Relazione Esercizio Timed Automata

Francesco Galla‘, francesco.galla@edu.unito.it

1 Modello A

MODELLO A. Si assume che il canale sia perfetto, e quindi nè il messaggio, nè lack possono essere persi. Il tempo di trasmissione sul link è variabile allinterno di un intervallo limitato, con poca variabilità (quindi upper e lower bound del tempo di trasmissione sul link sono molto vicini, diciamo non sopra un decimo del tempo di trasmissione).

1.1 Il modello Uppaal

Il modello è composto da 3 template, rappresentanti un sender, un receiver e un link.

1.1.1 Sender

Tra gli stati del processo *Sender*, si nota lo stato iniziale di *wait* **waitPacket** in cui il processo attende un evento per cominciare la preparazione di un pacchetto. Una volta che il pacchetto è pronto per essere mandato (con un tempo compreso nell'intervallo $[1, 4]$ del clock *sc*) allora si passa allo stato **canSend**. Da qui, il pacchetto viene mandato sul canale di sincronizzazione ML (urgente) e il *Sender* passa a **waitACK**. La ricezione dell'ACK avviene sul canale di sincronizzazione LM (urgente) e il clock *sc* viene resettato. Ricevuto l'ACK, il *Sender* rimane nello stato di **receivedACK** in cui occupa un tempo nell'intevallo $[2, 4]$ del clock *sc* per processarlo, per poi resettare nuovamente il clock e ritornare allo stato iniziale.

1.1.2 Receiver

Il processo *Receiver* segue una dinamica simile a quella del *Sender*: anche questo ha 4 stati, di cui l'iniziale **waitPacket** è l'attesa di un pacchetto (mandato dal *Sender* e trasportato dal *Link* al *Receiver* sul canale di sincronizzazione LR (urgente). Ricevuto il pacchetto, il receiver utilizza un tempo compreso nell'intevallo $[2, 4]$ per processarlo in **receivedPacket** e passare nello **donePacket** resettando il clock *rc*. Qui impiega tra 2 e 3 secondi per costruire un ACK e passare nello stato **canACK**, da cui può mandare l'acknowledgement sul canale di sincronizzazione RL (urgente).

1.1.3 Link

Il processo *Link* è modellato senza perdite (canale perfetto) e utilizza 5 stati, di cui quello iniziale è considerato uno stato interno e modella le situazioni in cui i processi $Sender, Receiver$ stanno eseguendo operazioni locali (per cui non c'è comunicazione sul canale). Gli altri stati si comportano in maniera simile agli stati già visti negli altri due processi, utilizzando il clock lc.

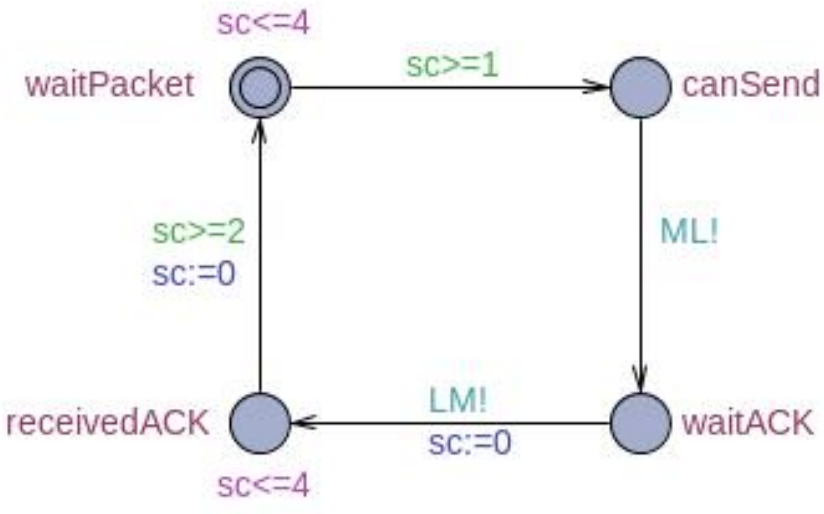

Figure 1: Sender A

1.2 Analisi TCTL delle proprietà

Sono state controllate 4 proprietà sul modello A. Essendo un modello *perfetto*, senza perdite nè rumore, si nota come rispetti tutte le proprietà verificate.

1. $AG(notdeadlock)$ è vera, quindi la traccia del sistema di processi è infinita.
2. $sender.waitACK -- > (sender.receivedACK)$ è vera, in quanto senza perdite il sender sarà sempre in grado di ricevere l'ACK entro un tempo finito.
3. $AFreceiver.receivedPacket$ è vera, in quanto non ci sono constraint sul tempo di attesa di $receiver.waitPacket$, pertanto un pacchetto verrà sempre ricevuto.
4. Il tempo minimo e massimo tra l'invio del messaggio e la ricezione dell'ACK corrispondente da parte del *Sender* sono ottenibili grazie a un clock aggiuntivo fc che viene resettato ogni volta che un pacchetto viene

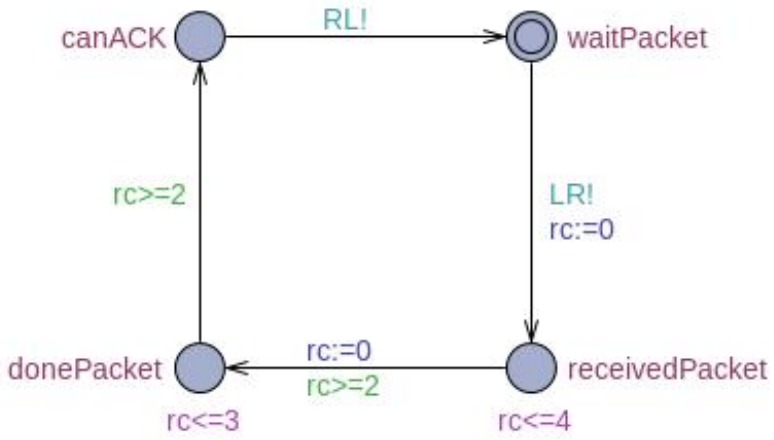

Figure 2: Receiver A

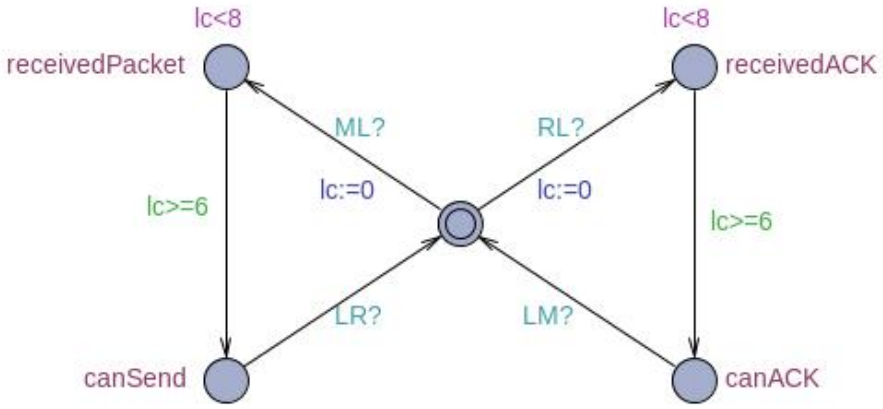

Figure 3: Link A

mandato sul canale ML. Per questo nello stato *sender.canSend* contiene il tempo minimo e massimo espressi come *bound* dell'intervallo, in questo caso $[19, 31]$.

2 Modello B

MODELLO B. Si modifichi il modello in modo da prevedere la possibilit che il link perda un messaggio o un ack (la perdita equivale sia alla perdita effettiva che al caso di messaggio corrotto, perch lalgoritmo si comporta in modo analogo). Non cambiate il protocollo, modellate semplicemente un link con perdita o corruzione del messaggio.

2.1 Il modello Uppaal

Il modello è composto da 3 template, rappresentanti un sender, un receiver e un link. Il sender e il receiver mantengono la struttura del modello A, mentre nel *Link* vengono introdotte due perdite possibili: una per il pacchetto e una per l'ACK.

2.1.1 Link

Il *Link* in questo caso ha possibilità di perdere i pacchetti (scelta non deterministica) in due stati: $receivedPacket$ e $receivedACK$, ovvero quelli in cui è stato mandato un messaggio dal sender al link o dal receiver al link. Si nota come l'invariante degli due stati sia $lc < 8$, dove con $lc == 8$ si sa con certezza che il pacchetto è stato perso oppure ricevuto correttamente. Per implementare la perdita è stato utilizzato un posto aggiuntivo rispetto al modello A, chiamato *lostPacket*, che riporta allo stato fantasma del link senza però mandare il pacchetto o l'ACK.

2.2 Analisi TCTL delle proprietà

1. $AG(notdeadlock)$ è falsa, infatti la traccia del simulatore mostra come un pacchetto perso causi deadlock perchè *Sender* e *Receiver* sono rimasti in attesa di risposta e il canale non ha più comunicazioni al suo interno.

2. $sender.waitACK -- > (sender.receivedACK)$ è falsa, per lo stesso motivo per cui la 1. lo è.

3. $EFreceiver.receivedPacket$ è vera, perchè nonostante la possibilità di deadlock, c'è almeno una traccia in cui il pacchetto viene consegnato correttamente. Questo significa che i pacchetti non sono persi con sicurezza ma tramite una scelta non deterministica.

4. Il tempo minimo e massimo tra l'invio del messaggio e la ricezione dell'ACK corrispondente da parte del *Sender* sono ottenibili grazie a un clock aggiuntivo fc che viene resettato ogni volta che un pacchetto viene mandato sul canale ML. Dato che abbiamo una probabilità di perdita all'interno del canale, possiamo stabilire con sicurezza solo il tempo **minimo**, che risulta equivalente a quello del modello A.

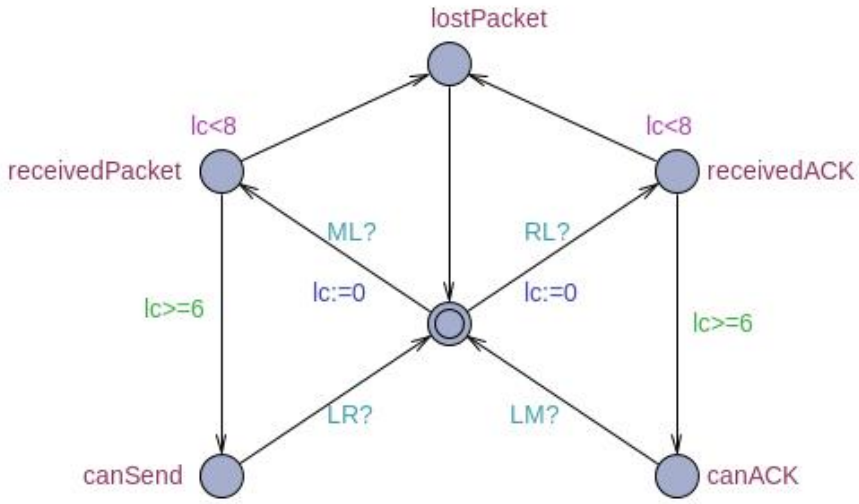

Figure 4: Link B

3 Modello C

MODELLO C. Mantenendo il comportamento del canale previsto per il modello B, si modelli il mittente e il ricevente descritti dallalgoritmo 3.

3.1 Il modello Uppaal

Il modello è composto da 3 template, rappresentanti un sender, un receiver e un link. Il sender è stato implementato in due versioni, una che utilizza un timer singolo e una che utilizza due timer. Il link non cambia rispetto al modello B.

Si utilizzano inoltre 3 variabili globali booleane aggiuntive: $next, ack, frame$. La scelta di utilizzare variabili booleane è data dal fatto che in STOP&WAIT i messaggi sono mandati singolarmente, per cui basta differenziare due messaggi successivi per riconoscere i duplicati, come effettuato nel *Receiver*.

3.1.1 Sender (timer singolo)

Si estende il template *Sender* del Modello A con la possibilità di inviare nuovamente i messaggi nel caso questi venissero persi, nel caso non fosse ricevuto un ACK e nel caso l'ACK ricevuto non corrispondesse a quello atteso. Per fare ciò sono state utilizzate le variabili globali di cui sopra e uno stato *lostPacket* il cui accesso viene controllato da una guardia sul timer $x0$ (singolo). Se il timer "scatta", ovvero se trascorre il tempo stabilito nello stato $waitACK$, allora è forzato lo stato *lostPacket* che riportando a *canSend* senza cambiare i valori

di $frame$ e $next$ permette la ritrasmissione. Inoltre, per il controllo della correttezza del'ACK, si utilizza una transizione con guardia che porta dallo stato $receivedACK$ allo stato $lostPacket$, per ritrasmettere nel caso l'ACK ricevuto non fosse corretto.

3.2 Sender (timer doppio)

Per il sender a timer doppio, è stata duplicata la struttura del primo sender di modo da permettere una scelta tra $next == 1$ e $next == 0$. In questo modo si può discriminare tra i due tipi di frame possibili.

3.2.1 Receiver

Al receiver del Modello A viene aggiunta la capacità di riconoscere pacchetti duplicati tramite una variabile locale $correctFrame$ e uno stato $dupPacket$. La transizione a questo stato viene controllata da una guardia che discrimina se il frame ricevuto è corrispondente a quello atteso. Nel caso il frame non corrispondesse, viene aggiornata la variabile ack e viene mandato l'ACK per il pacchetto ricevuto, seppur duplicato, per permettere il proseguimento del sender.

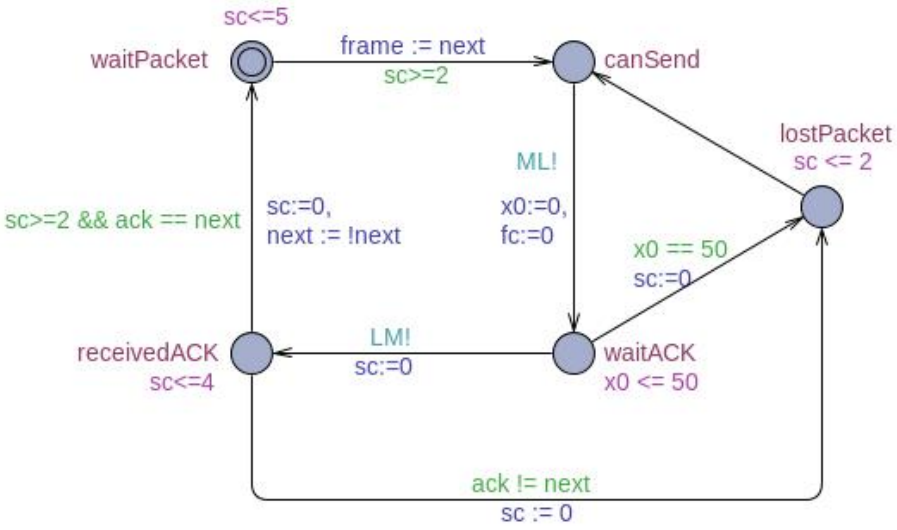

Figure 5: Sender C (1 timer)

3.3 Analisi TCTL delle proprietà

Il sistema rispetta le proprietà seguenti sia con un timer sia con due, dimostrando che non c'è necessità per questo modello di due timer. Inoltre, il valore dei timer (settato a 50 in entrambi i casi) deve essere maggiore del tempo impiegato da

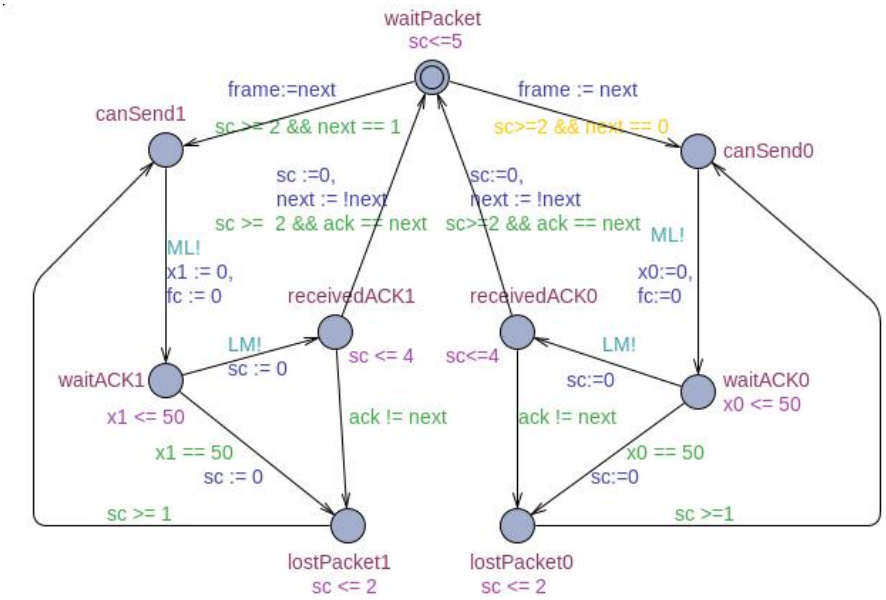

Figure 6: Sender C (2 timer)

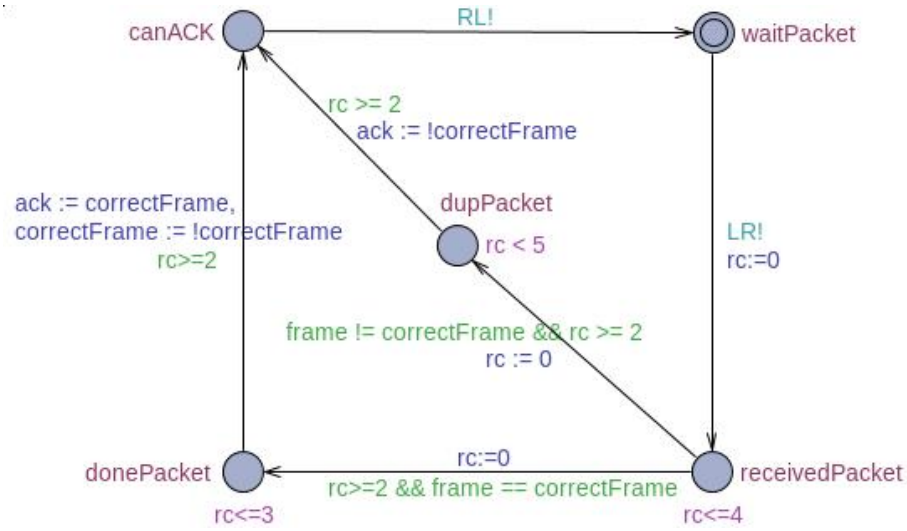

Figure 7: Receiver C

un ACK a tornare al sender, ossia due volte il tempo impiegato dal canale per spedire un pacchetto più l'eventuale tempo di elaborazione nel receiver. Altrimenti si ha *deadlock*, perchè il sender non riesce a spedire nuovamente il pacchetto se canale è occupato dall'ACK del receiver. Il valore minimo del timer

per questo modello è 42.

1. $AG(notdeadlock)$ è vera, quindi la traccia del sistema di processi è infinita.

2. $sender.waitACK0||sender.waitACK1-->(sender.receivedACK0||sender.receivedACK1)$ è falsa, in quanto il canale può subire perdite.

3. $E <> (sender.receivedACK0\&\&ack == next)$ è vera, infatti esiste almeno un caso in cui l'ACK ricevuto dal sender è corretto (e quindi il modello è in grado di passare messaggi correttamente).

4. $link.lostPacket-->(sender.canSend0||sender.canSend1||receiver.canACK)$ è vera, in quanto in caso di pacchetto perso il sistema non si blocca ma permette al sender di mandare di nuovo il messaggio o al receiver di trasmettere l'ACK.

4. Il tempo minimo e massimo tra l'invio del messaggio e la ricezione dell'ACK corrispondente da parte del *Sender* sono ottenibili grazie a un clock aggiuntivo fc che viene resettato ogni volta che un pacchetto viene mandato sul canale ML. Per questo nello stato *sender.canSend* contiene il tempo minimo e massimo espressi come *bound* dell'intervallo, in questo caso $[21, 32]$.