vengano soddisfatte se si escludono i comportamenti “irrealistici”. Ad esempio, la proprietà “entrambi i semafori sono verdi infinitamente spesso” (ovvero: “non accadrà mai che entrambi i semafori sono rossi”) è violata dalla trace “i” ma è perfettamente ragionevole se consideriamo un sistema di due semafori concorrenti (avrei voluto scrivere indipendenti, non l’ho messo perché “indipendenti” non è proprio quello che sono due semafori che non sono mai entrambi rossi,cioè due semafori entrambi rossi possono comunque essere indipendenti).

Più formalmente, dato un insieme di azioni, una *fairness assumption* è una tripla di tre *fairness constraint*, ciascuno dei quali è un insieme di insiemi di azioni.

Una generica run soddisfa una certa proprietà sotto una certa fairness assuption se essa rispetta tutti e tre i suoi constraint, ovvero:

1. l’*uncoditionally constraint* : và rispettato indefinitamente;
2. lo *strong constraint* : ;
3. il *weak constraint* : .

La proprietà “entrambi i semafori sono verdi infinitamente spesso” diventa soddisfatta dal sistema sotto (in simboli: “”).

Notiamo che i fairness constraints sono utili solo solo per le liveness properties (si può anche dimostrare formalmente che per le safety properties sono del tutto irrilevanti).

COSA MANCA:

* comprensione di def. Formale di strong e weak constraint; (ricordarsi anche )
* comprensione di esempio a slide 32
* slide 35

1. Model checking di sistemi a transizione di stati finiti su proprietà LT
   1. Model checking di safety properties

Avevamo visto che le safety properties possono essere specificate anche come FSA: gli FSA riconoscitori dei “bad prefix” che definiscono le safety properties. Per la verità, è possibile definire safety properties il cui riconoscitore dei “bad prefix” è una macchina di Turing. Per semplicità, ci limiteremo a safety properties *regolari*, ovvero il cui riconoscitore dei “bad prefix” è un FSA (dunque, prossimamente, non faremo mai distinzioni tra proprietà regolari e non). Per queste, si può dimostrare che, sia:

* , una generica safety property;
* , l’FSA riconoscitore dei bad prefix di ;
* , un generico sistema a transizione di stati finito;

dove:

- : è l’automa prodotto costruibile tra e ;

- è l’invariante definito come la proprietà che non finisca nello stato finale.

Immagine che contiene testo, diagramma, linea, Piano

Il contenuto generato dall'IA potrebbe non essere corretto.

Vediamo un esempio:

Qui abbiamo un che rappresenta un semaforo tedesco e l’FSA che rappresenta la proprietà “prima di ci deve essere ”, mentre in basso a destra vediamo l’automa del prodotto. La costruzione dell’automa del prodotto è una versione semplificata di quella dell’automa del prodotto tra due FSA:

* l’insieme di stati dell’automa è un sottoinsieme del prodotto cartesiano tra gli stati di TS ed FSA;
* lo stato iniziale è lo stato “composto” dagli stati iniziali di TS ed FSA (in questo caso, dunque, );
* partendo dallo stato iniziale, ci disegnano le transizioni “simultanee” di TS ed FSA. Ad esempio:
  + da , il TS può andare solo in ;
  + da , l’FSA può andare solo in ;

dunque, porta nello stato e così via.

L’automa del prodotto non presenta stati che contengono uno stato finale dell’FSA, dunque il TS soddisfa la proprietà richiesta.

Nei model checker, se un sistema non rispetta una certa proprietà è apprezzabile che, oltre ad accorgersene, restituiscano anche l’esecuzione che ha violato la proprietà.

* 1. Model checking di liveness properties

Immagine che contiene schizzo, disegno, diagramma, cerchio

Il contenuto generato dall'IA potrebbe non essere corretto.Anche per le liveness properties possiamo utilizzare un meccanismo analogo a quello visto per le safety properties, ma non possiamo usare gli FSA, in quanto le liveness properties sono definite “all’infinito” e gli FSA accettano stringhe di lunghezza finita. Usiamo gli *automi di Büchi*:

Formalmente, gli automi di Büchi sono definiti esattamente come gli FSA, cambia solo il fatto che una stringa è accettata dall’automa di Büchi se e solo se l’automa “passerà” infinite volte sugli stati finali (non so se è almeno uno stato finale oppure ciascuno stato finale). Per l’automa in figura, possiamo scrivere la seguente espressione -regolare, che rappresenta tutte e sole le stringhe accettate dall’automa:

* [l’elevazione a significa “ripetizione per infinite volte]

ovvero tutte e sole le stringhe che “passano” per infinite volte.

Non ci addentriamo nella teoria, non banale, dei linguaggi (ovvero dei linguaggi di stringhe di infinita lunghezza), solo due osservazioni ci interessano:

* gli automi di Büchi sono una generalizzazione degli FSA, è sufficiente, infatti porre un autoanello sullo stato finale etichettato con (alfabeto delle stringhe riconosciute dall’automa);
* differentemente dagli FSA, negli automi di Büchi il non determinismo introduce potere espressivo. Esistono, infatti, linguaggi che possono essere riconosciuti solo da automi di Büchi non deterministici, ad esempio:

Immagine che contiene schizzo, disegno, linea, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.non può essere riconosciuto da un automa di Büchi deterministico come:

Immagine che contiene schizzo, disegno, cerchio, diagramma

Il contenuto generato dall'IA potrebbe non essere corretto.Infatti, questo accetterebbe tutte le stringhe che hanno almeno una , mentre la versione non deterministica sì:

Ora, modellizziamo così il complemento della liveness property da verificare, costruiamo l’automa del prodotto (in maniera analoga a quanto visto con gli FSA) e cerchiamo un ciclo in esso che include uno stato di accettazione (ricordiamo come avviene l’accettazione negli automi di Büchi).

L’unica puntualizzazione che facciamo rispetto alla verifica delle safety properties è il fatto che è molto complesso costruire il complemento di un auotma di Büchi non deterministico ().

* 1. La logica temporale per il model checking

Nella logica temporale, la veridicità di una formula non dipende solo dalla formula stessa ma dal tempo nel quale essa è enunciata (es. “oggi piove” : oggi è falsa, domani potrebbe essere vera).

* + 1. *LTL* (“*Linear Temporal Logic*”)

Potremmo definirla come una logica preoposizionale arricchita dalla definizione di “tempo”. In altre parole, la semantica di una formula in logica temporale è data su “parole” infinite, dove per “parole” intendiamo il grado di verità della formula in un certo istante.

Siano:

* , un generico set di lettere predicative;
* e , due generiche formule LTL;

una formula LTL può essere:

i)

ii)

iii)

iv)

v)

vi)

dove:

- (“next step”) : è vera se e solo se nel prossimo istante è vera;

- (“until”) : è vera se e solo se è vera fino a quando non diventa vera. [e nell’istante corrente?]

Osservazioni:

* spesso possono venire comodi:
  + (“eventually”) : vera se e solo se prima o poi è vera (o anche in questo istante);
  + (“globally”) : vera se e solo se per l’istante corrente e tutti gli istanti seguenti è vera;
* per ciascun operatore indichiamo col simbolo “nero” (es. per “” usiamo “”) il corrispettivo operatore enunciato nel passato, ovvero:
  + (“previous step”) : è vera se e solo se nel precedente istante era vera;
  + (“since”) : è vera se e solo se … ;
  + (“sometimes in the past”) : ;
  + (“always in the past”) : ;
* , e □ (e analogamente le loro versioni “nel passato”) possono essere generalizzati al caso di “ istanti” (per intendiamo l’istante attuale):
  + : è vera se e solo se …;
  + : è vera se e solo se è vera nel prossimo -esimo istante;
  + : è vera se e solo se è vera in almeno uno dei prossimi istanti (anche quello attuale);
  + : è vera se e solo se è vera in tutti i prossimi istanti (compreso quello attuale).

Qualche esempio:

* “in nessun istante ambedue i processi possono utilizzare la sezione critica” :
* “ogni processo entra nella sezione critica indefinitamente” :
* “a una richiesta segue, prima o poi, una risposta” :
* “una volta rosso, il semaforo non può diventare subito verde” :
  + 1. Esprimere fairness asssumptions in LTL

Cose a cui stare attenti:

* costruire l’interleaving: esempio dei semafori nella spiegazione della fairness;
* FSA riconoscitore dei bad-prefix che “non accetta più caratteri” quando arriva nello stato finale;
* “indefinitamente = infinitely often” per me
  + 1. LTL Model checking

Per applicare l’LTL al model checking, occorre scrivere le formule in *forma normale positiva*. In questa forma, molto semplicemente, deve essere applicato solo alle lettere proposizionali e non alle espressioni (ad esempio, non è in forma normale positiva, mentre sì).

Capiamo dunque che per portare una generica formula in forma normale positiva necessitiamo dei duali degli operatori della logica temporale. Oltre alle leggi di De Morgan (che, naturalmente, continuano a valere), valgono le seguenti uguaglianze:

|  |  |
| --- | --- |
|  |  |
|  |  |

dove:

* [ è detto *Release*]

Immagine che contiene diagramma, linea, schizzo, cerchio

Il contenuto generato dall'IA potrebbe non essere corretto.Si può dimostrare che qualunque formula LTL può essere rappresentata per mezzo di un automa di Büchi (potenzialmente non deterministico), ad esempio , può essere rappresentata come:

Si può dimostrare che il automi di Büchi non deterministici sono strettamente più espressivi dell’LTL (es. “ è vera solo negli step pari” può essere modellata solo dagli automi di Büchi).

Quanto al model checking, si dimostra che:

dove:

* : è l’insieme di -word che soddisfa .

Anziché generare l’automa di e complementarlo si complementa e se ne genera l’automa (molto più efficiente computazionalmente, in quanto la complementazione di un atoma di Büchi ha complessità esponenziale).

Immagine che contiene testo, schermata, diagramma, Carattere

Il contenuto generato dall'IA potrebbe non essere corretto.Il procedimento generale è il seguente:

Si osservi che:

* la complessità del processo è , dove:
  + è il numero di stati del ;
  + è il numero di letterali di .
* questo algoritmo si può applicare man mano che si costruisce l’automa di Büchi, il che potrebbe risparmiare la costruzione completa dell’automa!

Notiamo che per verificare la soddisfacibilità di una formula LTL è possibile verificare che il relativo automa non accetti solo l’insieme vuoto (si applica una DFS innestata).

* 1. Bounded model checking
  2. a