RELAZIONE ESERCIZIO SUI TIMED AUTOMATA CON IL TOOL UPPAAL

Laboratorio lungo 9 CFU

Marco Edoardo Santimaria
Matricola 912404

26 agosto 2023

Indice

1 Modello A e Modello B

1.1 Modellazione

Il modello A, e' stato modellato nel seguente modo:

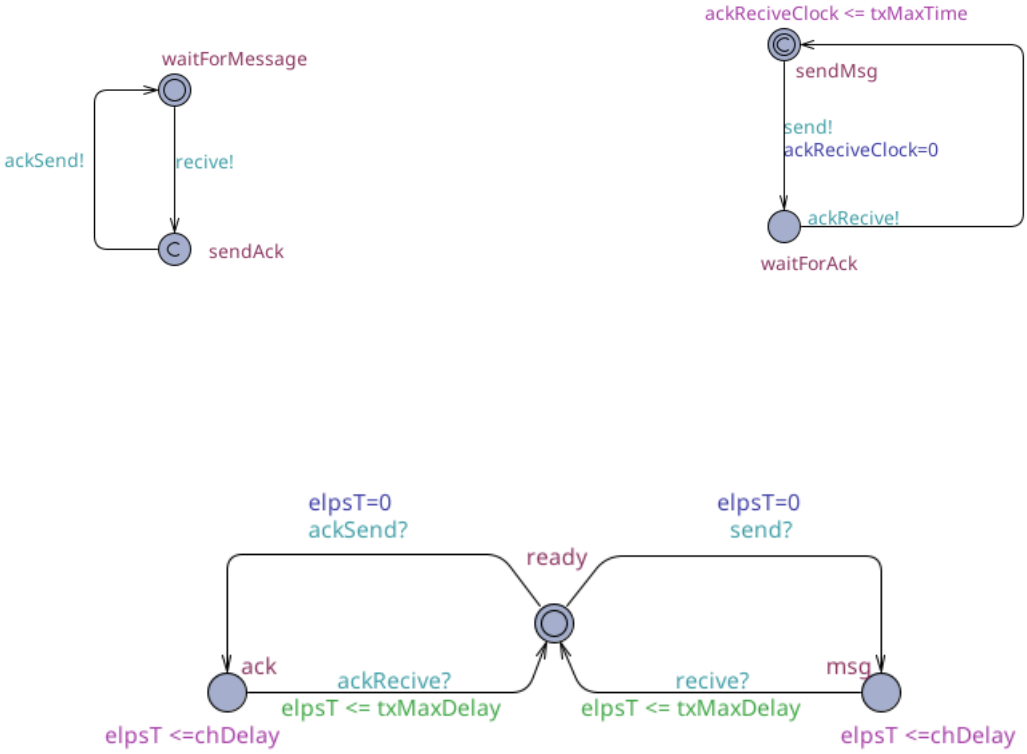

Figura 2: Canale di comunicazione per modello A

La posizione committed nel reciver e' stata inserita per modellare il fatto che il ricevitore deve inviare subito un messaggio di ack per il pacchetto ricevuto.
Le variabili che hanno un ruolo all'interno del modello del sistema sono le seguenti:

- **sender (s)**
  - **clock ackReciveClock** Timer associato alla ricezione dell'ack per il pacchetto inviato (ne basta uno essendo la finestra di trasmissione a 1.
- **channel (c)**
  - **clock elpsT** elapsedTime: ovvero tempo che il canale ha impiegato a inviare un messaggio
- **reciver (r)**
  - nessuna variabile utilizzata
- **condivise**
  - **chan send** Canale di invio dei messaggi
  - **chan recive** Canale di ricezione dei messaggi
  - **chan ackSend** Canale di invio degli ack
  - **chan ackRecive** Canale di ricezione degli ack

- **int[1,5] chDelay** Variabile che esprime il ritardo introdotto dal canale di comunicazione
- **int txMaxTime** Variabile che esprime il tempo massimo di trasmissione di un pacchetto e di ricezione del suo ack. Usata solo per verifica delle proprieta' in questo modello.
- **s,r,c** Instanze dei template di Sender, Reciver e di Channel

1.1.1 Verifica delle proprieta'

le proprieta' sono state modellate nel seguente modo, e hanno tutte una valutazione come true

Le proprieta' definite per il modello sono le seguenti, e hanno tutte la valutazione true:

- **Assenza di deadlock**

```
A[] not deadlock
```

- **Verifica che il ritardo massimo introdotto sulla linea sia sempre minore uguale al tempo massimo concesso per la trasmissione**

```
A[] s.ackReciveClock <= txMaxTime
```

- **Verifica del tempo masismo effettivo di trasmissione**

```
A[] s.ackReciveClock <= (2*chDelay)
```

  In questo casi si e' andato a verificare che il ritardo sia sempre minore uguale al ritardo introdotto sulla linea di trasmissione.

- **Proprieta' di corretto funzionamento**

```
s.sendMsg --> c.msg
c.msg --> r.sendAck
r.sendAck --> c.ack
c.ack --> s.sendMsg
```

1.2 Variazione del canale per modello B

La modifica effettuata e' stata la seguente: Per il resto il modello rimane identico. Questa modifica ha cambiato anche la

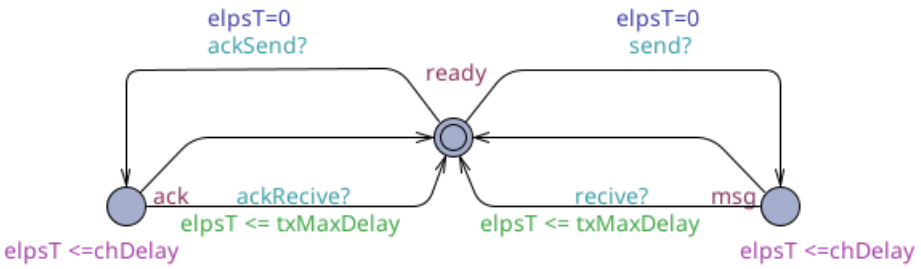

Figura 3: Canale di comunicazione per modello B

valutazione delle proprieta', dove in particolare sono state rese false tutte le proprieta' sopra descritte ad eccezione di:

- c.msg – r.sendAck
- r.sendAck – c.ack

Che ovviamente sono rimaste vere, in quanto riguardano una situazione in cui il canale ha svolto il suo compito senza perdere pacchetti.

2 Modello C

2.1 Modellazione

Il modello e' stato realizzato con questi tre automi:

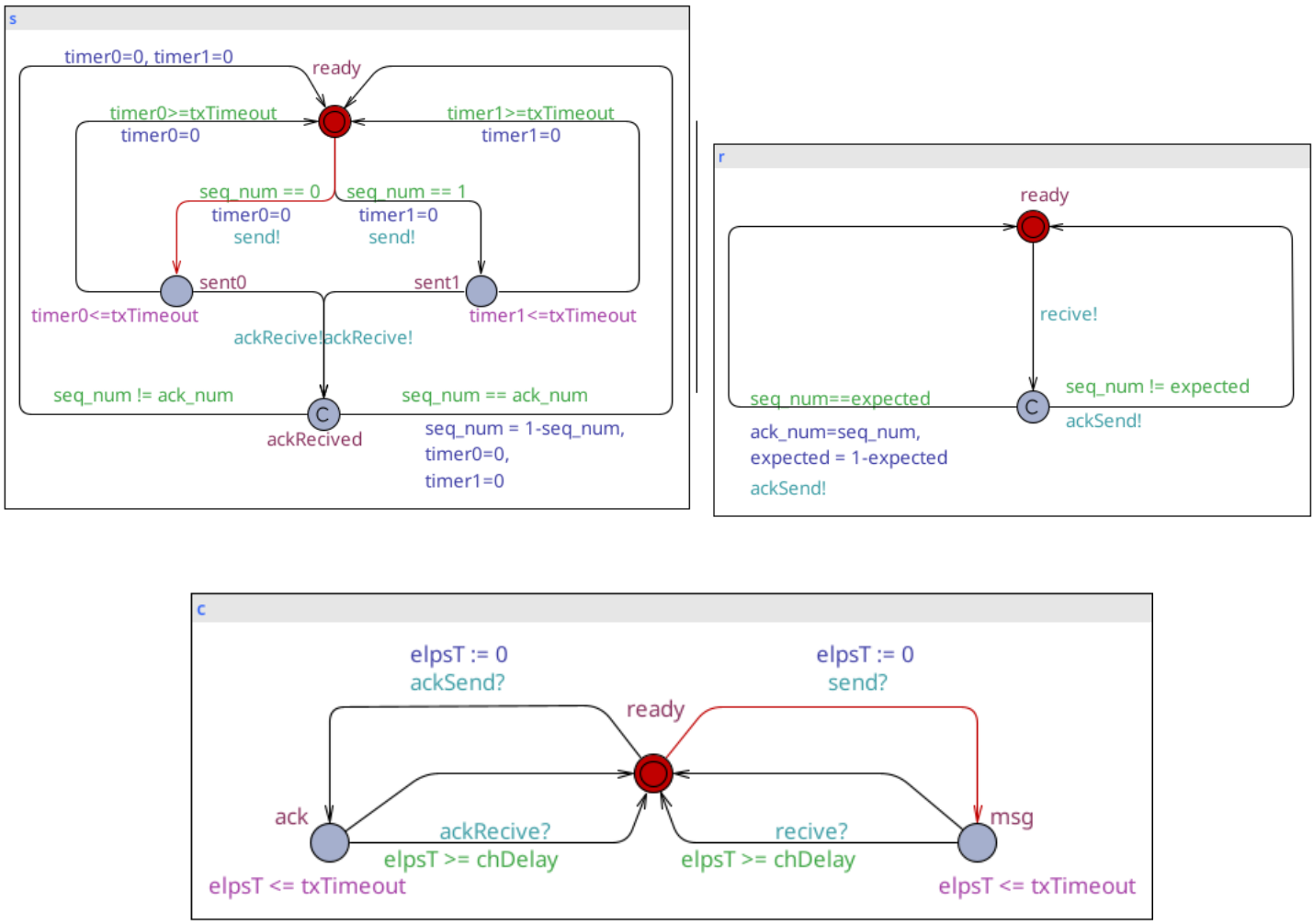

Figura 5: Canale di comunicazione per modello A

Le due posizione committed, sono state usate solo per esprimere che una volta ricevuto il messaggio esso deve essere elaborato subito (ovvero non ci devono essere altre azioni eseguite prima dell'elaborazione del messaggio).
Non e' stato impostato alcun delay nella elaborazione dei messaggi sia da parte del sender che del reciver in quanto ho ipotizzato un tempo di elaborazione negligibile in contronto ai ritardi e al tempo impiegato dalle altre componenti del modello (anche perche' quasi sempre nella realta' l'elaborazione dei pacchetti a livello di trasport e di data lynk sono realizzati con accellerazione hardare dove possibile).
Le variabili che hanno un ruolo all'interno del modello del sistema sono le seguenti:

- **sender (s)**
  - **clock timer0** Timer associato all'invio del messaggio con numero di sequenza 0
  - **clock timer1** Timer associato all'invio del messaggio con numero di sequenza 1
- **channel (c)**
  - **clock elpsT** elapsedTime: ovvero tempo che il canale ha impiegato a inviare un messaggio
- **reciver (r)**

- **int expected** il numero di sequenza che il ricevitore si aspetta dal trasmettitore

- **condivise**
  - **chan send** Canale di invio dei messaggi
  - **chan recive** Canale di ricezione dei messaggi
  - **chan ackSend** Canale di invio degli ack
  - **chan ackRecive** Canale di ricezione degli ack
  - **int[1,5] chDelay** Variabile che esprime il ritardo introdotto dal canale di comunicazione
  - **int txTimeout** Variabile che esprime il timeout, che quando scaduto (time0/timer1 == txTimeout), considera il messaggio come perso e procede al reinvio.
  - **int[0,1] seq_num** Numero di sequenza del messaggio che viene inviato al momento
  - **int[0,1] ack_num** Numero di sequenza a cui il messaggio ack si riferisce
  - **s,r,c** Instanze dei template di Sender, Reciver e di Channel

2.2 Verifica delle proprieta'

Le proprieta' definite per il modello sono le seguenti, e hanno tutte la valutazione true:

- **Assenza di deadlock**

```
A[] not deadlock
```

- **Se il numero di sequenza e' 0 (o 1 nel caso speculare), non effettuo mai un sent1 (o sent0 nel caso speculare**

```
(s.sent0 && s.seq_num == 0) --> (not s.sent1)
```

  e anche

```
(s.sent1 && s.seq_num == 1) --> (not s.sent0)
```

  Questa specifica e' da spiegare perche' nasconde piu di quello che traspare. Se mi trovo con l'automa sender in sent0, allora posso avere due casi: o ho inviato appena il messaggio oppure ho appena effettuato una ritrasmissione. quello che vado a garantire e; che fino a che sto cercando di inviare un messaggio con numero di sequenza == 0, non andro' mai a effettuare una send1 (invio con numero di sequenza con 1), perche' se cosi' fosse vorrebbe dire che ho ricevuto un ack diverso da quello per cui sto inviando un messaggio.

- **Verifica del tempo minimo di ricezione**

```
A[] s.ackRecived imply (s.timer0 >= 2*chDelay || s.timer1 >= 2*chDelay)
```

  In questo caso ho verificato che il tempo minimo di ricezione e' fissato a chDelay

- **Verifica che il timer scatta se il pacchetto e' stato perso**

```
A[] (   (s.timer0 > txTimeout && seq_num==0) ||
        (s.timer1 > txTimeout && seq_num==1 )
    ) imply (c.ready && r.ready)
```

2.3 Commenti vari

Per questo esercizio e' stato richiesto di rispondere a una serie di domande, in particolare:

1. E' possibile identificare un valore di timer che ci permette di affermare con certezza che se il timer scatta il messaggio è stato perso? Questo valore che impatto ha sull'efficienza del canale trasmissivo?

   Questo e' possibile ed e' stato fatto con la verifica. in particolare essendo stato impostato il tempo di ritardo di un messaggio a meno della meta' dello scadere del timeout, si puo affermare che allo scadere del timeout il messaggio e' andato perso. Ovviamente il valore di questo timer influisce sulla larghezza di banda e sulla reattivita' che il protocollo supporta: questo timer deve essere in ogni caso sempre maggiore del doppio del ritardo introdotto dal canale (invio e ricezione del ack). piu ci si distanzia verso l'alto da questa soglia del doppio del ritardo introdotto dal canale, piu si rallenta il protocollo senza avere alcun beneficio in termini di correttezza. e' altresi vero che nella realta' questo timeout deve tenere conto di jitter e di eventuale sovraccarico del canale (che potrebbe essere modellato con un ritardo randomico entro un certo intervallo).

2. L'algoritmo prevede l'utilizzo di due timer, uno per ogni possibile numerazione dei messaggi: sono davvero necessari ambedue?

   No, il secondo timer e' superfluo. Questo perche' essendo la finestra di ricezione di un solo pacchetto, ed essendo il protocollo stop and wait, basta un timer che misura il tempo di "viaggio" del pacchetto in viaggio. Anche se la finstra di ricezione fosse maggiore di uno, si potrebbe usare l'approccio di TCP (ovvero inviare un ack sempre per l'ultimo pacchetto ricevuto, e allo scadere del timer, riavviarlo all'ultimo pacchetto piu vecchio senza ack).