Linguaggi e Modelli per il Global Computing
Modellazione e verifica di un sistema concorrente

Filippo Sestini

29 settembre 2015

Indice

Sommario

Si vuole illustrare un semplice esempio di modellazione di un sistema concorrente in *Calculus of Communicating Systems*, e mostrare come, attraverso alcuni formalismi, sia possibile verificarne il comportamento e valutarne alcune caratteristiche rispetto a una specifica.

1 Descrizione del sistema

Il sistema preso in considerazione è il protocollo CSMA (*Carrier Sense Multiple Access*) con *Collision Detection*, parte del sublayer *Media Access Control*.

Media Access Control è un sottostrato *data-link* che si occupa di gestire e controllare la comunicazione tra più terminali, chiamati *Media Access Controller*, collegati ad una rete costituita da un *medium* condiviso.

Dello strato MAC fanno parte diversi protocolli, tra cui il più diffuso CSMA/CD. CSMA/CD è un protocollo principalmente utilizzato come Medium Access Control in reti Ethernet half-duplex. La gestione dell'accesso al medium condiviso avviene attraverso *carrier sensing* e periodi di *contention*, attraverso i quali viene assegnato l'accesso esclusivo del medium ad una stazione per la trasmissione.

2 Modellazione in CCS

La clausola 3 dello standard IEEE 802.3 (Ethernet) descrive la specifica del servizio offerto da un generico protocollo MAC al client, identificato dal livello di rete superiore, delineando una trasmissione di tipo *half-duplex* e un'interfaccia costituita principalmente da due operazioni, rispettivamente per l'invio e la ricezione di messaggi verso una o più stazioni di destinazione. La clausola 4 presenta invece un'accurata descrizione del protocollo CSMA/CD, sulla quale è basato, con il dovuto grado di astrazione e semplificazione, il protocollo modellato di seguito.

Il modello prevede, per semplicità, la comunicazione tra due stazioni collegate a un medium condiviso, identificate con MAC1 e MAC2 (dove MAC sta per Media Access Controller.) Ogni controller può inviare o ricevere messaggi dall'altro capo del canale trasmissivo. Per effettuare la trasmissione di un messaggio, è necessario che la stazione trasmettitrice acquisisca l'utilizzo esclusivo del canale medium, passando attraverso un periodo di *contention* durante il quale entrambe le stazioni hanno la libertà di iniziare a trasmettere sul canale.

Durante il periodo di *contention* possono verificarsi *collisioni* nella trasmissione. Si ha una collisione quando le due stazioni avviano contemporaneamente una trasmissione: a seguito di una collisione, il periodo di *contention* reinizia e le stazioni tentano nuovamente di acquisire il canale. Un stazione riesce ad acquisire l'utilizzo esclusivo del canale se riesce a trasmettere oltre il periodo di *contention* senza collisioni.

Lo standard del protocollo prevede che, a seguito di più collisioni successive, i controller modifichino il tempo di attesa tra un tentativo e l'altro per massimizzare le probabilità di successo. Il modello realizzato di seguito astrae da questo dettaglio, e rappresenta le collisioni allo stesso modo: di conseguenza, c'è la possibilità che i due sistemi collidano per un numero indeterminato di volte, potenzialmente infinito. Come si vedrà in Sezione 3.1, comunque, tale dettaglio è trascurabile e non altera le possibilità di verifica del modello, nè il significato del suo risultato.

Dall'analisi dello standard si può ricavare una rappresentazione del protocollo da verificare, e una specifica che identifichi le caratteristiche che si vogliono soddisfatte per questo protocollo. Vengono poi utilizzati due approcci nel verificare l'aderenza del modello di protocollo realizzato alla specifica. Il primo consiste nell'esprimere tale specifica, e quindi il comportamento che si vuole ottenere, con termini in CCS, e stabilire una relazione di equivalenza tra i processi della specifica e dell'implementazione. Il secondo utilizza la logica di Hennessy-Milner per esprimere proprietà desiderate del processo modellato, e verificare che queste siano soddisfatte. La sezione che segue contiene una rappresentazione in CCS della specifica del sistema.

2.1 Specifica

La specifica del protocollo da modellare ha le caratteristiche riassunte di seguito:

1. L'accesso al canale deve avvenire in modo mutuamente esclusivo;
2. Il protocollo deve offrire operazioni di invio e ricezione di messaggi attraverso il canale trasmissivo. Ad una trasmissione terminata con successo deve seguire la ricezione del messaggio nella stazione di destinazione;
3. È possibile richiedere l'invio di un nuovo messaggio quando quello precedente non è ancora stato ricevuto dal client della stazione di destinazione; è però necessario che questo sia ricevuto prima di poter effettivamente iniziare una nuova trasmissione.

A questo livello di astrazione, il comportamento che dovrà assumere il medium condiviso è sostanzialmente equivalente ad un semaforo binario, come rappresentato in Figura 1 e dalla seguente espressione:

$$MediumSpec \stackrel{\text{def}}{=} p.v.MediumSpec$$

Lo scambio di messaggi, essendo di tipo *half-duplex*, può essere invece specificato da due processi rispettivamente per la comunicazione da MAC1 a MAC2 e, specularmente, da MAC2 a MAC1.

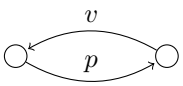

Figura 1: Specifica del medium

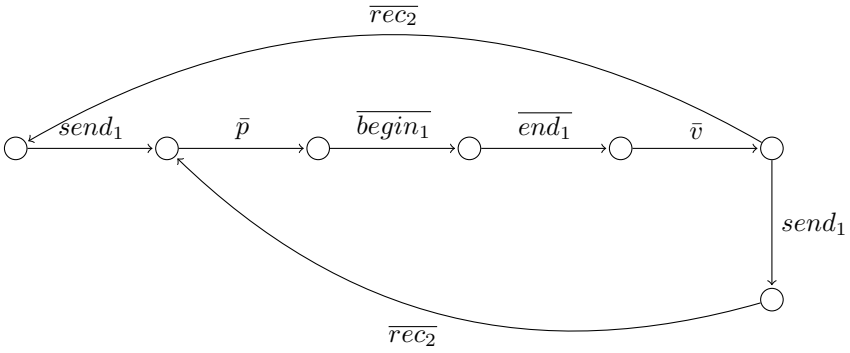

Figura 2: Specifica della comunicazione tra MAC1 e MAC2

Questi processi sono rappresentati graficamente in Figura 2 e Figura 3 e dalle seguenti codifiche in CCS:

$$\begin{aligned} Spec_{12} &\stackrel{\text{def}}{=} send_1.SendingSpec_{12} \\ SendingSpec_{12} &\stackrel{\text{def}}{=} \bar{p}.\overline{begin_1}.\bar{v}.(\overline{rec_2}.Spec_{12} + send_1.\overline{rec_2}.SendingSpec_{12}) \end{aligned}$$

$$\begin{aligned} Spec_{21} &\stackrel{\text{def}}{=} send_2.SendingSpec_{21} \\ SendingSpec_{21} &\stackrel{\text{def}}{=} \bar{p}.\overline{begin_2}.\bar{v}.(\overline{rec_1}.Spec_{21} + send_2.\overline{rec_1}.SendingSpec_{21}) \end{aligned}$$

I canali utilizzati dai processi della specifica sono i seguenti:

- *send*: invio di un messaggio;

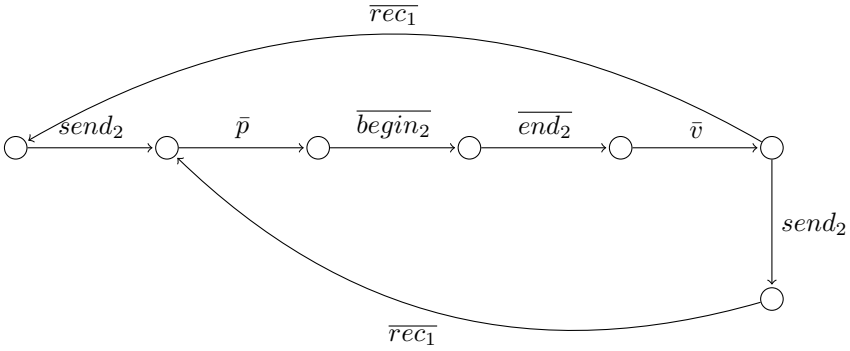

Figura 3: Specifica della comunicazione tra MAC2 e MAC1

- *begin*: inizio della trasmissione esclusiva sul canale;
- *end*: termine della trasmissione esclusiva sul canale;
- *rec*: ricezione di un messaggio;
- *p*: acquisizione del semaforo;
- *v*: rilascio del semaforo.

A seguito di una richiesta di invio di un messaggio (*send*), la stazione tenta di acquisire il controllo esclusivo del canale. Terminata la trasmissione, il messaggio viene ricevuto dal client della stazione di destinazione (*rec*), oppure, poichè i due controller operano in parallelo, è possibile richiedere l'invio di un altro messaggio prima che quello appena trasmesso sia ricevuto dal client della stazione di destinazione. Infine, le etichette *begin* e *end* identificano l'inizio e la fine della trasmissione esclusiva sul medium da parte di uno dei due controller.

Indicando i processi in Figura 2 e 3 rispettivamente con $Spec_{12}$ e $Spec_{21}$, e il processo del *medium* in Figura 1 con $MediumSpec$, la specifica del servizio si può rappresentare con la seguente espressione in CCS:

$$Spec \stackrel{\text{def}}{=} (Spec_{12}|Spec_{21}|MediumSpec) \setminus \{p, \bar{p}, v, \bar{v}\}$$

2.2 Protocollo

Seguendo quanto indicato dallo standard del protocollo, il processo di ogni controller viene modellato come composizione parallela di due processi, uno trasmettitore e uno ricevitore, i quali hanno poi accesso ad un *medium* condiviso. I messaggi utilizzati dal protocollo sono i seguenti:

- *send*: presa in carico dell'invio di un messaggio alla stazione;
- *begin*: inizio trasmissione esclusiva nel canale;
- *end*: termine trasmissione esclusiva nel canale;
- *bt* (begin transmission): inizio trasmissione nel canale;
- *et* (end transmission): termine trasmissione nel canale;
- *br* (*begin reception*): inizio ricezione messaggio nel ricevitore;
- *er* (*end reception*): termine ricezione messaggio nel ricevitore;
- *rec* (*receive*): output del messaggio ricevuto al client;
- *c* (*collision*): avvenuta collisione nel canale trasmissivo;
- *acq* (*acquisition*): avvenuta acquisizione del canale trasmissivo;
- *bad*: avvenuta collisione nel canale trasmissivo durante la ricezione. La porzione di messaggio ricevuta deve essere scartata.

Quando viene richiesto al controller di inviare un messaggio (*send*), questo inizia la trasmissione sul canale (*bt*). Durante la trasmissione possono esserci trasmissioni da parte dell'altra stazione collegata, in seguito ai quali avviene una collisione (*c*) che riporta il controller ad effettuare un nuovo tentativo. Nel caso il segnale iniziale riesca a propagarsi fino al ricevitore di destinazione senza collisioni (*acq*), il periodo di *contention* termina e viene garantito alla stazione al momento in trasmissione l'utilizzo esclusivo del canale (*begin* e *end*.) Terminata la trasmissione, il trasmettitore (*et*) e il ricevitore (*er*) rilasciano il canale. Il messaggio ricevuto dal controller viene poi comunicato al client (*rec*.)

I processi trasmettitore sono rappresentati graficamente in Figura 4, e possono essere rappresentati in CCS come segue:

$$\begin{aligned} Transmitter_1 &\stackrel{\text{def}}{=} send_1.Transmitting_1 \\ Transmitting_1 &\stackrel{\text{def}}{=} \overline{bt_1}.Transmit_1 \\ Transmit_1 &\stackrel{\text{def}}{=} c.Transmitting_1 + acq_1.\overline{begin_1}.\overline{end_1}.\overline{et}.Transmitter_1 \end{aligned}$$

$$\begin{aligned} Transmitter_2 &\stackrel{\text{def}}{=} send_2.Transmitting_2 \\ Transmitting_2 &\stackrel{\text{def}}{=} \overline{bt_2}.Transmit_2 \\ Transmit_2 &\stackrel{\text{def}}{=} c.Transmitting_2 + acq_2.\overline{begin_2}.\overline{end_2}.\overline{et}.Transmitter_2 \end{aligned}$$

A seguito di un'operazione di invio (*send*), il processo trasmettitore avvia la trasmissione sul canale (*bt*), entrando nel periodo di *contention*. Durante questo periodo, può avvenire la collisione con l'altra stazione (*c*), oppure, nel caso il segnale riesca a propagarsi con successo alla stazione di destinazione, il trasmettitore acquisisce l'uso esclusivo del canale (*acq*). La trasmissione termina con il rilascio del canale trasmissivo (*et*).

I processi ricevitore sono mostrati graficamente in Figura 5, e si possono invece codificare come segue:

$$Receiver_1 \stackrel{\text{def}}{=} br_1.(bad.Receiver_1 + er.\overline{rec_1}.Receiver_1$$

$$Receiver_2 \stackrel{\text{def}}{=} br_2.(bad.Receiver_2 + er.\overline{rec_2}.Receiver_2$$

La presenza di un messaggio in ricezione viene segnalata al processo ricevitore con l'input sul canale *br*, al quale segue l'inizio della ricezione. Durante la ricezione, il sistema può ancora trovarsi in *contention* e possono avvenire collisioni. In tal caso, viene segnalato al ricevitore di scartare la porzione di messaggio rifevuta fino a quel momento (*bad*). Nel caso non vi siano collisioni, la ricezione procede correttamente e termina (*er*). A questo punto, è possibile comunicare al client il messaggio appena ricevuto (*rec.*)

Il processo del *medium* è rappresentato graficamente in Figura 6, e ha invece la seguente codifica:

$$\begin{aligned} Medium &\stackrel{\text{def}}{=} bt_1.Contention_1 + bt_2.Contention_2 \\ Contention_1 &\stackrel{\text{def}}{=} bt_2.Collision + \overline{br_2}.(bt_2.\overline{bad}.Collision + DoMedium_1) \\ Contention_2 &\stackrel{\text{def}}{=} bt_1.Collision + \overline{br_1}.(bt_1.\overline{bad}.Collision + DoMedium_2) \\ DoMedium_1 &\stackrel{\text{def}}{=} \overline{acq_1}.et.\overline{er}.Medium \\ DoMedium_2 &\stackrel{\text{def}}{=} \overline{acq_2}.et.\overline{er}.Medium \\ Collision &\stackrel{\text{def}}{=} \overline{c_1}.\overline{c_2}.Medium \end{aligned}$$

Entrambe le stazioni possono avviare la trasmissione sul medium durante il periodo di *contention*. Nel caso di trasmissione simultanea, il *medium* comunica alle stazioni coinvolte l'avvenuta collisione. Nel caso fosse stata avviata una ricezione, viene segnalato anche il ricevitore (*bad.*) A seguito dell'acquisizione del canale da una stazione (*acq*), la trasmissione procede fino alla fine di questa (*et* e *er*.)

I processi corrispondenti ai due *Medium Access Controller* sono quindi:

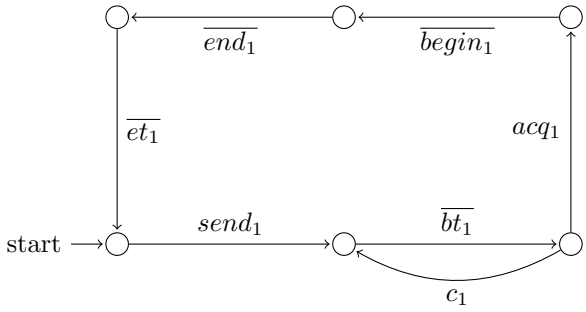

Figura 4: Rappresentazione del processo trasmettitore

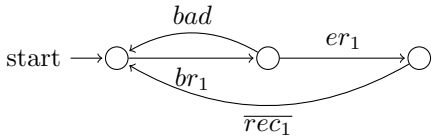

Figura 5: Rappresentazione del processo ricevitore

$$MAC_1 \stackrel{\text{def}}{=} Transmitter_1 | Receiver_1$$
$$MAC_2 \stackrel{\text{def}}{=} Transmitter_2 | Receiver_2$$

Sia l'insieme $L \subseteq \mathcal{L}$ tale che:

$$L = \{bt_1, bt_2, et, br, er, c, acq_1, acq_2, bad\}$$

il sistema complessivo *Protocol* è definito in CCS come segue:

$$Protocol \stackrel{\text{def}}{=} (MAC_1 | MAC_2 | Medium) \setminus L$$

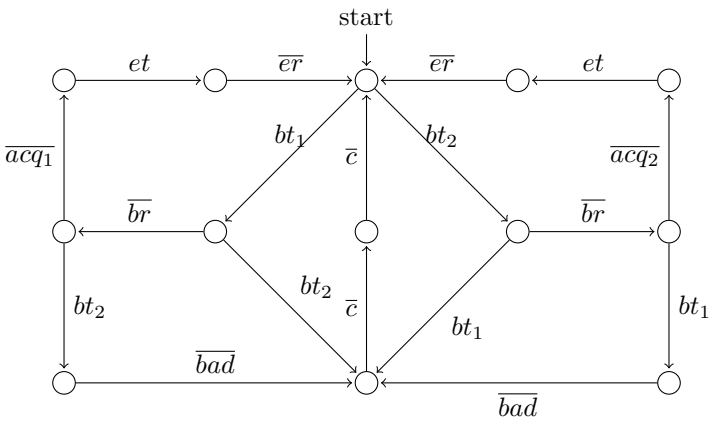

Figura 6: Rappresentazione del processo *Medium*

3 Verifica formale

Ci si è serviti dell'*Edinburg Concurrency Workbench*, un tool per la verifica di equivalenze tra processi e proprietà espresse in una logica che ha diretta corrispondenza con il μ-calcolo.

3.1 *Equivalence checking*

Prima di verificare l'aderenza del protocollo modellato alla specifica, e quindi verificare l'equivalenza tra il comportamento dei due sistemi, è necessario stabilire di quale concetto di equivalenza tra processi ci si voglia servire.

3.1.1 Bisimilarità forte

Data la sensibilità della bisimilarità forte alle transizioni τ, si può immaginare che tale equivalenza non sia adatta allo scopo, osservando in particolare che i due sistemi non soddisfano le stesse proprietà in logica di Hennessy-Milner. Ad esempio, essendo $F = \langle send_1 \rangle \langle \tau \rangle \langle \tau \rangle$, è immediato verificare che

$$Protocol \models F, \qquad Spec \not\models F$$

Per il teorema di Hennessy-Milner, i due processi non sono fortemente bisimili. A conferma di questo fatto, il risultato negativo del comando `strongeq` (*strong equivalence*) del *Concurrency Workbench* sui due processi.

3.1.2 Bisimilarità debole

I due processi sono provati essere differenti nella loro possibilità di effettuare transizioni interne. Ai fini di questa verifica, ad ogni modo, interessa solamente confrontare rispetto al loro comportamento osservabile. La bisimilarità debole soddisfa questi requisiti, per la sua caratteristica di astrarre dalle transizioni interne. I processi *Protocol* e *Spec* risultano infatti debolmente bisimili, come è possibile verificare con il comando `eq` del *Concurrency Workbench*.

Ad ulteriore prova della bisimilarità debole, si può verificare, con il comando `dfweak`, che i due processi soddisfano le stesse formule in logica di Hennessy-Milner debole, il che implica la bisimilarità debole: questo comando evidenzia le formule in logica di Hennessy-Milner debole che differenziano i due processi, e in questo caso non ne vengono trovate.

Divergenza Si noti che il processo *Protocol* presenta la possibilità di effettuare un numero infinito di transizioni τ, e risulta quindi potenzialmente divergente. Nonostante questo, il protocollo modellato risulta debolmente bisimile alla specifica, che invece non contiene alcun ciclo di transizioni τ. Questo è possibile grazie ad un'importante proprietà della bisimilarità debole nota come *fair abstraction from divergence*: la bisimilarità debole assume che se un processo è in grado di uscire da un ciclo di transizioni interne, prima o poi lo farà.

Tale proprietà della bisimulazione debole risulta in questo caso corretta, e non altera il risultato della verifica: permette di considerare un modello del protocollo CSMA/CD che astragga per semplificazione da quelle tecniche, come l'applicazione di algoritmi di backoff, che nel protocollo reale fanno di modo di evitare potenziali cicli di collisioni infinite. La bisimilarità debole assume, correttamente, che questo avvenga, permettendo di ragionare su dettagli del protocollo ritenuti più importanti.

Ad ogni modo, ci si aspetta che tale differenza possa emergere in qualche modo con altri tipi di relazioni. Il comando `diveq` (equivalenza osservazionale rispetto alla divergenza) del *Workbench* sui processi *Protocol* e *Spec* conferma la differenza tra i due processi data dalla presenza di cicli τ potenzialmente infiniti del protocollo. Una ulteriore prova di questa differenza si ha verificando che i due processi, come esposto in Sezione 3.2.2, non soddisfano le stesse formule in logica di Hennessy-Milner con ricorsione, utilizzata per la definizione di proprietà temporali dei processi.

3.1.3 Congruenza osservazionale

Come si può verificare dalla rappresentazione in CCS dei processi, e con il comando `stable` del *Concurrency Workbench*, i due processi sono stabili (non contengono transizioni τ iniziali), e sono pertanto legati dalla congruenza osservazionale. Questo può essere automaticamente verificato con il comando `cong` del *Workbench*.

3.1.4 Weak trace equivalence

Seppur meno forte dell'equivalenza osservazionale, la *weak trace equivalence* sarebbe comunque sufficiente a dimostrare in questo caso la corretta implementazione della specifica da parte del protocollo, verificando sostanzialmente che le sequenze di operazioni visibili eseguibili dai dei processi siano uguali.

L'equivalenza è verificata con il comando `mayeq`, che nel caso dei processi *Protocol* e *Spec* restituisce `true`. Si noti che questo risultato poteva essere direttamente inferito dalla bisimilarità debole, che implica la *weak trace equivalence*.

3.2 *Property checking*

È possibile modellare le proprietà attese del protocollo con formule in logica di Hennessy-Milner, e verificarne il soddisfacimento grazie al tool di verifica utilizzato. Di seguito vengono esposte alcune proprietà; la loro traduzione per il *Concurrency Workbench* può essere trovata nel file `csma.cwb`.

3.2.1 Deadlock

Un processo è in *deadlock* se raggiunge uno stato nel quale non è possibile procedere in alcun modo con la computazione. Questa condizione, nel contesto del CCS, è quella di un processo che non è in grado di effettuare alcuna transizione:

$$Deadlock = [\mathsf{Act}]ff$$

La possibilità che possa verificarsi un *deadlock* si può esprimere con l'operatore temporale $Pos(F)$. Negando la formula, si ottiene una proprietà soddisfatta da tutti i processi privi di *deadlock*:

$$NoDeadlock = \neg Pos(Deadlock)$$

Questa proprietà può essere verificata facilmente nel *Concurrency Workbench* con il comando `deadlocks`.

3.2.2 Cicli τ

Si vuole verificare la proprietà di divergenza dei processi. Come detto in precedenza, ci si aspetta che il processo *Protocol* possieda questa proprietà. $TauCycle$ può essere definita ricorsivamente come:

$$X \stackrel{\max}{=} \langle\tau\rangle X;$$

Il comando `checkprop` del *Concurrency Workbench* conferma che

$$Protocol \vDash Pos(Livelock)$$
$$Spec \nvDash Pos(Livelock)$$

3.2.3 Mutua esclusione

Per accertare la mutua esclusione nell'accesso al *medium* condiviso, è sufficiente verificare che

$$P \vDash Inv([end_{12}]ff \vee [end_{21}]ff)$$

In particolare si vuole che, in qualsiasi stato della computazione, almeno per uno dei due processi coinvolti non sia possibile uscire dalla trasmissione sul *medium*. Questo si ha solamente se è garantito l'accesso esclusivo.

3.2.4 Ricezione in seguito a trasmissione

Si vuole verificare che, a seguito di una trasmissione terminata con successo, il messaggio venga effettivamente ricevuto dalla stazione di destinazione. Tale proprietà si può esprimere come segue, utilizzando gli operatori temporali $Even(F)$ e $Inv(F)$:

$$EvenReceive_1 = Inv([\overline{end_1}]\, Even(\langle\overline{rec_2}tt\rangle))$$

Analogamente,

$$EvenReceive_2 = Inv([\overline{end_2}]\, Even(\langle\overline{rec_1}tt\rangle))$$

Quindi, si vuole che

$$Protocol \vDash EvenReceive_1 \wedge EvenReceive_2$$

3.2.5 Ritrasmissione

Le operazioni rec_1 e rec_2 sono atte a comunicare al client della stazione ricevete la ricezione di un messaggio. Si tratta quindi di uno scambio tra il MAC di destinazione e il client, e avviene indipendentemente dalle trasmissioni sul *medium*. Deve essere quindi possibile, a seguito di una trasmissione terminata, avviare la trasmissione di un altro messaggio, anche se quello precedente non è ancora stato ricevuto:

$$\begin{aligned} CanResend_1 &= Inv([\overline{end_1}]\langle\langle send_1\rangle\rangle tt) \\ CanResend_2 &= Inv([\overline{end_2}]\langle\langle send_2\rangle\rangle tt) \\ CanResend &= CanResend_1 \wedge CanResend_2 \end{aligned}$$

Si vuole quindi che:

$$Protocol \vDash CanResend$$

3.2.6 Assenza di starvation

Si vuole verificare che sia sempre possibile, per una stazione, acquisire il canale trasmissivo a seguito di un'azione *send*. In altre parole, si vuole assicurare che non sia possibile il verificarsi di uno scenario dove una stazione riesce costantemente ad acquisire il canale trasmissivo e un'altra non ci riesce mai. Si vuole quindi che sia soddisfatta la proprietà

$$\begin{aligned} NoStarvation_1 &= Inv([send_1]\; Pos(\langle\overline{begin_1}\rangle tt)) \\ NoStarvation_2 &= Inv([send_2]\; Pos(\langle\overline{begin_2}\rangle tt)) \\ NoStarvation &= NoStarvation_1 \wedge NoStarvation_2 \end{aligned}$$