Corso di Linguaggi e Modelli Computazionali LM GreatSPN e MC4CSL^{TA}

Chiara Contoli

Corso di Laurea in Ingengneria e Scienze Informatiche
Alma Mater Studiorum – University of Bologna
via Venezia 52, 47023 Cesena, Italy
chiara.contoli@studio.unibo.it

1 Introduzione

Lo scopo di questo progetto è sperimentare l'utilizzo di due tool:

1. GreatSPN: consente la modellazione di sistemi tramite il formalismo del linguaggio grafico delle reti di Petri;
2. MC4CSL^{TA}: è un model checker probabilistico per la logica stocastica CSLTA che esegue il modeling formale e l'analisi quantitativa di sistemi , specificati come catene di Markov Tempo Continue (CTMCs).

2 GreatSPN

Questo tool nasce dalla collaborazione tra il Dipartimento di Elettronica del Politecnico di Torino e il dipartimento di informatica dell'Università di Torino. Inizialmente viene sviluppato con l'intento di fornire uno strumento per la specifica e la valutazione di performance per le architetture di computer. Nel corso degli anni si è evoluto introducendo una interfaccia grafica, portabilità, modularità e una maggiore efficienza nei moduli per l'analisi dei sistemi. In particolare, l'attenzione si è focalizzata sul rendere il tool più user friendly e sul miglioramento degli algoritmi risolutivi come *steady-state* e *transient* applicati alla catena di Markov sottostante, ma anche nell'introduzione di nuovi algoritmi per l'analisi di modelli che contengono transizioni di tipo diverso. Un'ulteriore miglioria nella capacità di validazione è stata ottenuta introducendo algoritmi per la computazione delle *Place* e *Transition Invariants*, che hanno consentito un più facile controllo sulle condizioni strutturali necessarie o sufficienti per le proprietà di *boundness* e di *ergodicità* prima di una esaustiva enumerazione dello spazio degli stati. Non va infatti dimenticato il problema della crescita esponenziale dello spazio degli stati da esplorare. Un altro aspetto importante dell'evoluzione del tool è che la versione attuale è interamente scritta in C , che ha consentito una maggiore portabilità e una maggiore efficienza nell'implementazione degli algoritmi.
Nei successivi paragrafi verranno illustrate le funzionalità messe a disposizione che saranno poi applicate ad alcuni casi di studio.

2.1 Reti di Petri

Le reti di Petri (PNs) sono uno strumento grafico e matematico per la modellazione e la descrizione di sistemi concorrenti. Formalmente una PN è descrivibile con una tupla di cinque elementi: *(P, T, I, O, H)* . Una PN è un grafo composto da:

- un insieme di nodi P detti *places*;
- un insieme di "nodi" T detti *transitions* (o *transizioni*);
- un insieme di archi entranti $I: T \rightarrow Bag(P)$ da places a transizioni (esprimono le precondizioni);
- un insieme di archi uscenti $O: T \rightarrow Bag(P)$ da transizioni a places (esprimono gli effetti);
- un insieme di archi $H: T \rightarrow Bag(P)$ detti *inibitori* da places a transizioni (esprimono inibizione);

Una condizione di correttezza importante è: dato un place P e una transizione T, c'è uno o nessun arco inibitore che va da P a T (ovvero, c'è al più un arco inibitore). Gli archi inibitori in alcune formulazioni di base non sono presenti, in quanto considerati una estensione.
Le PNs hanno inoltre una funzione che definisce la nozione di stato (distribuito) del sistema che viene chiamato *marking*. Il marking rappresenta quindi lo stato del sistema in un dato momento. La funzione è così definita $M: P \rightarrow \{0, 1, 2, ...\}$, cioè una funzione dai places ai numeri naturali. Il sistema è quindi descritto dalla struttura della PN e dal *marking iniziale* M_0 . M_0 rappresenta la distribuzione iniziale dei *tokens* all'interno dei places. Lo stato è qualcosa che è distribuito su tutti i places.
I places sono rappresentati come cerchi, le transizioni come barre, gli archi come frecce pesate e gli archi inibitori come "frecce" la cui testa è un cerchio. I tokens sono rappresentati come punti neri e il marking è definito inscrivendo i tokens nei places. L'evoluzione del sistema è descritto dallo *scatto* (o *firing*) delle transizioni. Una transizione per poter scattare deve essere *abilitata*, ovvero devono essere soddisfatte le precondizioni (il numero di token presenti nei place di input deve essere maggiore o uguale al peso degli archi di input; il peso, salvo diversa specifica, di defautl è uno). In un certo istante possono esserci più transizioni abilitate, ma scatta una sola transizione per volta. Lo scatto di una transizione determina la rimozione dei tokens dai places di input e l'immissione dei tokens nei places di output.
Partendo dal marking iniziale M_0 è possibile computare l'insieme di tutti i marking raggiungibili, il *Reachability Set* (RS) del modello. Il RS non contiene informazioni relative alla sequenza di transizioni che ha portato al raggiungimento di un certo marking. Questa informazioni è invece contenuta nel *Reachability Graph* (RG), i cui nodi sono etichettati con il marking raggiungibile e gli archi con la transizione che il sistema deve far scattare per andare da uno stato all'altro.

2.2 Reti di Petri Stocastiche

Una estensione delle reti di Petri sono le reti di Petri Stocastiche (SPNs); anch'esse sono un formalismo basato su grafi per la modellazione di sistemi in cui si ha la necessità di modellare aspetti che coinvolgono la variabile tempo. Si differenziano dalla formulazione classica poiché questa non ha alcuna nozione di tempo e quindi venivano utilizzate principalmente per l'analisi di proprietà logiche del sistema. Con l'introduzione di aspetti temporali è invece possibile introdurre un'analisi quantitativa delle performance. Questo si traduce nell'associare un ritardo alla transizione. Le SPN sono PN in cui il ritardo nello scatto delle transizioni sono variabili aleatorie con distribuzione esponenziale negativa: ogni transizione t_i è associata ad un ritardo di scatto random la cui funzione di densità di probabilità è un esponenziale negativo con rate λ_i . Quindi, formalmente, una SPN ha un ulteriore funzione *W:* $T \rightarrow R^+$ tale che il ritardo associato alla transizione t è una variabile casuale, distribuita come un esponenziale negativo, con rate *W(t)*. Si aggiunge quindi un rate alle transizioni.
Dal punto di vista della rappresentazione grafica la transizione così definita è una barra spessa e bianca. La semantica delle SPNs è rappresentata dal *race model* (o modello delle *corse*); quando un marking simultaneamente abilita più transizioni (in conflitto o concorrenti), si assume che tutte le attività associate a queste transizioni eseguano in parallelo, quindi il prossimo marking è dato dalla transizione il cui ritardo di scatto è minimo, cioè la transizione che vince la corsa. Lo scatto della transizione che vince implica che le attività associate ad essa sono completate. Anche in questo caso è ancora valida la definizione di RS e RG ma in questo caso gli archi del RG sono etichettate, oltre che con il nome della transizione, anche con i *rates*. Un aspetto importante delle SPNs è che data la proprietà di *assenza di memoria* derivata dalla distribuzione dei ritardi associati alle transizioni, le SPNs sono isomorfe alle catene di Markov tempo continue (CTMCs) in cui:

1. gli stati della CTMC sono in corrispondenza uno a uno con i marking della SPN;
2. la frequenza di transizione dallo stato i (corrispondente al marking M_i) allo stato j (M_j) della CTMC è uguale alla somma dei rate delle transizioni che connettono i corrispondenti marking nel grafo di raggiungibilità della rete.

La conversione da una SPN alla CTMC è la seguente: il RS della SPN viene generato e le frequenze di scatto delle transizioni abilitate sono usate per costruire la matrice $\mathbf{Q}$ dei rate della CTMC. Se la CTCM è ergodica, è possibile computare la distribuzione delle probabilità di stato (steady-state) dei marking risolvendo l'equazione matriciale $\pi\mathbf{Q} = 0$ con l'ulteriore vincolo che la sommatoria delle probabilità di stato deve essere uguale a uno, ovvero $\Sigma_i \pi_i = 1$ in cui π è il vettore delle probabilità di stato. Dal vettore delle probabilità di stato è possibile ottenere informazioni quantitative della SPN. Le difficoltà potrebbero sorgere a causa della complessità computazione degli algoritmi per questo tipo di soluzione, quando il numero dei marking raggiungibili cresce. Questo è il problema principale associato all'uso delle SPN.

2.3 Reti di Petri generalizzate

Le Generalised Stochastic Petri Nets (GSPNs) sono il formalismo che introdusse gli archi inibitori, e sono SPNs in cui esistono transizioni che scattano in tempo nullo. Si distinguono infatti due tipi di transizioni:

1. *immediate* (o immediate): non richiedono alcun tempo per completare, scattano in tempo zero;
2. *timed* (o temporizzate): hanno tempo di scatto che è un esponenziale negativo dopo che sono state abilitate;

Dal punto di vista della rappresentazione grafica le transizioni immediate sono rappresentate come barre nere sottili e sono solitamente usate per modellare aspetti del sistema che non hanno un tempo fisico associato con esso, oppure aspetti che hanno un tempo talmente piccolo paragonato al tempo delle attività del sistema. Nel caso delle transizioni immediate il valore *W(t)* rappresenta il peso associato alle transizioni. Quando due o più transizioni immediate sono in conflitto, la selezione di quella che scatta per prima va in base al peso, normalizzato in maniera tale per cui si ottiene funzione di distribuzione discreta di probabilità.
La presenza delle transizioni immediate determina due tipi di marking nel RS che si distinguono in:

- *vanishing*: marking in cui almeno una transizione immediata è abilitata, quindi in questo stato il tempo speso dal sistema è nullo;
- *tangibile*: marking in cui non vi è alcuna transizione immediata abilitata, quindi il sistema in questo stato trascorre del tempo;

L'esecuzione di un modello GSPN non è identico ad una funzione di un processo di Markov, a causa delle discontinuità temporali introdotte dai vanishing marking. È tutta via possibile rimuovere questi marking dall'analisi, dal momento che non contribuiscono alla misurazione del comportamento del modello. Le performance di una GSPN possono quindi essere analizzate esaminando solamente l'evoluzione dei tangibile marking. É stato dimostrato che c'è una corrispondenza tra i modelli GSPN e le CTMC. Gli indici di performance, come il numero medio di tokens in un place e il throughput delle transizioni possono essere associati con un modello GSPN.Sono computati sia a partire dalle probabilità di stato che dalle probabilità di transito della CTMC associata. Un'altra estensione/peculiarità introdotta è la possibilità di definire marking dipendenti dai rate: il rate di una transizione è quindi funzione dello stato del sistema.

2.4 Reti di Petri Colorate

Le Stochastic Well Formed Nets (SWN) sono la versione colorata delle SPNs che consentono di costruire rappresentazioni più compatte e parametriche di sistemi simmetrici, "fondendo" sotto reti simili. In questo modo è possibile costruire rappresentazioni molto concise di sistemi che avrebbero richiesto versioni non

colorate molto più grandi. La fusione di reti similari richiede una notazione addizionale per distinguere i tokens che finiscono per essere messi nello stesso place. I tokens non sono più indistinguibili, ma appartengono ad un place cui verrà assegnato un colore di dominio.

2.5 Modellare con GreatSPN

GreatSPN è provvisto di un'interfaccia grafica che consente di editare il modello secondo il formalismo grafico delle reti di Petri. Consente di modellare :

- reti di Petri nella formulazione base;
- reti di Petri generalizzate;
- reti di Petri stocastiche;
- reti di Petri colorate;

Oltre alla modellazione sono presenti diversi moduli che consentono l'analisi sia qualitativa che quantitativa del sistema. Tali funzionalità saranno illustrate nel seguito.
L'editor di modellazione si presenta come mostrato in figura 1 . Dalla menu

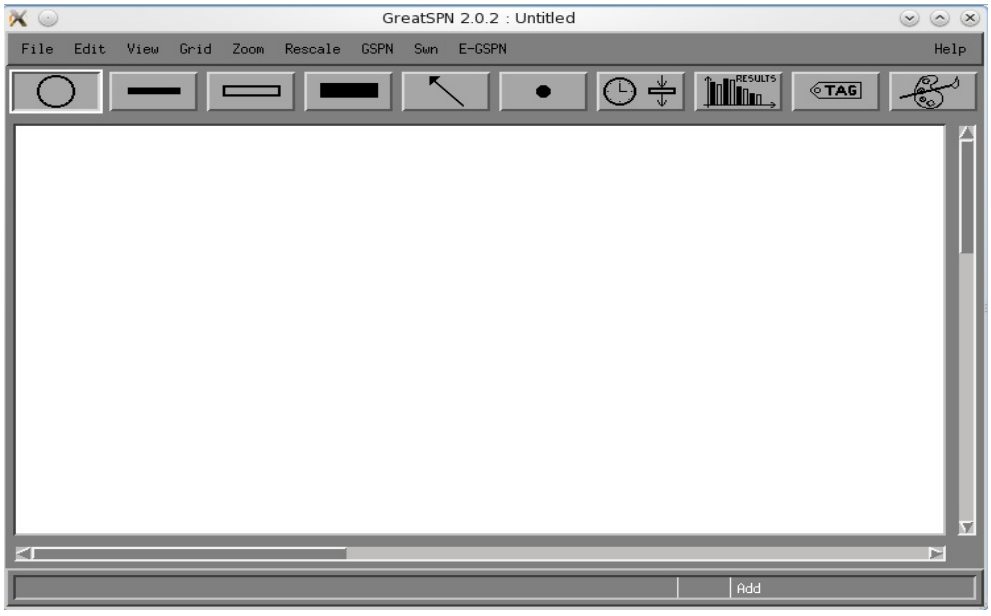


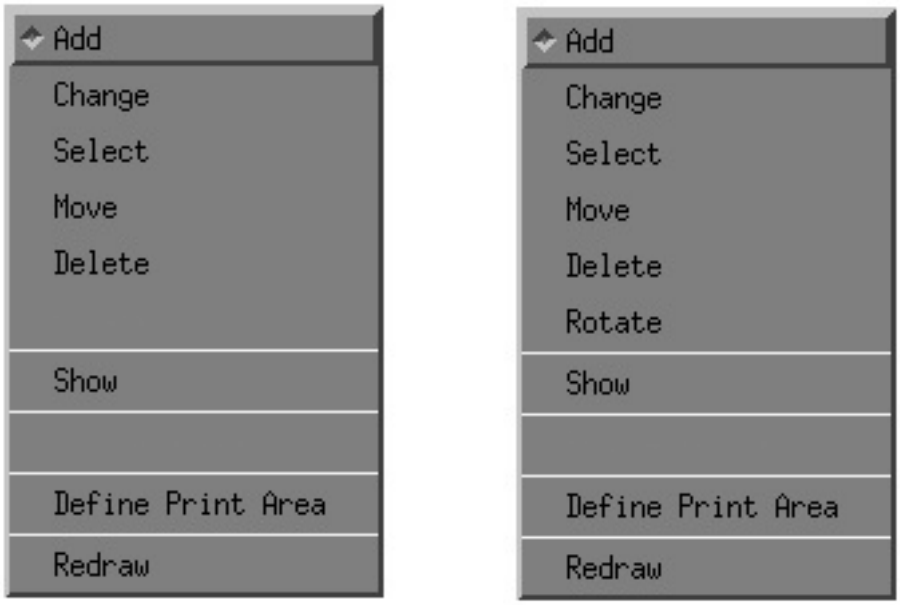
Figura 1. GreatSPN editor

bar è possibile accedere a tutte le funzionalità del tool via interfaccia grafica, ma come sarà illustrato nel seguito, queste sono disponibili anche da linea di

comando. Dalla object bar sottostante invece è possibile gestire la creazione e la configurazione dei parametri della PN. L'azione di default è l'aggiunta, indicata con *Add* nel menu che è visibile facendo clic in una qualsiasi area del editor con il tasto destro del mouse. Partendo dal primo a sinistra si ha:

- *place*;
- *transizione immediata*;
- *transizione temporizzata* con distribuzione esponenziale negativa;
- *transizione temporizzata* con distribuzione deterministica;
- *arco*;
- *token*;
- *definizione parametri per le transizioni temporizzate*;
- *risultati analisi*;
- *etichette*;
- *reti colorate*;

Dal menu di cui sopra si selezionano i componenti e si possono gestire le etichette (sia dei place che delle transizioni) ai quali è possibile associare nomi e valori che nel seguito possono essere assegnati ai place e transition presenti. Il menu dal quale è possibile scegliere il tipo di azione da eseguire sul componente attualmente selezionato è visibile in figura 2 . In figura 2(a) è sono visibili



(a) `menu1` (b) `menu2`

Figura 2. Menu

le azioni che possono essere fatte sul componente place, mentre in figura 2(b)

quelle sul componente transizione. Dall'azione *Change* è possibile impostare le caratteristiche dei componenti.
La figura 3 rappresenta il modello di un sistema a coda di tipo *M/M/1* con due classi di utenza in cui una è più prioritaria dell'altra. Questa rete è una GSPN poiché contiene sia transizioni immediate che temporizzate. Come anticipato l'azione di default è l'aggiunta (*Add*), quindi "disegnare" il sistema è sufficiente selezionare i componenti e disporli come più si ritiene opportuno. Per l'aggiunta degli archi è necessario selezionare il componente e poi fare clic da un place ad una transizione (per le precondizioni) e poi da una transizione ad un place (per gli effetti). Per modellare un arco inibitore è necessario fare clic con il tasto centrale del mouse sulla transizione che si vuole inibire e poi fare clic sul place. In questo modo, fino a che vi sarà uno o più tokens nel place selezionato la transizione non potrà essere abilitata, e quindi non potrà scattare. Mano a mano che vengono aggiunti i componenti questi sono automaticamente

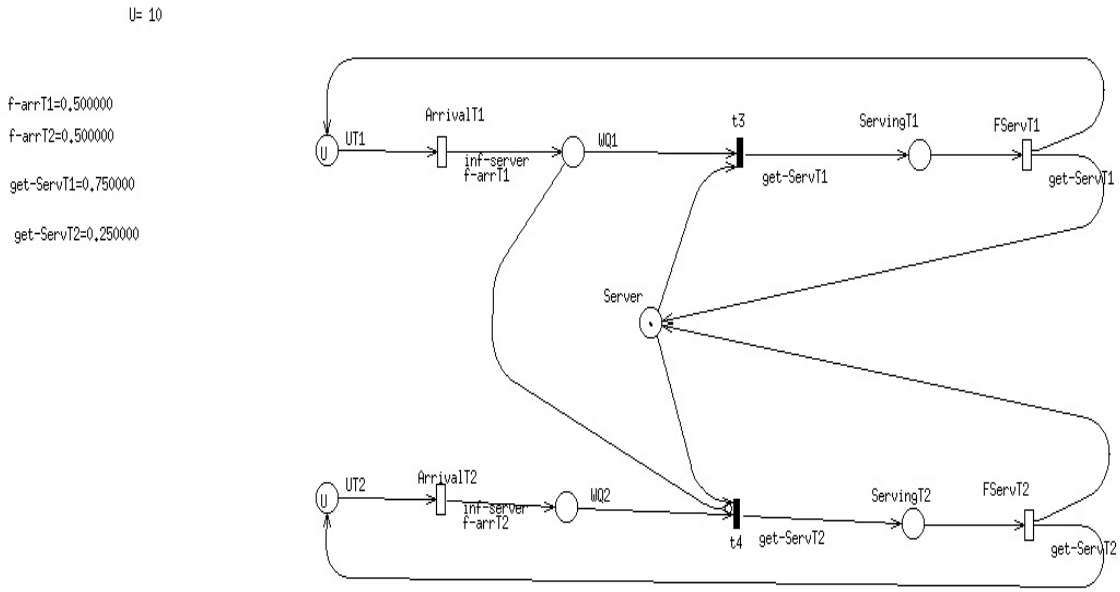


Figura 3. Sistema a coda M/M/1

etichettati dal tool, ma è possibile cambiarle dalla object bar scegliendo l'opzione *TAG* e scegliendo l'opzione *Change* dal menu a tendina. Per quanto riguarda le transizioni, di default il peso associato alle transizioni immediate, o il rate associato alle transizioni temporizzate è *1.0* .Per modificare tale valore vi sono due possibilità:

– selezionare il componente transition dalla object bar, usare l'azione Change

dal menu a tendina e selezionare la transizione interessata, andando a modificare con un valore numerico il campo *rate or rate parameter* sulla finestra che appare;

– selezionare dalla object bar il componente "transizioni con clock", l'azione Add e fare clic su un punto dell'editor; nella finestra che appare si deve specificare il nome che verrà usato come *Label*, mentre nel campo *Rate* si deve specificare il valore numerico da assegnare alla frequenza. Un esempio di questa modalità è visibile in figura 3 in alto a sinistra dove vi sono le label assegnate alle frequenze di transizione e i rispettivi rate. Le label *f-arrT1*, *f-arrT2*, *get-ServT1* e *get-ServT2* sono quelle che vengono chiamate rate parameter. Per essere assegnate alle transizioni è necessario selezionare dalla object bar il simbolo della transizione che si desidera modificare e poi selezionare la transizione sulla rete, scegliendo l'opzione Change. Nel campo *Rate or Rate parameter* va specificato il nome della label definita precedentemente.

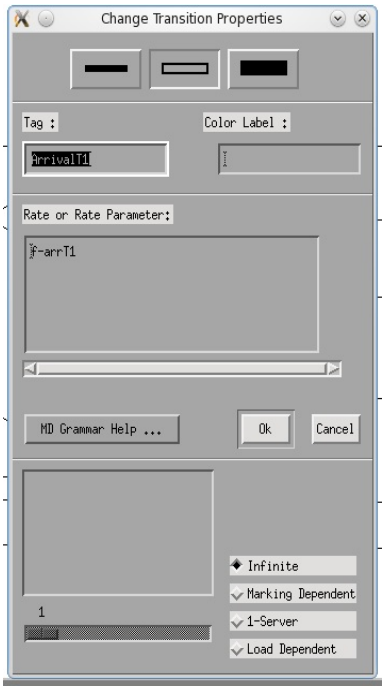


Figura 4. Proprietà transizioni temporizzate

Quando si assegnano i rate alle transizioni temporizzate c'è anche la possibilità di stabilire l'abilitazione in base alle dipendenze. In figura 4 si possono vedere le semantiche associabili alle transizioni:

- *Infinite*: non appena uno o più tokens entrano nel place, questo viene considerato e genera anch'esso un firing delay che entra in competizione con gli altri per lo scatto. É l'opzione attiva di default.
- *Marking dependent*: lo scatto di questa transizione è funzione del marking di un insieme di place della rete.
- *1-Server*: i token sono processati in serie, un token arrivato non viene processato per competere con gli altri.
- *Load dependant*: è un caso particolare del marking dependent.

Queste sono anche dette semantiche dei servers.
In maniera analoga si può definire anche il marking iniziale: selezionando dall'object bar il componente token, selezionando l'azione Add e facendo clic in un punto qualunque dell'editor, si impostano sulla finestra a comparsa la label (ad esempio *U*, come si vede in figura 3) e il numero di token. Il marking parametrico così creato può essere assegnato ad un qualunque place scegliendo al posto di Add l'azione Change, e selezionando il place al quale si vuole assegnare il marking. Dalla finestra a comparsa nella casella *Marking* va specificato il nome della label del marking parametrico creato in precedenza. In tal modo, il place selezionato avrà un numero di token pari al numero specificato in precedenza. L'alternativa è usare direttamente l'azione Change e inserire nella casella *Marking* il valore numerico.

2.6 Analisi con GreatSPN

La fase di modellazione genera due file:

- *<nomerete>.net*
- *<nomerete>.def*

che insieme costituiscono la descrizione della rete in formato ASCII. In base alle analisi che vengono condotte vengono via via generati altri file. GreatSPN consente l'analisi di:

place e transition invariants: consentono un'analisi delle condizioni necessarie o sufficienti per avere *ergodicità* e condizione di *boundness* prima di una esaustiva enumerazione dello spazio degli stati. Si ricorda che con P-invariants si intende un insieme di posti tale per cui la somma dei token rimane costante in tutti i marking, mentre i T-invariants sono sequenze di scatti che riportano la rete allo stato iniziale. Per poter eseguire l'analisi delle invarianti, selezionare dalla menu bar *GSPN→Struct→P Invariants* oppure *GSPN→Struct→T Invariants* . In entrambi i casi comparirà una finestra dalla quale, premendo il pulsante *Start* è possibile eseguire l'analisi. L'esito è visibile sulla console grafica, come mostrato in figura 5 e 6. A fronte di queste analisi vengono anche generati due file, uno con estensione *.pinv*, l'altro con estensione *.tin* .
structural boundness: una rete è detta *k-bounded* se esiste un valore intero positivo k tale per cui il numero di token in ogni place della rete è al

Figura 5. Analisi P-invariants

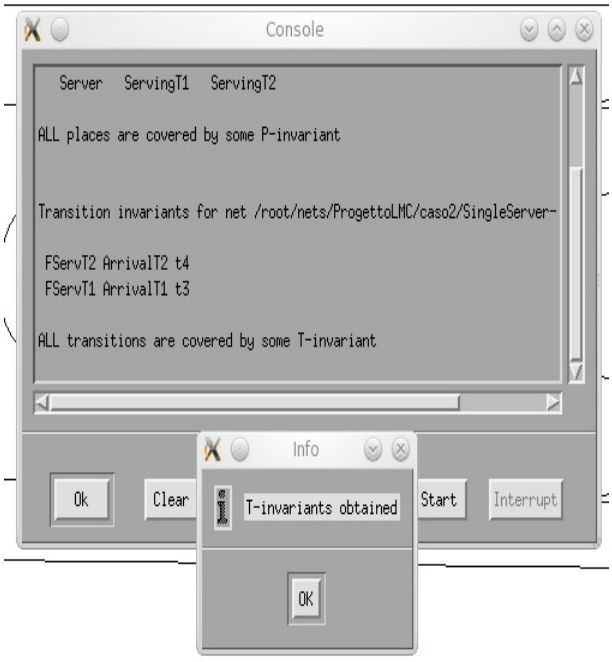


Figura 6. Analisi T-invariants

più uguale a k. Al fine di eseguire l'analisi da GUI è possibile selezionare *GSPN→Struct→Structural Boundness*, il cui risultato è visibile in figura 7 .

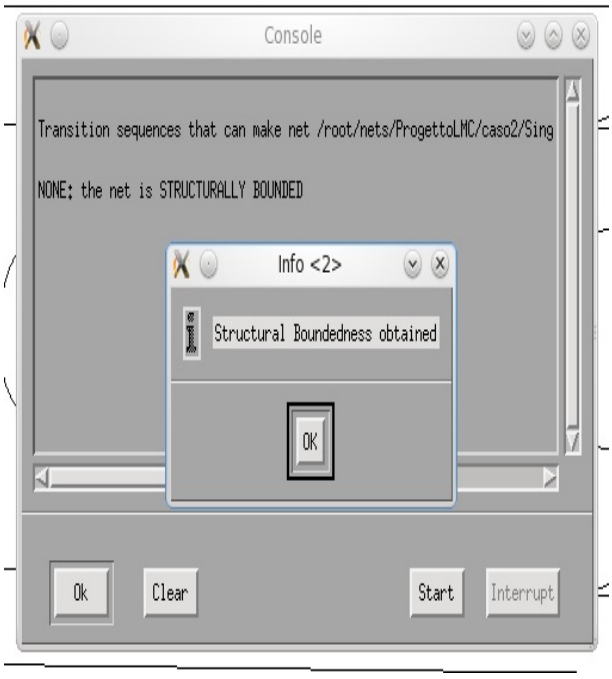


Figura 7. Boundness

Reachability graph: la generazione del grafo di raggiungibilità può essere fatto sia tramite GUI, selezionando *GSPN→Solve→Compute TRG*, oppure tramite linea di comando, eseguendo da terminale il comando *newRG netdirectory/netname* , dove *netdirectory* è il percorso in cui si trova la rete e *netname* e il nome del file che contiene la descrizione della rete (senza l'estensione *.net*). Per ottenere i risultati visibili in figura 8 è stato dato il comando *newRG nets/ProgettoLMC/caso2/SingleServer-MultiUser-Prio*. Risultato analogo si ottiene da GUI come mostrato in figura 9 . Confrontando gli output si può vedere che in quello ottenuto lanciando il comando da GUI si vedono alcune proprietà della rete, e anche il numero di marking di tipo vanishing e di tipo tangible, a differenza di quello dato da terminale in cui si vedono solo il numero di marking tangible e non vanishing. Il TRG può essere visualizzato sia dopo la creazione da GUI, anche se in una versione non proprio user friendly, oppure può essere visualizzato tramite il comando *showRG netdirectory/netname -t*, che mostra un output più user friendly e

che per comodità qui ne viene riportato una parte in figura 10.
É anche possibile visualizzare il TRS tramite il comando *showRG netdirectory/netname -s* e se ne riporta una parte in figura 11. Per il calcolo del TRG l'algoritmo comincia mettendo il marking iniziale nel Reachability Set , poi tutte le transizioni abilitate nel nuovo marking trovato vengono fatte scattare. Lo scatto delle transizioni temporizzate proseguono usando una politica di tipo breadth-first, mentre i percorsi delle transizioni immediate seguono una politica di tipo depth-first fino a che non viene trovato un marking tangibile. Le proprietà verificate visibili in figura 9 possono essere ottunete anche tramite il comando *checkRG netdirectory/netname*.

```
[root@localhost ~]# newRG nets/ProgettoLMC/caso2/SingleServer-MultiUser-Prio

Place invariants for net nets/ProgettoLMC/caso2/SingleServer-MultiUser-Prio:

   UT1   WQ1   ServingT1
   UT2   WQ2   ServingT2
   Server   ServingT1   ServingT2

ALL places are covered by some P-invariant


Start Reachability Graph construction

real    0m0.006s
user    0m0.000s
sys     0m0.003s
     50
    100
    150
    200
    221 Tangible markings

[root@localhost ~]#
```

Figura 8. Computazione TRG da terminale

Embedded Mark Chain (EMC): è la possibilità di generare la catena di Markov associata alla rete. Per la costruzione dell'Embedded Markov Chain viene usato il comando *newMT netdirectory/netname*, oppure da GUI con *GSPN→Solve→Compute EMC*. Il risultato è visibile in figura 12, mentre una porzione dell'EMC è riportata in figura 13 e 14. La catena di Markov viene quindi ottenuta molto agevolmente, anche per grandi dimensioni.
Stead-state solution: computa la soluzione a regime della CTMC associata alla rete, ovvero calcola il vettore delle probabilità di stato. Questo può esse-

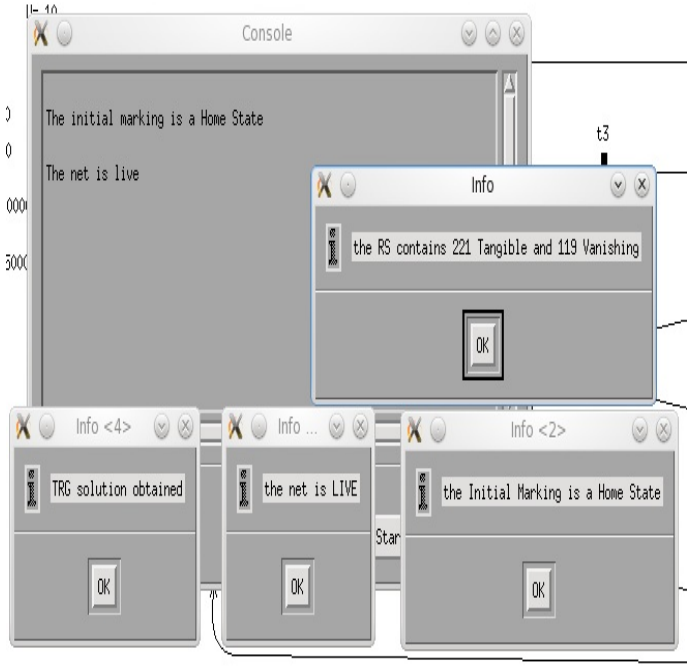


Figura 9. Computazione TRG da GUI

```
*** Start of TRG ***

From #1 (2 timed trans)   [ 10 @ UT1, 10 @ UT2, Server ]
  -Timed-> ArrivalT2  start vanishing paths:
        -VP-> t4  -to_tang->  #2
  -Timed-> ArrivalT1  start vanishing paths:
        -VP-> t3  -to_tang->  #3

From #2 (3 timed trans)   [ 10 @ UT1, 9 @ UT2, ServingT2 ]
  -Timed-> FServT2  -to_tang->  #1
  -Timed-> ArrivalT2  -to_tang->  #4
  -Timed-> ArrivalT1  -to_tang->  #5

From #3 (3 timed trans)   [ 9 @ UT1, 10 @ UT2, ServingT1 ]
  -Timed-> FServT1  -to_tang->  #1
  -Timed-> ArrivalT2  -to_tang->  #6
  -Timed-> ArrivalT1  -to_tang->  #7

From #4 (3 timed trans)   [ 10 @ UT1, 8 @ UT2, WQ2, ServingT2 ]
  -Timed-> FServT2  start vanishing paths:
        -VP-> t4  -to_tang->  #2
  -Timed-> ArrivalT2  -to_tang->  #8
  -Timed-> ArrivalT1  -to_tang->  #9

From #5 (3 timed trans)   [ 9 @ UT1, 9 @ UT2, WQ1, ServingT2 ]
  -Timed-> FServT2  start vanishing paths:
        -VP-> t3  -to_tang->  #3
  -Timed-> ArrivalT2  -to_tang->  #9
  -Timed-> ArrivalT1  -to_tang->  #10
--More--
```

Figura 10. Porzione di TRG

```
*** Start of TRG ***

From #1 (2 timed trans)   [ 10 @ UT1, 10 @ UT2, Server ]
  -Timed-> ArrivalT2  start vanishing paths:
        -VP-> t4  -to_tang->  #2    [ 10 @ UT1, 9 @ UT2, ServingT2 ]
  -Timed-> ArrivalT1  start vanishing paths:
        -VP-> t3  -to_tang->  #3    [ 9 @ UT1, 10 @ UT2, ServingT1 ]

From #2 (3 timed trans)   [ 10 @ UT1, 9 @ UT2, ServingT2 ]
  -Timed-> FServT2  -to_tang->  #1    [ 10 @ UT1, 10 @ UT2, Server ]
  -Timed-> ArrivalT2  -to_tang->  #4    [ 10 @ UT1, 8 @ UT2, WQ2, ServingT2 ]
  -Timed-> ArrivalT1  -to_tang->  #5    [ 9 @ UT1, 9 @ UT2, WQ1, ServingT2 ]

From #3 (3 timed trans)   [ 9 @ UT1, 10 @ UT2, ServingT1 ]
  -Timed-> FServT1  -to_tang->  #1    [ 10 @ UT1, 10 @ UT2, Server ]
  -Timed-> ArrivalT2  -to_tang->  #6    [ 9 @ UT1, 9 @ UT2, WQ2, ServingT1 ]
  -Timed-> ArrivalT1  -to_tang->  #7    [ 8 @ UT1, 10 @ UT2, WQ1, ServingT1 ]

From #4 (3 timed trans)   [ 10 @ UT1, 8 @ UT2, WQ2, ServingT2 ]
  -Timed-> FServT2  start vanishing paths:
        -VP-> t4  -to_tang->  #2    [ 10 @ UT1, 9 @ UT2, ServingT2 ]
  -Timed-> ArrivalT2  -to_tang->  #8    [ 10 @ UT1, 7 @ UT2, 2 @ WQ2, ServingT2 ]
  -Timed-> ArrivalT1  -to_tang->  #9    [ 9 @ UT1, 8 @ UT2, WQ1, WQ2, ServingT2 ]

From #5 (3 timed trans)   [ 9 @ UT1, 9 @ UT2, WQ1, ServingT2 ]
  -Timed-> FServT2  start vanishing paths:
        -VP-> t3  -to_tang->  #3    [ 9 @ UT1, 10 @ UT2, ServingT1 ]
  -Timed-> ArrivalT2  -to_tang->  #9    [ 9 @ UT1, 8 @ UT2, WQ1, WQ2, ServingT2 ]
  -Timed-> ArrivalT1  -to_tang->  #10    [ 8 @ UT1, 9 @ UT2, 2 @ WQ1, ServingT2 ]
--More--
```

Figura 11. Porzione di TRS

```
[root@localhost ~]# newMT nets/ProgettoLMC/caso2/SingleServer-MultiUser-Prio

Start EMC construction

real    0m0.005s
user    0m0.000s
sys     0m0.001s
      1
    100
    200
    221 rows computed


number of doubles : 12
maximum row length : 3
maximum tree depth : 3
average tree depth : 1.242857

CPU time (s.) : user 0, system 0
Max memory size : 1104 Kbyte
Page faults : reclaims 229, faults 0
Context Switches : voluntary 1, forced 13
Swap : 0

[root@localhost ~]#
```

Figura 12. Generazione dell'Embedded Markov Chain (EMC)

```
[root@localhost ~]# showmtx nets/ProgettoLMC/caso2/SingleServer-MultiUser-Prio
>>    to mark #1, weight #1, 2 entries
      from mark #2, rate #3
      from mark #3, rate #5
>>    to mark #2, weight #2, 3 entries
      from mark #1, rate #1
      from mark #4, rate #3
      from mark #6, rate #5
>>    to mark #3, weight #4, 3 entries
      from mark #1, rate #1
      from mark #5, rate #3
      from mark #7, rate #5
>>    to mark #4, weight #2, 3 entries
      from mark #2, rate #2
      from mark #8, rate #3
      from mark #11, rate #5
>>    to mark #5, weight #2, 1 entries
      from mark #2, rate #2
>>    to mark #6, weight #4, 3 entries
      from mark #3, rate #4
      from mark #9, rate #3
      from mark #12, rate #5
>>    to mark #7, weight #4, 3 entries
      from mark #3, rate #4
      from mark #10, rate #3
      from mark #13, rate #5
>>    to mark #8, weight #2, 3 entries
      from mark #4, rate #2
      from mark #14, rate #3
```

Figura 13. Porzione di una Embedded Markov Chain (EMC)

```
    12 rate values:
Double# 1 -> 0.500000
Double# 2 -> 0.444444
Double# 3 -> 0.111111
Double# 4 -> 0.363636
Double# 5 -> 0.272727
Double# 6 -> 0.800000
Double# 7 -> 0.200000
Double# 8 -> 0.571429
Double# 9 -> 0.428571
Double# 10 -> 4.000000
Double# 11 -> 1.000000
Double# 12 -> 1.333333
[root@localhost ~]#
```

Figura 14. Rate dell'Embedded Markov Chain (EMC)

re fatto sia tramite GUI da *GSPN→Solve→GSPN Solution→Steady State*, oppure da terminale con il comando *newSO netdirectory/netname*. Eseguendo invece il comando *showprob netdirectory/netname*. Al termine di questa analisi è possibile inoltre vedere anche la distribuzione dei token in ciascun place e il throughput di ciascuna transizione. Per vedere la distribuzione dei place è necessario selezionare dalla object bar l'icona *Results*, poi dal menu delle azioni selezionare l'azione *Show* e fare clic il place del quale si vuole visualizzare la distribuzione dei tokens, come si può vedere da figura 15 .
Transient solution: computa la soluzione in transitorio della CTMC associata alla rete. Questo può essere fatto sia tramite GUI da *GSPN→Solve→GSPN Solution→Transient*; quando si sceglie questa soluzione viene richiesto l'inserimento del tempo di transitorio.

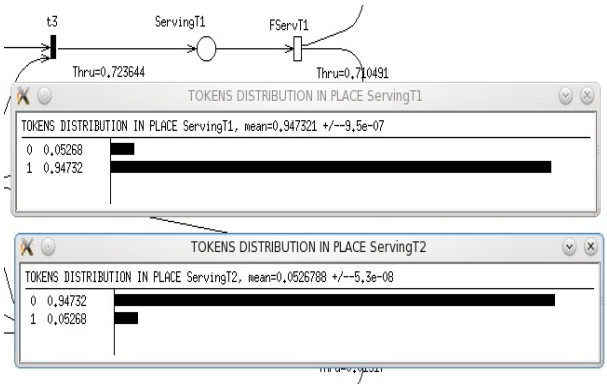


Figura 15. Distribuzione dei tokens e througput

Simulazione: dalla menu bar *GSPN→Simulation...* è possibile accedere a questa funzionalità che consente di simulare in maniera interattiva l'evoluzione della rete. Infatti, una volta fatta partire la simulazione, le transizioni abilitate in quel momento lampeggiano, ed è a discrezione dell'utente quale fare scattare. In figura 16 si può vedere l'interfaccia di simulazione, nella quale sono abilitate le transizioni *ArrivaleT1* e *ArrivaleT2*, cioè le uniche abilitate a partire dal marking iniziale.

Come anticipato le Stochastic Well Formed Nets sono un'estensione colorata delle reti di Petri e possono essere analizzate tramite i risolutori SWN da menu, oppure da linea di comando. Da questa analisi si possono ottenere i medesimi risultati ottenibili analizzando reti non colorate, ovvero numero medio di token per place e throughput delle transizioni, i quali sono indipendenti dal colore delle classi. Per mostrare l'analisi disponibile per le PN colorate si prende come caso di studio di riferimento il sistema a polling con più servitori e più code da servire ciclicamente. La versione colorata consente una rappresentazione più compatta

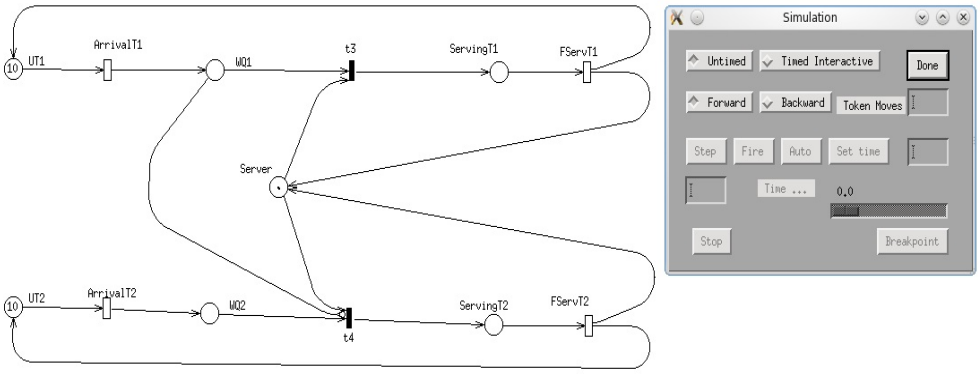


Figura 16. Simulazione della rete

del medesimo modello che si avrebbe con una rappresentazione non colorata. La rete è visibile in figura 17 . Per l'analisi di reti colorate vi sono due possibilità,

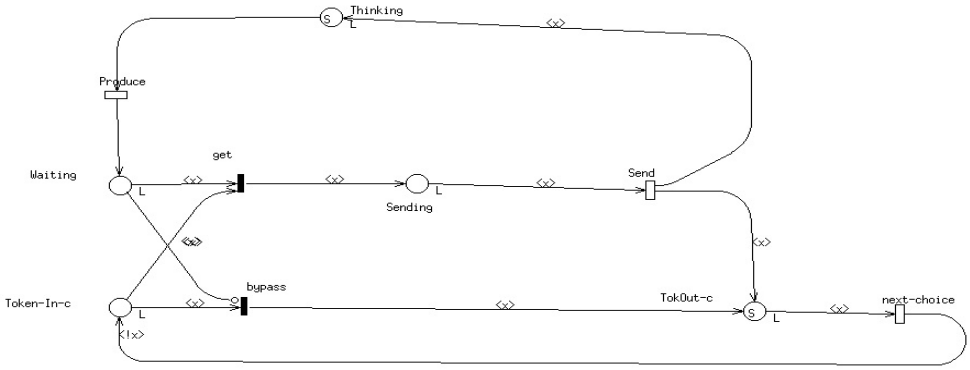


Figura 17. Modello sistema a polling, versione colorata

entrambe eseguibili sia da GUI che da terminale:

1. *ordinaria*: effettua l'analisi così come è stata conosciuta per le reti non colorate;
2. *simbolica*: specifica per i modelli SWN;

Si fa notare che l'analisi ordinaria al momento ha qualche problema, sia se eseguita da GUI, sia se eseguita da linea di comando. Nella versione simbolica viene utilizzato un marking simbolico. L'analisi delle invarianti e delle proprietà strut-

turali sono sempre disponibili così come illustrate in precedenza. Considereremo le funzionalità per il caso *simbolico* che vengono illustrate nel seguito:

Simulazione: per i modelli colorati, eseguibile da *SWN→Symbolic→Simulation* o da linea di comando con *swn_sym_sim netdirectory/netname option start* in cui le *option* sono:

- **-f**: batch spacing, ovvero la lunghezza della fase di evoluzione tra un batch e l'altro;
- **-t**: transitorio iniziale, ovvero la lunghezza della fase iniziale;
- **-m**: minimum batch length, ovvero la dimensione minima del batch;
- **-M**: maximum batch length, ovvero la dimensione massima del batch;
- **-a**: accuratezza, ovvero la precisione dell'approssimazione nella stima dei parametri;
- **-c**: livello di confidenza, ovvero il livello di confidenza nella stima dei parametri;
- **-s**: seed number;
- **-o**: mostra a video;

Sia da GUI che da linea di comando è possibile vedere i risultati della simulazione batch per batch. Eseguendo la simulazione da GUI i medesimi parametri possono essere specificati nei campi appropriati della finestra visibile in figura 18 . Quando si lavora con le reti di Petri colorate, in caso di

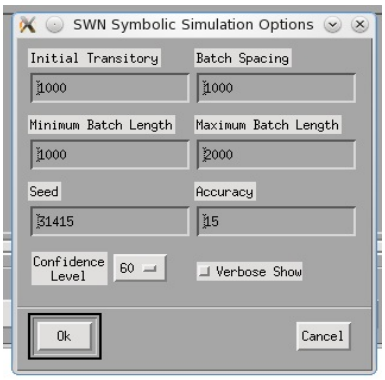


Figura 18. Parametri della simulazione per SWN

ambiguità, GreatSPN le segnala poco prima di eseguire la run di simulazione con i parametri specificati.

ComputeTRG: da GUI *SWN→Symbolic→Compute RG*, per ottenere il grafo di raggiungibilità del modello e ottenere gli indici di performance nello steady state (a regime).

Transient: da GUI *SWN→Symbolic→Transient*, per ottenere il grafo di raggiungibilità del modello e ottenere gli indici di performance nel transitorio.

In figura 19 è possibile vedere una parte dell'esito della generazione del RG. La soluzione in transitorio richiede l'inserimento dei parametri di cui sopra.

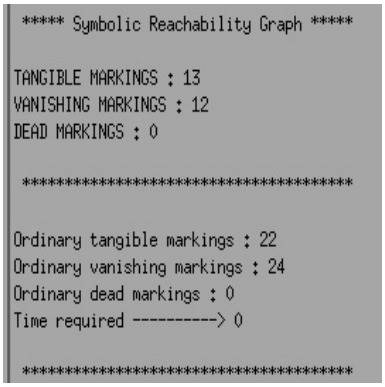


Figura 19. Computazione RG, PN colorate

Un'altra funzionalità messa a disposizione dal tool è la possibilità di gestire modelli i cui tempi delle transizioni temporizzate non hanno distribuzione esponenziale ma generalizzata, ovvero una distribuzione più simile a quello che avviene nella realtà, soprattutto in alcuni modelli. Graficamente una transizione generalizzata adotta il medesimo formalismo di quelle con distribuzione esponenziale negativa. Viene poi trattato in maniera diversa in fase di analisi, grazie alla funzionalità dell'*Extended - GSPN*. Per poter effettuare l'analisi si deve creare un file che ha una sintassi secondo una certa grammatica e con estensione *.dis* . Ogni riga del file rappresenta una descrizione di ciascuna transizione che deve essere considerata generalizzata in fase di analisi. L'analisi è poi disponibile dalla menu bar *E-GSPN* e si articola in:

Simulation: lancia la simulazione;
Compute RG: computa il grafo di raggiungibilità e ottiene gli idici di performance a regime;
Transient: computa il grafo di raggiungibilità e ottiene gli idici di performance in transitorio;

Questa è una delle parti ancora sperimentali del tool e in continua evoluzione, unita alla parte delle SWN ed al momento non funzionante.

3 Model Checking con GreatSPN

GreatSPN possiede un modulo dedicato al model checking chiamato *ord_ rgMEDD*; oltre a questo model checker è stato sviluppato anche a un altro model checker, non ancora integrato stabilmente con GreatSPN, chiamato

$MC4CSL^{TA}$.
Il model checking è un metodo utilizzato per la verifica dei sistemi formali; data un certa proprietà p, un modello M e uno stato iniziale s_0, si vuole verificare se p è valida nel modello M che ha configurazione iniziale s_0 , ovvero se M, $s_0 \models p$.

3.1 ord_rgMEDD

Il model checking di ord_rgMEDD è Computation Tree Logic (CTL), cioè solo qualitativo. La logica CTL ha una espressività diversa dalla Linear Temporal Logic (LTL). LTL fornisce proprietà di singoli percorsi. Per sistemi non deterministici le proprietà devono valere su tutti i possibili percorsi. La CTL ha la nozione di computazione ad albero a partire da un certo stato, che significa che, a partire da un certo stato, verifica tutti i possibili percorsi. La sintassi usata dalla CTL è la seguente: $\phi ::= p|\phi \rightarrow \phi|F|A\psi|E\psi$, $\psi ::= X\phi|G\phi|F\phi|\phi U\phi$. ϕ esprime proprietà sugli stati, ψ sui percorsi; A ed E sono i nuovi simboli introdotti dalla logica CTL, mentre gli altri hanno la medesima semantica di quelli della logica LTL, ovvero sempre applicabili ad un unico percorso.
Il modo più semplice per usare il model checker è scrivere all'interno di un file di testo *nome_della_rete.ctl* le proprietà che si desiderano verificare; il file deve essere contenuto nella stessa directory in cui vi è il modello della rete. È eseguibile da linea di comando tramite *ord_rgMEDD <netdir>/<net> -B INTEGER -C -o* dove netdir è la cartella contenente il modello del sistema e net è il file del modello senza estensione. L'opzione *-B* serve per specificare il bound dei place e, di default, se non viene specificato niente, è 255. Il valore specificato ha impatto sul tempo di computazione. L'opzione *-C* specifica che devono essere valutate le formule dentro al file; l'opzione *-o* che i risultati devono essere visualizzati in output. Su ogni riga del file va specificata non più di una formula; la valutazione di ogni formula è indipendente e viene fatta sulla marcatura iniziale.
Le formule CTL possono essere scritte secondo la seguente grammatica:

```
<CTLformula> ::= <atomicProposition> | <CTLformula> |
                         <CTLformula> ‘‘and’’ <CTLformula> |
                         <CTLformula> ‘‘or’’ <CTLformula> |
                         ‘‘not’’ <CTLformula> | <CTLformula> ‘‘->’’ <CTLformula> |
                         ‘‘E’’ ‘‘X’’ <CTLformula> | ‘‘E’’ ‘‘G’’ <CTLformula> |
                         ‘‘E’’ ‘‘[’’ <CTLformula> ‘‘U’’ <CTLformula> ‘‘]’’ |
                         ‘‘A’’ ‘‘X’’ <CTLformula> | ‘‘A’’ ‘‘F’’ <CTLformula> |
                         ‘‘E’’ ‘‘F’’ <CTLformula> | ‘‘A’’ ‘‘G’’ <CTLformula> |
                         ‘‘A’’ ‘‘[’’ <CTLformula> ‘‘U’’ <CTLformula> ‘‘]’’

<atomicProposition> ::= <inequality> | <boolvalue> | ‘‘ndeadlock’’ |
         ‘‘deadlock’’ | ‘‘enk <var>’’

<boolvalue> ::=  ‘‘true’’ | ‘‘false’’

<inequality> ::=  ‘‘(’’ <expression> <comp oper> <number expr> ‘‘)’’

<comp oper> ::= ‘‘<’’ | ‘‘>’’ | ‘‘<=’’ | ‘‘>=’’ | ‘‘=’’ | ‘‘! =’’
```

```
<expression> ::=“(” <expression> <arit oper> <number expr> “)” | <term> |
                        “(” <number expr> “)”

<arit oper>  ::= “+” | “?” | “?” | “/”

<term> ::= <number expr> “?” <var> | <number expr> “/” <var> | <var>

<number expr> ::= “(” <number expr> <arit oper> <number expr> “)” | <number>

<var>  ::= [(A-Z)(a-z)][(A-Z)(a-z)(0-9)]?

<numbr> ::= R^+
```

Per il test del model checking si è fatto riferimento alla rete di figura 3 e sono state verificate le seguenti proprietà:

A G not(ServingT1 $>$ 0 and ServingT2 $>$ 0)
E F (ServingT1 $>$ 0)
E F (ServingT2 $>$ 0)
E F (WQ1 $>$ 0 or WQ2 $>$ 0)
A F (WQ1 $>$ 0 or WQ2 $>$ 0)
E F (ServingT1 $>$ 0 and ServingT2 $>$ 0)
A G (UT1 + WQ1 + ServingT1 = 10)
A G not (ServingT1 $>$ 1)

Alcuni risultati sono riportati nelle figure 20, 21 e 22. Le proprietà sopra riportate consento di esprimere *raggiungibilità*, *safety/mutua esclusione* e consente di verificare se gli utente prima o poi avranno accesso al servizio.

```
Start CTL

----------------------------------------
--- A G not(ServingT1 > 0 and ServingT2 > 0) ---
        Evaluation: true
        Time set generation: 0 sec
        Time set generation and valutation: 0 sec
        Time print: 0 sec


================= MEMORY CTL==================
Peak Used Memory (RS): 31116B
Current Used  Memory (RS): 17868B
Peak Used Memory (potential RG): 276596B
Current Used  Memory (potential RG): 62376B
Peak Used Memory (potential RG + Real): 8032B
Current Used  Memory (potential RG +Real): 8032B
==============================================
```

Figura 20. Model checking: CTL esempio 1

L’estensione è scritta in C++ la sua particolarità è l’uso di strutture dati efficienti, i *Decision Diagram*, per codificare lo spazio degli stati.

```
--- E F (ServingT1 > 0) ---
        Evaluation: true
        Time set generation: 0 sec
        Time set generation and valutation: 0 sec
        Time print: 0 sec


================ MEMORY CTL==================
Peak Used Memory (RS): 31116B
Current Used  Memory (RS): 20636B
Peak Used Memory (potential RG): 276596B
Current Used  Memory (potential RG): 61740B
Peak Used Memory (potential RG + Real): 8032B
Current Used  Memory (potential RG +Real): 7996B
============================================

=================== NODE ====================
Peak Node Number (RS): 628
Used Node Number (RS): 434
Peak Node Number (potential RG): 3086
Used Node Number (potential RG): 1088
Peak Node Number (potential RG + Real): 120
Used Node Number (potential RG +Real): 119
============================================


--- E F (ServingT2 > 0) ---
        Evaluation: true
        Time set generation: 0.01 sec
        Time set generation and valutation: 0.01 sec
        Time print: 0 sec
```

Figura 21. Model checking: CTL esempio 2

```
================================================


--- A G (UT1 + WQ1 + ServingT1 = 10) ---
        Evaluation: true
        Time set generation: 0 sec
        Time set generation and valutation: 0 sec
        Time print: 0 sec


================ MEMORY CTL==================
Peak Used Memory (RS): 42448B
Current Used  Memory (RS): 42448B
Peak Used Memory (potential RG): 74064B
Current Used  Memory (potential RG): 14108B
Peak Used Memory (potential RG + Real): 7876B
Current Used  Memory (potential RG +Real): 7876B
================================================

=================== NODE ====================
Peak Node Number (RS): 892
Used Node Number (RS): 892
Peak Node Number (potential RG): 1342
Used Node Number (potential RG): 312
Peak Node Number (potential RG + Real): 152
Used Node Number (potential RG +Real): 152
============================================


--- A G not (ServingT1 > 1) ---
        Evaluation: true
        Time set generation: 0 sec
        Time set generation and valutation: 0 sec
        Time print: 0 sec
```

Figura 22. Model checking: CTL esempio 3

3.2 MC4CSL^{TA}

É un model cheker non ancora integrato stabilmente con GreatSPN; è un model checker efficiente per catene di Markov per la logica stocastica CLS^{TA}. Anch'esso, come ord_rgMEDD è scritto in C++ . Il model checking stocastico, a differenza del model checking classico che fornisce solamente una risposta booleana, consente di calcolare la probabilità che un certo evento si verifichi nel corso del tempo. Quindi, questo tipo di model checking, oltre a dire se una proprietà può o non può essere rispettata in un certo modello, consente anche di sapere con quale proprietà può verificarsi. Le catene di Markov tempo continue modellano sistemi i cui tempi di transizione seguono una distribuzione esponenziale negativa, hanno cioè un certo rate associato. In tal caso si utilizza la Continuous Stochastic Logic (CSL) che aggiunge alcuni operatori assenti nelle altre logiche. La sintassi è la seguente: $\phi ::= p|\phi \rightarrow \phi|F|P_J(\psi)|S_J(\psi)$, $X\phi|\phi U\phi^{'}|\phi U^{'}\phi^{'}$
L'operatore $\boldsymbol{P}$ ha la seguente semantica:

$s \models P_J(\psi)$ se $Pr\{\pi \in Path(s)|\pi \models \psi\} \in J$.

L'operatore $\boldsymbol{S}$, detto anche operatore steady-state, computa la probabilità che ϕ sia valida sul periodo. La semantica di tale operatore è:

$s \models S_J(\phi)$ se $\lim_{t\rightarrow+\infty} Pr\{\pi \in Path(s)|\pi@t \models \phi\} \in J$.

Si introduce anche l'operatore $U^{'}$ di tipo bounded che limita il tempo al quale qualcosa deve accadere, la cui semantica è: $\pi \models \phi U^{\leq t}\phi^{'}$ se $\exists\tau \in R_0^+, \tau < t, \pi\langle\tau ...\rangle \models \phi^{'} \wedge \forall\tau^{'}, \tau^{'} < \tau, \pi\langle\tau^{'} ...\rangle \models \phi$.
Nella CSL^{TA} le formule sono sia query steady-state o formule su percorsi, dove i percorsi accettati sono specificati attraverso i *Deterministic Timed Automata* (DTA). Il model checker prende in ingresso una CTMC etichettata, una query ed una o più DTA coinvolti nella query (che è possibile esprimere in forma parametrica) e computa la probabilità necessaria per raggiungere il valore di verità della formula. CSL^{TA} è una logica stocastica per catene di Markov tempo continue. Un espressione in linguaggio CSL^{TA} è definita secondo la seguente grammatica:
expr ::= True | False | <ap> | (<expr>) | <expr> && <expr> | <expr> || <expr> | STEADY (<expr>) | PROB_TA <∼> <λ> <dta> (<parameters ...>
dove *<ap>* è una proposizione atomica (condizione booleana), <∼>un operatore di confronto $\{\leq, <, >, \geq\}$, *<λ>* è un valore nell'intervallo [0, 1], *<dta>* è il nome del file dta in cui vi è il percorso di ricerca dta, e *<parameters ...>* sono i parametri del dta. Gli operatori CSL^{TA} $S_{\sim\lambda}()$ e $P_{\sim\lambda}()$ sono chiamati *STEADY* e *PROB_TA*, rispettivamente.Il linguaggio delle proposizioni atomiche include espressioni di tipo marking-dependent sulle variabili di stato della CTMC.
Il model checker viene eseguito da linea di comando e carica modelli progettati con i tool GreatSPN, Prism ed MRMC. CSL^{TA} è quindi una logica capace di esprimere proprietà sul comportamento di una CTMC, in particolare in termini di possibile esecuzioni della CTMC (come ad esempio la probabilità che un insieme di percorsi che esibiscono un certo comportamento sia sopra/sotto una certa soglia). La verifica di tali formule è fatta su CTMC espresse come GSPN.

Questo model checker è stato validato con PRISM e Cosmos. Per poter eseguire una query si deve scrivere *./DSPN-Tool-Release -load <model> -cslta <expr>* in cui *<model>* è il nome della rete (senza estensione *.net*) e *<expr>* segue la grammatica di cui sopra. Si fa notare che affinché l'esecuzione del comando vada a buon fine i files *.net* e *.def* devono essere nella medesima directory. L'esecuzione del comando innesta automaticamente la computazione del grafo di raggiungibilità (il TRG) e poi viene effettuata la valutazione della espressione specificata su tutti gli stati della CTMC. Il model checker effettua due tipi di analisi:

1. *forward*;
2. *backward*;

Di default viene eseguita un'analisi numerica di tipo backward, mentre specificando l'opzione *-cslta0 <expr> o -csltaN <state> <expr>* viene effettuata un'analisi numerica di tipo forward. Gli approcci Forward e backward si differenziano per il modo in cui formulano il sistema di equazioni lineari che computano l'operatore *P()*. Il metodo forward comincia dal vettore delle probabilità *alfa* al tempo zero e computa il limite del vettore delle probabilità di raggiungere un certo stato T. L'analisi backward invece parte da una situazione certa di essere in un certo stato T a regime, e computa, per ogni stato, la probabilità di raggiungere lo stato T, allo stesso costo come nella computazione di tipo forward partendo da un singolo stato iniziale. Nonostante sia la stessa cosa, anche nell'approccio backward viene costruito lo spazio degli stati $M \ x \ A$ in avanti, iniziando da uno o più stati iniziali, ed è solo la soluzione numerica che funziona all'indietro.CSL^{TA} è più espressivo di CSL e questo si traduce in un algoritmo di model checking più complesso: la verifica di una formula richiede la soluzione steady-state di un Markov Regenerative Process (MRP) ottenuto come prodotto scalare tra la catena di Markov e il DTA. Un esempio di DTA è riportato di seguito:

```
DTA SampleDTA = {
  CLOCKVALUESET = { alpha }
  ACTIONSET = { a, b }
  ATOMICPROPOSITIONSET = { PHI, PSI }
  LOCATIONS = {
    INITIAL l_0 : PHI;
            l_1 : PHI && !PSI;
            l_2 : PHI && PSI;
      FINAL l_3 : PSI;
  }
  EDGES = {
    l_0 -> l_0 (x < alpha, {a}, RESET);
    l_0 -> l_2 (x = 0, #);
    l_1 -> l_0 (x = alpha, #, RESET);
    l_2 -> l_1 (x < alpha, Act);
    l_2 -> l_2 (x > alpha, Act);
    l_2 -> l_3 (x > 0, {b});
  }
```

}

Il model checker viene fornito con già alcuni DTA parametrici, che sono l'until operator $[0, \beta)$, $[0, \infty)$, $[\alpha, \infty)$ con $(0 < \alpha < \infty)$, $(0 < \alpha < \beta < \infty)$ e $[\alpha, \infty)$ con $(0 < \alpha < \infty)$.
Eseguendo il comando per lanciare il model checker i risultati saranno visualizzati sul terminale, oppure, aggiungendo in fondo l'opzione *-wrsat nomefile* il vettore delle probabilità di stato sarà scritto nel file *nomefile* . Considerando come esempio la rete di figura 3, ed eseguendo ad esempio il comando *./DSPN-Tool-Release -load /Users/chiaracontoli/. . . /Progetto/Model/caso2/SingleServer-MultiUser-Prio -cslta "#ServingT1>=50" -wrsat statfile*, si avrà su terminale l'output di figura 23 e 24 . Il risultato visibile in *statfile* è il vettore delle probabilità di

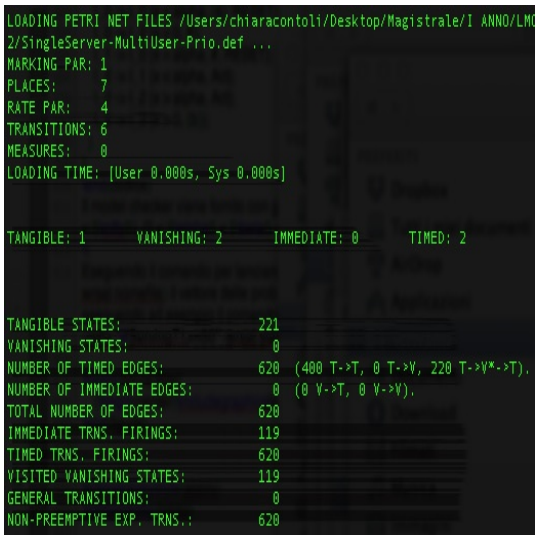


Figura 23. Model checking: CTL esempio 1

stato, che per ragioni di spazio non viene riportato, e che contiene tutti valori nulli, nel caso specifico, poiché non vi sarà mai un numero di utenti in servizio maggiore-uguale a 50 (lo stesso risultato si ottiene anche specificando la condizione *"#ServingT1>1"*). L'output prodotto sono i valori di verità della formula specificata (0 o 1) oltre che le probabilità dei percorsi della CTMC che soddisfano la formula.
Sono stati eseguiti i seguenti comandi:

./DSPN-Tool-Release -load /Users/chiaracontoli/Desktop/. . . /Model/caso2/SingleServer-MultiUser-Prio -dta-path ../DTA/ -cslta PROB_TA > 0.10 until_0B (3 | | #WQ1>=0, #ServingT1=1) -wrsat mm1_ex1.txt

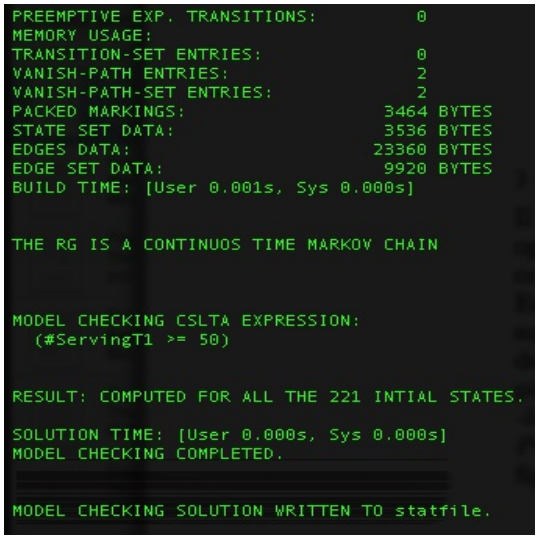


Figura 24. Model checking: CTL esempio 1

Il risultato viene scritto sul file, ed al terminale viene riportato quando segue:

```
LOADING PETRI NET FILES /Users/chiaracontoli/Desktop/Magistrale/I ANNO/LMC/
Progetto/Model/caso2/SingleServer-MultiUser-Prio.net AND
/Users/chiaracontoli/Desktop/Magistrale/I ANNO/LMC/
Progetto/Model/caso2/SingleServer-MultiUser-Prio.def ...
MARKING PAR: 1
PLACES:      7
RATE PAR:    4
TRANSITIONS: 6
MEASURES:    0
LOADING TIME: [User 0.000s, Sys 0.000s]


ADDING "../DTA/" TO THE DTA SEARCH PATHs.
TANGIBLE: 1       VANISHING: 2       IMMEDIATE: 0
TIMED: 2



TANGIBLE STATES:                   221
VANISHING STATES:                    0
NUMBER OF TIMED EDGES:             620
(400 T->T, 0 T->V, 220 T->V*->T).
NUMBER OF IMMEDIATE EDGES:           0  (0 V->T, 0 V->V).
TOTAL NUMBER OF EDGES:             620
```

```
IMMEDIATE TRNS. FIRINGS:           119
TIMED TRNS. FIRINGS:               620
VISITED VANISHING STATES:          119
GENERAL TRANSITIONS:                 0
NON-PREEMPTIVE EXP. TRNS.:         620
PREEMPTIVE EXP. TRANSITIONS:         0
MEMORY USAGE:
TRANSITION-SET ENTRIES:              0
VANISH-PATH ENTRIES:                 2
VANISH-PATH-SET ENTRIES:             2
PACKED MARKINGS:                  3464 BYTES
STATE SET DATA:                   3536 BYTES
EDGES DATA:                      23360 BYTES
EDGE SET DATA:                    9920 BYTES
BUILD TIME: [User 0.002s, Sys 0.000s]

THE RG IS A CONTINUOS TIME MARKOV CHAIN

MODEL CHECKING CSLTA EXPRESSION:
  PROB_TA > 0.1 "until_0B" (3 | | (#WQ1 >= 0), (#ServingT1 = 1))

LOADING DTA FILE "../DTA//until_0B.dta" ...
DTA PARTITIONED INTO ZONES.
  <S,L,C> STATES IN MxA: 1

NUMBER OF TANGIBLE <S*L*C> STATES:
113 (INCLUDING TOP, BOT)
NUMBER OF CTMC STATES:                 221
NUMBER OF DTA LOCATIONS:               6
NUMBER OF TIMED CLOCK ZONES:           2
NUMBER OF DETERMINISTIC TRANSITIONS:   1
NONZERO ENTRIES IN Q:                  321
NONZERO ENTRIES IN Qbar:               101
NONZERO ENTRIES IN DELTA:              111
NON-TRIVIAL INITIAL STATES:            111

THERE ARE 2 REGENERATIVE STATES.

111 SUBORDINATED MARKOV CHAINS WILL BE COMPUTED.
COMPUTING SMC 1/111...
P MATRIX (113 x 113) HAS 222 NONZERO ENTRIES.
C MATRIX (113 x 113) HAS 4248 NONZERO ENTRIES.
EMC BUILD TIME: [User 0.019s, Sys 0.000s]

BACKWARD STEADY STATE SOLUTION.
FORWARD GAUSS-SEIDEL: iteration=1, err=1.32177e-02

FORWARD GAUSS-SEIDEL: convergence in 5 iterations, err=7.85494e-08

BACKWARD STEADY STATE SOLUTION OF THE MxA PROCESS COMPUTED.
```

```
RESULT: COMPUTED FOR ALL THE 221 INTIAL STATES.

SOLUTION TIME: [User 0.020s, Sys 0.000s]
MODEL CHECKING COMPLETED.

MODEL CHECKING SOLUTION WRITTEN TO mm1_ex1.txt.
```

./DSPN-Tool-Release -load /Users/chiaracontoli/Desktop/.../Model/caso2/SingleServer-MultiUser-Prio -dta-path ../DTA/ -cslta PROB_TA > 0.10 until_0B (1 | | #WQ2>=0, #ServingT2=1) -wrsat mm1_ex2.txt

Per eseguire un'analisi di tipo farward, e quindi calcolare le probabilità dallo stato iniziale, si deve utilizzare l'opzione *-cslta0 <expr>*, invece di *-cslta* . In generale, un'analisi di tipo forward a partire da un qualunque stato, anche diverso da quello iniziale, è possibile specificando l'opzione *-csltaN <state> <expr>* . Ripetendo i comandi precedenti in modalità forward:

./DSPN-Tool-Release -load /Users/chiaracontoli/Desktop/.../Model/caso2/SingleServer-MultiUser-Prio -dta-path ../DTA/ -cslta0 PROB_TA > 0.10 until_0B (1 | | #WQ2>=0, #ServingT2=1) -wrsat forward_mm1_ex2.txt
il cui risultato a terminale è il seguente

```
LOADING PETRI NET FILES /Users/chiaracontoli/Desktop/Magistrale/I ANNO/LMC/Progett
MARKING PAR: 1
PLACES:      7
RATE PAR:    4
TRANSITIONS: 6
MEASURES:    0
LOADING TIME: [User 0.000s, Sys 0.000s]

ADDING "../DTA/" TO THE DTA SEARCH PATHs.
TANGIBLE: 1       VANISHING: 2       IMMEDIATE: 0
TIMED: 2

TANGIBLE STATES:                    221
VANISHING STATES:                     0
NUMBER OF TIMED EDGES:              620
(400 T->T, 0 T->V, 220 T->V*->T).
NUMBER OF IMMEDIATE EDGES:            0  (0 V->T, 0 V->V).
TOTAL NUMBER OF EDGES:              620
IMMEDIATE TRNS. FIRINGS:            119
TIMED TRNS. FIRINGS:                620
VISITED VANISHING STATES:           119
GENERAL TRANSITIONS:                  0
NON-PREEMPTIVE EXP. TRNS.:          620
PREEMPTIVE EXP. TRANSITIONS:          0
MEMORY USAGE:
```

```
TRANSITION-SET ENTRIES:              0
VANISH-PATH ENTRIES:                 2
VANISH-PATH-SET ENTRIES:             2
PACKED MARKINGS:                  3464 BYTES
STATE SET DATA:                   3536 BYTES
EDGES DATA:                      23360 BYTES
EDGE SET DATA:                    9920 BYTES
BUILD TIME: [User 0.002s, Sys 0.000s]


THE RG IS A CONTINUOS TIME MARKOV CHAIN

MODEL CHECKING CSLTA EXPRESSION:
  STATE 0 |= PROB_TA > 0.1 "until_0B" (1 | | (#WQ2 >= 0), (#ServingT2 = 1))

LOADING DTA FILE "../DTA//until_0B.dta" ...
DTA PARTITIONED INTO ZONES.
  <S,L,C> STATES IN MxA: 1

NUMBER OF TANGIBLE <S*L*C> STATES:
113 (INCLUDING TOP, BOT)
NUMBER OF CTMC STATES:                 221
NUMBER OF DTA LOCATIONS:               6
NUMBER OF TIMED CLOCK ZONES:           2
NUMBER OF DETERMINISTIC TRANSITIONS:   1
NONZERO ENTRIES IN Q:                  411
NONZERO ENTRIES IN Qbar:               11
NONZERO ENTRIES IN DELTA:              111
NON-TRIVIAL INITIAL STATES:            111


COMPUTING STEADY STATE SOLUTION OF THE MARKOV RENEWAL PROCESS MxA...
THERE ARE 2 REGENERATIVE STATES.

1 SUBORDINATED MARKOV CHAINS WILL BE COMPUTED.
COMPUTING SMC 1/1...
P MATRIX (113 x 113) HAS 112 NONZERO ENTRIES.
C MATRIX (113 x 113) HAS 223 NONZERO ENTRIES.
EMC BUILD TIME: [User 0.000s, Sys 0.000s]

FORWARD STEADY STATE SOLUTION OF NON-ERGODIC DTMC WITH 2 RECURRENCE CLASSES.
FORWARD GAUSS-SEIDEL: iteration=1, err=6.89655e-01

FORWARD GAUSS-SEIDEL: convergence in 7 iterations, err=1.04167e-08

FORWARD STEADY STATE SOLUTION OF THE MxA PROCESS COMPUTED.

RESULT: TRUE. ACCEPTANCE PROBABILITY IS 0.563222046723

SOLUTION TIME: [User 0.003s, Sys 0.001s]
```

```
MODEL CHECKING COMPLETED.

MODEL CHECKING SOLUTION WRITTEN TO forward_mm1_ex2.txt.
```

Guardando all'interno del file forward_mm1_ex2.txt si può vedere l'esito della valutazione della formula nello stato iniziale; nell'esempio specifico la probabilità di cui sopra è del 0.5632220467, ovvero del 56.32% .
./DSPN-Tool-Release -load /Users/chiaracontoli/Desktop/.../Model/caso2/SingleServer-MultiUser-Prio -dta-path ../DTA/ -cslta0 PROB_TA > 0.10 until_0B (3 | | #WQ1>=0, #ServingT1=1) -wrsat forward_mm1_ex1.txt
Guardando all'interno del file si può vedere che la valutazione della formula nello stato iniziale è pari a 0.7483705437, cioè quasi il 75% .
./DSPN-Tool-Release -load /Users/chiaracontoli/Desktop/.../Model/caso2/SingleServer-MultiUser-Prio -dta-path ../DTA/ -csltaN 10 PROB_TA > 0.10 until_0B (3 | | #WQ1>=0, #ServingT1=1) -wrsat forwardN_mm1_ex1.txt
In questo caso è stato specificato uno stato, il decimo, e la valutazione della formula dà come esito una probabilità certa.

Per l'operatore $P_{\sim\lambda}()$ vengono utilizzate soluzioni numeriche proposte in diversi algoritmi, tra cui Gauss-Seidel (usato di default), Jacobi, Symmetric SOR e i metodi degli sottospazi di Krylov. Tutti questi metodi possono essere impostati tramite opportune opzioni:

-jor: per usare l'algoritmo di Jacobi;
-ssor: per usare l'algoritmo di Symmetric SOR;
-gmres -M <n>: per il metodo di Krylov per un'analisi steady state implicita, con l'opzione -M per specificare il massimo numero di iterazioni di Arnoldi. Si può anche specificare l'opzione *-epsilon <real>* per dare un livello di accuratezza della soluzione.

In certuni casi si potrebbe incorrere in un errore, ovvero quando la computazione tramite un certo algoritmo non converge; ad esempio, l'algoritmo di Gauss-Seidel se in meno di 10000 iterazioni non converge, genera errore: *FORWARD GAUSS-SEIDEL: failed to converge in 10000 iterations. ERROR: Solution method does not converge.* . Questo errore lo si è ottenuto tentando la sequente valutazione:

./DSPN-Tool-Release -load /Users/chiaracontoli/Desktop/.../Model/caso2/SingleServer-MultiUser-Prio -dta-path ../DTA/ -csltaN 10 PROB_TA > 0.10 until_0B (1 | | #WQ2>=0, #ServingT2=1) -wrsat forwardN_mm1_ex2.txt

Provando ad utilizzare un altro algoritmo risolutivo, ad esempio il Jacobi, si ottiene invece una convergenza, con valutazione di probabilità pari a 0.1626142029, circa 16.26% . L'analisi con Jacobi è stata ottenuta eseguendo

./DSPN-Tool-Release -load /Users/chiaracontoli/Desktop/.../Model/caso2/SingleServer-MultiUser-Prio -dta-path ../DTA/ -jor

-csltaN 10 PROB_ TA > 0.10 until_ 0B (1 | | #WQ2>=0, #ServingT2=1)
-wrsat forwardN_ mm1_ ex2.txt

Le opzioni specificate vengono eseguite in ordine sequenzialmente.

Non è stato possibile testare il funzionamento dell'operatore STEADY poiché al momento non è ancora implementato. Inoltre, la sintassi dell'operatore dovrebbe essere la seguente: *STEADY <∼> <λ> (<expr>)* .

4 Caso di studio: il protocollo TCP

Una delle principali applicazioni delle Reti di Petri consiste nella validazione dei protocolli; verificare l'affidabilità di un protocollo, soprattutto in scenari sempre più complessi, diventa un punto chiave e fornisce un notevole valore aggiunto.
Lo scenario che si vuole proporre è il seguente: al fine di semplificare la trattazione del problema, ci poniamo in uno caso in cui abbiamo un lato che funge solo da server (esegue quindi apertura passiva della connessione) ed uno che funge solo da client (quindi effettua solo apertura attiva della connessione). Questa è chiaramente una semplificazione, poiché in uno scenario reale, le connessioni possono essere aperte in maniera attiva/passiva da ambo le parti; così facendo, si ottiene una PN semplificata, poiché nel caso reale, entrambe le parti dovrebbero avere la parte di rete che gestisce la connessione in maniera attiva e quella in maniera passiva. Inoltre, questa versione semplificata esclude il caso in cui si possono verificare aperture di connessione simultanee. Analogo discorso viene fatto per la chiusura: quell'attiva viene fatta solo dal client.
La PN è stata modellata a partire dalla macchina a stati del TCP, ovvero modellando le azioni tipiche di come transizioni e i l'evoluzione di tali azioni come stati. In figura 25 è riportata una prima versione del TCP in cui si distinguono la fase di handshake e quella di chiusura. Gli stati

- S_CLOSED e C_CLOSED
- LISTEN
- SYN_SENT
- SYN_RCVD
- S_ESTABLISHED e C_ESTABLISHED
- AC_FIN_FIN_WAIT_1 e AC_FIN_FIN_WAIT_2
- AC_TIME_WAIT
- CLOSE_WAIT
- LAST_ACK

corrispondono esattamente ai medesimi stati presenti nella macchina a stati del TCP. Gli altri stati

- SYN_BY_C
- SYN_ACK_BY_S
- ACK_BY_C
- FIN_BY_AC e FIN_BY_PC
- ACK_BY_AC e ACK_BY_PC

sono condivisi dal Client e dal Server e servono per modellare il fatto che il Client (Server) ha ricevuto un certo pacchetto da parte del Server (Client). Le transizioni sono state modellate tutte come distribuzione esponenziale, considerando che le dinamiche che modellano il tempo di trasmissione, ricezione ed elaborazione dei messaggi sono di un ordine inferiore rispetto alle dinamiche di connessione/disconnessione. Una volta modellata la rete è stata fatta per prima cosa l'analisi delle P/T-invariants e della proprietà di boundness, così come mostrano figura 28. Il Reachability Graph, per le medesime ragioni di cui sopra, è stato calcolato per il caso di un marking in cui il numero di tokens nel place C_CLOSED è Client = 15, e il numero di tokens nel place S_CLOSED è Server = 2. Con questo marking il TRG arriva a 8571 stati di tipo tangible, così come mostrato in figura 27. Così come per il caso di studio visto in precedenza, il TRS e il TRG sono possono visualizzabili tramite i tool disponibili a linea di comando, e sono mostrati nelle figura **??** . L'EMC associata alla PN di figura 25 possiede anch'essa 8571 stati e 229 rate con i quali transita da uno stato all'altro. Se ne riporta uno stralcio in figura 29 . L'analisi in steady state che è stata fatta come esempio prevedeva Client = Server = 2, che per gli stati C_ESTABLISHED e S_ESTABLISHED ha riportato i risultati di figura 30 . Per quanto riguarda il model checking, con il model cheker ord_rgMEDD sono state verificate le seguenti proprietà:

E F(LISTEN > 0 or SYN_SENT > 0)
A X(LISTEN > 0 or SYN_SENT > 0)
A G(SYN_SENT > 0 -> A F(C_ESTABLISHED > 0))
A G(SYN_SENT > 0 -> A F(S_ESTABLISHED >0))
A G(AC_FIN_WAIT_1 > 0 -> A F(AC_TIME_WAIT > 0))
A G not(SYN_RCVD > 2)

Con queste proprietà si vuole verificare che i client e i server effettuano apertura attiva e passiva, che a fronte dell'invio di un SYN, prima o poi il client e server saranno connessi, che a fronte dell'invio di un FIN prima o poi entrambi chiuderanno la sessione aperta e che non potranno mai esserci aperte un numero di connessioni superiori al numero di server disponibili. L'output delle verifiche effettuate viene visualizzato sia a video che in un file di output temporaneo in cui è presente: ogni proprietà, la valutazione (true o false) e una lista dei marking a partire dai quali la formula è verificata. per comodità. L'elenco dei marking a partire dai quali la formula è verificata non è riportata a video, ma nel file *nome_rete.ctl.output* che viene generato all'esecuzione del model checking. Di seguito se ne riporta una porzione come esempio:

```
 ********** E F(LISTEN > 0 or SYN_SENT > 0) *********
Evaluation: true
S_CLOSED(2) C_CLOSED(15)
S_CLOSED(2) C_CLOSED(14) SYN_BY_C(1) SYN_SENT(1)
S_CLOSED(2) C_CLOSED(13) SYN_BY_C(2) SYN_SENT(2)
S_CLOSED(2) C_CLOSED(12) SYN_BY_C(3) SYN_SENT(3)
S_CLOSED(2) C_CLOSED(11) SYN_BY_C(4) SYN_SENT(4)
S_CLOSED(2) C_CLOSED(10) SYN_BY_C(5) SYN_SENT(5)
```

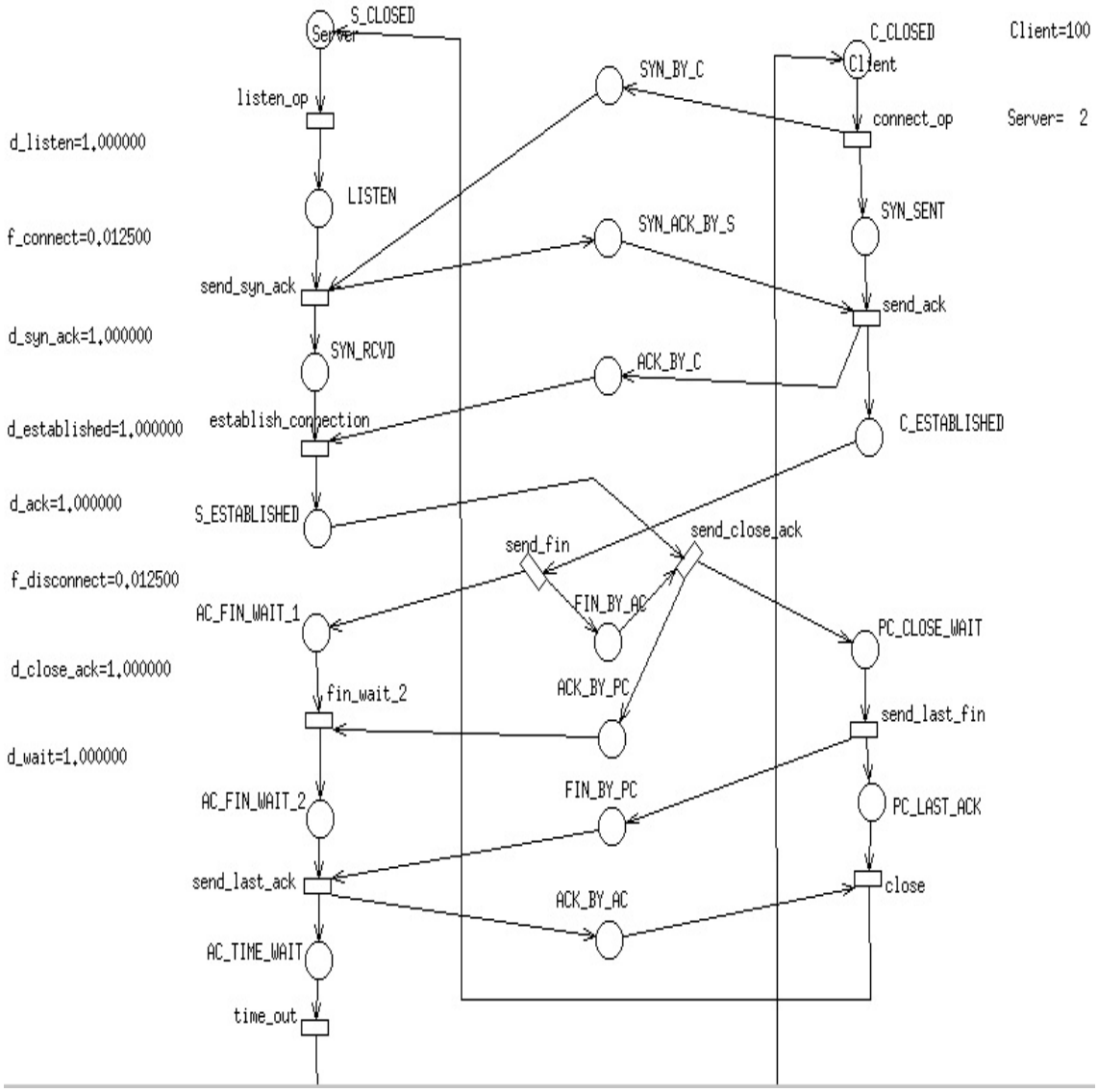
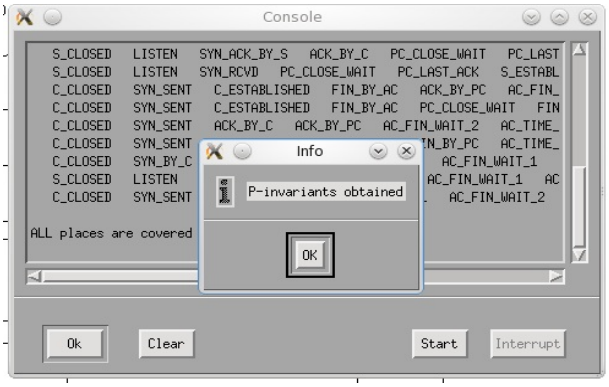
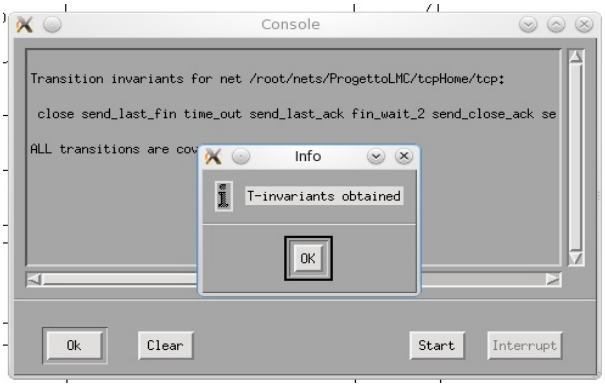


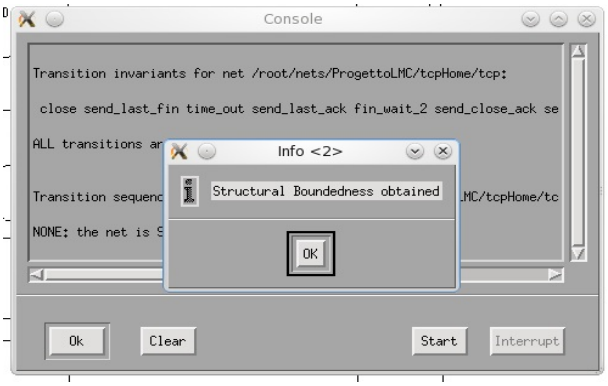
Figura 25. Protocollo TCP



(a) `P-Invariants`



(b) `T-Invariants`



(c) `Boundness`

Figura 26. Invarianti

Figura 27. TRG del protocollo TCP da GUI

```
S_CLOSED(2) C_CLOSED(9) SYN_BY_C(6) SYN_SENT(6)
S_CLOSED(2) C_CLOSED(8) SYN_BY_C(7) SYN_SENT(7)
S_CLOSED(2) C_CLOSED(7) SYN_BY_C(8) SYN_SENT(8)
S_CLOSED(2) C_CLOSED(6) SYN_BY_C(9) SYN_SENT(9)
S_CLOSED(2) C_CLOSED(5) SYN_BY_C(10) SYN_SENT(10)
S_CLOSED(2) C_CLOSED(4) SYN_BY_C(11) SYN_SENT(11)
S_CLOSED(2) C_CLOSED(3) SYN_BY_C(12) SYN_SENT(12)
S_CLOSED(2) C_CLOSED(2) SYN_BY_C(13) SYN_SENT(13)
S_CLOSED(2) C_CLOSED(1) SYN_BY_C(14) SYN_SENT(14)
S_CLOSED(2) SYN_BY_C(15) SYN_SENT(15)
S_CLOSED(1) C_CLOSED(15) LISTEN(1)

 ********** A X(LISTEN > 0 or SYN_SENT > 0) *********
Evaluation: true
S_CLOSED(2) C_CLOSED(15)
S_CLOSED(2) C_CLOSED(14) SYN_BY_C(1) SYN_SENT(1)
S_CLOSED(2) C_CLOSED(13) SYN_BY_C(2) SYN_SENT(2)
S_CLOSED(2) C_CLOSED(12) SYN_BY_C(3) SYN_SENT(3)
S_CLOSED(2) C_CLOSED(11) SYN_BY_C(4) SYN_SENT(4)
S_CLOSED(2) C_CLOSED(10) SYN_BY_C(5) SYN_SENT(5)
S_CLOSED(2) C_CLOSED(9) SYN_BY_C(6) SYN_SENT(6)
S_CLOSED(2) C_CLOSED(8) SYN_BY_C(7) SYN_SENT(7)
S_CLOSED(2) C_CLOSED(7) SYN_BY_C(8) SYN_SENT(8)
S_CLOSED(2) C_CLOSED(6) SYN_BY_C(9) SYN_SENT(9)
.....
.....
 ********** A G(SYN_SENT > 0 -> A F(C_ESTABLISHED > 0)) *********
Evaluation: true
S_CLOSED(2) C_CLOSED(15)
S_CLOSED(2) C_CLOSED(14) SYN_BY_C(1) SYN_SENT(1)
S_CLOSED(2) C_CLOSED(13) SYN_BY_C(2) SYN_SENT(2)
S_CLOSED(2) C_CLOSED(12) SYN_BY_C(3) SYN_SENT(3)
```

```
Initial marking is #1 :
   [ 2 @ S_CLOSED, 15 @ C_CLOSED ]


*** Start of TRG ***

From #1 (2 timed trans)   [ 2 @ S_CLOSED, 15 @ C_CLOSED ]
  -Timed-> listen_op  -to_tang->  #2
  -Timed-> connect_op (enabl=15) -to_tang->  #3

From #2 (2 timed trans)   [ S_CLOSED, 15 @ C_CLOSED, LISTEN ]
  -Timed-> listen_op  -to_tang->  #4
  -Timed-> connect_op (enabl=15) -to_tang->  #5

From #3 (2 timed trans)   [ 2 @ S_CLOSED, 14 @ C_CLOSED, SYN_BY_C, SYN_SENT ]
  -Timed-> listen_op  -to_tang->  #5
  -Timed-> connect_op (enabl=14) -to_tang->  #6

From #4 (1 timed trans)   [ 15 @ C_CLOSED, 2 @ LISTEN ]
  -Timed-> connect_op (enabl=15) -to_tang->  #7

From #5 (3 timed trans)   [ S_CLOSED, 14 @ C_CLOSED, SYN_BY_C, SYN_SENT, LISTEN ]
  -Timed-> send_syn_ack  -to_tang->  #8
  -Timed-> listen_op  -to_tang->  #7
  -Timed-> connect_op (enabl=14) -to_tang->  #9

From #6 (2 timed trans)   [ 2 @ S_CLOSED, 13 @ C_CLOSED, 2 @ SYN_BY_C, 2 @ SYN_SENT ]
  -Timed-> listen_op  -to_tang->  #9
  -Timed-> connect_op (enabl=13) -to_tang->  #10

From #7 (2 timed trans)   [ 14 @ C_CLOSED, SYN_BY_C, SYN_SENT, 2 @ LISTEN ]
  -Timed-> send_syn_ack  -to_tang->  #11
  -Timed-> connect_op (enabl=14) -to_tang->  #12

From #8 (3 timed trans)   [ S_CLOSED, 14 @ C_CLOSED, SYN_SENT, SYN_RCVD, SYN_ACK_BY_S ]
```

(a) TRG

```
Initial marking is #1 :
   [ 2 @ S_CLOSED, 15 @ C_CLOSED ]


*** Start of TRG ***

From #1 (2 timed trans)   [ 2 @ S_CLOSED, 15 @ C_CLOSED ]
  -Timed-> listen_op  -to_tang->  #2    [ S_CLOSED, 15 @ C_CLOSED, LISTEN ]
  -Timed-> connect_op (enabl=15) -to_tang->  #3    [ 2 @ S_CLOSED, 14 @ C_CLOSED, SYN_BY_C, SYN_SENT ]

From #2 (2 timed trans)   [ S_CLOSED, 15 @ C_CLOSED, LISTEN ]
  -Timed-> listen_op  -to_tang->  #4    [ 15 @ C_CLOSED, 2 @ LISTEN ]
  -Timed-> connect_op (enabl=15) -to_tang->  #5    [ S_CLOSED, 14 @ C_CLOSED, SYN_BY_C, SYN_SENT, LISTEN ]

From #3 (2 timed trans)   [ 2 @ S_CLOSED, 14 @ C_CLOSED, SYN_BY_C, SYN_SENT ]
  -Timed-> listen_op  -to_tang->  #5    [ S_CLOSED, 14 @ C_CLOSED, SYN_BY_C, SYN_SENT, LISTEN ]
  -Timed-> connect_op (enabl=14) -to_tang->  #6    [ 2 @ S_CLOSED, 13 @ C_CLOSED, 2 @ SYN_BY_C, 2 @ SYN_SENT ]

From #4 (1 timed trans)   [ 15 @ C_CLOSED, 2 @ LISTEN ]
  -Timed-> connect_op (enabl=15) -to_tang->  #7    [ 14 @ C_CLOSED, SYN_BY_C, SYN_SENT, 2 @ LISTEN ]

From #5 (3 timed trans)   [ S_CLOSED, 14 @ C_CLOSED, SYN_BY_C, SYN_SENT, LISTEN ]
  -Timed-> send_syn_ack  -to_tang->  #8    [ S_CLOSED, 14 @ C_CLOSED, SYN_SENT, SYN_RCVD, SYN_ACK_BY_S ]
  -Timed-> listen_op  -to_tang->  #7    [ 14 @ C_CLOSED, SYN_BY_C, SYN_SENT, 2 @ LISTEN ]
  -Timed-> connect_op (enabl=14) -to_tang->  #9    [ S_CLOSED, 13 @ C_CLOSED, 2 @ SYN_BY_C, 2 @ SYN_SENT, LISTEN ]

From #6 (2 timed trans)   [ 2 @ S_CLOSED, 13 @ C_CLOSED, 2 @ SYN_BY_C, 2 @ SYN_SENT ]
  -Timed-> listen_op  -to_tang->  #9    [ S_CLOSED, 13 @ C_CLOSED, 2 @ SYN_BY_C, 2 @ SYN_SENT, LISTEN ]
  -Timed-> connect_op (enabl=13) -to_tang->  #10    [ 2 @ S_CLOSED, 12 @ C_CLOSED, 3 @ SYN_BY_C, 3 @ SYN_SENT ]

From #7 (2 timed trans)   [ 14 @ C_CLOSED, SYN_BY_C, SYN_SENT, 2 @ LISTEN ]
  -Timed-> send_syn_ack  -to_tang->  #11    [ 14 @ C_CLOSED, SYN_SENT, LISTEN, SYN_RCVD, SYN_ACK_BY_S ]
  -Timed-> connect_op (enabl=14) -to_tang->  #12    [ 13 @ C_CLOSED, 2 @ SYN_BY_C, 2 @ SYN_SENT, 2 @ LISTEN ]
```

(b) TRS

Figura 28. TRG e TRS

```
>>    to mark #8565, weight #154, 1 entries
      from mark #8561, rate #198
>>    to mark #8566, weight #154, 3 entries
      from mark #8562, rate #154
      from mark #8563, rate #126
      from mark #8564, rate #198
>>    to mark #8567, weight #198, 2 entries
      from mark #8564, rate #198
      from mark #8565, rate #154
>>    to mark #8568, weight #154, 2 entries
      from mark #8566, rate #154
      from mark #8567, rate #198
>>    to mark #8569, weight #154, 1 entries
      from mark #8567, rate #198
>>    to mark #8570, weight #154, 2 entries
      from mark #8568, rate #154
      from mark #8569, rate #154
>>    to mark #8571, weight #6, 1 entries
      from mark #8570, rate #154
    229 rate values:
Double# 1 -> 0.842105
Double# 2 -> 0.157895
Double# 3 -> 0.851064
Double# 4 -> 0.148936
Double# 5 -> 5.333333
Double# 6 -> 1.000000
Double# 7 -> 0.459770
Double# 8 -> 0.080460
```

Figura 29. Porzione EMC del TCP

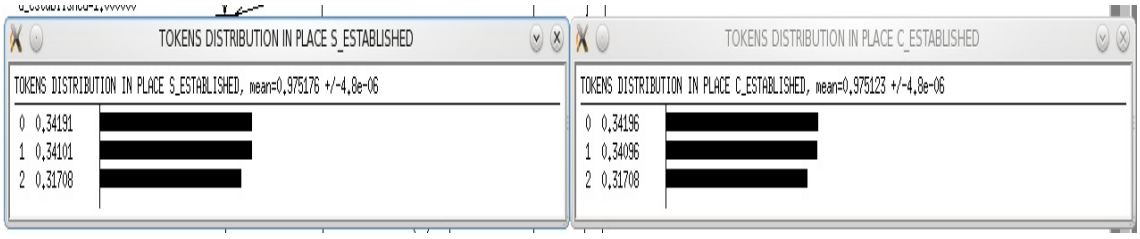


Figura 30. Analisi in steady state, stato ESTABLISHED

```
S_CLOSED(2) C_CLOSED(11) SYN_BY_C(4) SYN_SENT(4)
S_CLOSED(2) C_CLOSED(10) SYN_BY_C(5) SYN_SENT(5)
S_CLOSED(2) C_CLOSED(9) SYN_BY_C(6) SYN_SENT(6)
S_CLOSED(2) C_CLOSED(8) SYN_BY_C(7) SYN_SENT(7)
S_CLOSED(2) C_CLOSED(7) SYN_BY_C(8) SYN_SENT(8)
S_CLOSED(2) C_CLOSED(6) SYN_BY_C(9) SYN_SENT(9)
S_CLOSED(2) C_CLOSED(5) SYN_BY_C(10) SYN_SENT(10)
S_CLOSED(2) C_CLOSED(4) SYN_BY_C(11) SYN_SENT(11)
S_CLOSED(2) C_CLOSED(3) SYN_BY_C(12) SYN_SENT(12)
S_CLOSED(2) C_CLOSED(2) SYN_BY_C(13) SYN_SENT(13)
S_CLOSED(2) C_CLOSED(1) SYN_BY_C(14) SYN_SENT(14)
S_CLOSED(2) SYN_BY_C(15) SYN_SENT(15)
.....
.....
********** A G(SYN_SENT > 0 -> A F(S_ESTABLISHED > 0)) *********
Evaluation: true
S_CLOSED(2) C_CLOSED(15)
S_CLOSED(2) C_CLOSED(14) SYN_BY_C(1) SYN_SENT(1)
S_CLOSED(2) C_CLOSED(13) SYN_BY_C(2) SYN_SENT(2)
S_CLOSED(2) C_CLOSED(12) SYN_BY_C(3) SYN_SENT(3)
S_CLOSED(2) C_CLOSED(11) SYN_BY_C(4) SYN_SENT(4)
S_CLOSED(2) C_CLOSED(10) SYN_BY_C(5) SYN_SENT(5)
S_CLOSED(2) C_CLOSED(9) SYN_BY_C(6) SYN_SENT(6)
S_CLOSED(2) C_CLOSED(8) SYN_BY_C(7) SYN_SENT(7)
S_CLOSED(2) C_CLOSED(7) SYN_BY_C(8) SYN_SENT(8)
S_CLOSED(2) C_CLOSED(6) SYN_BY_C(9) SYN_SENT(9)
S_CLOSED(2) C_CLOSED(5) SYN_BY_C(10) SYN_SENT(10)
S_CLOSED(2) C_CLOSED(4) SYN_BY_C(11) SYN_SENT(11)
S_CLOSED(2) C_CLOSED(3) SYN_BY_C(12) SYN_SENT(12)
S_CLOSED(2) C_CLOSED(2) SYN_BY_C(13) SYN_SENT(13)
S_CLOSED(2) C_CLOSED(1) SYN_BY_C(14) SYN_SENT(14)
S_CLOSED(2) SYN_BY_C(15) SYN_SENT(15)
....
....
********** A G(AC_FIN_WAIT_1 > 0 -> A F(AC_TIME_WAIT > 0)) *********
Evaluation: true
S_CLOSED(2) C_CLOSED(15)
S_CLOSED(2) C_CLOSED(14) SYN_BY_C(1) SYN_SENT(1)
S_CLOSED(2) C_CLOSED(13) SYN_BY_C(2) SYN_SENT(2)
S_CLOSED(2) C_CLOSED(12) SYN_BY_C(3) SYN_SENT(3)
S_CLOSED(2) C_CLOSED(11) SYN_BY_C(4) SYN_SENT(4)
S_CLOSED(2) C_CLOSED(10) SYN_BY_C(5) SYN_SENT(5)
S_CLOSED(2) C_CLOSED(9) SYN_BY_C(6) SYN_SENT(6)
S_CLOSED(2) C_CLOSED(8) SYN_BY_C(7) SYN_SENT(7)
S_CLOSED(2) C_CLOSED(7) SYN_BY_C(8) SYN_SENT(8)
S_CLOSED(2) C_CLOSED(6) SYN_BY_C(9) SYN_SENT(9)
S_CLOSED(2) C_CLOSED(5) SYN_BY_C(10) SYN_SENT(10)
S_CLOSED(2) C_CLOSED(4) SYN_BY_C(11) SYN_SENT(11)
S_CLOSED(2) C_CLOSED(3) SYN_BY_C(12) SYN_SENT(12)
S_CLOSED(2) C_CLOSED(2) SYN_BY_C(13) SYN_SENT(13)
```

```
S_CLOSED(2) C_CLOSED(1) SYN_BY_C(14) SYN_SENT(14)
S_CLOSED(2) SYN_BY_C(15) SYN_SENT(15)
....
....
********** A G not(SYN_RCVD > 2) *********
Evaluation: true
S_CLOSED(2) C_CLOSED(15)
S_CLOSED(2) C_CLOSED(14) SYN_BY_C(1) SYN_SENT(1)
S_CLOSED(2) C_CLOSED(13) SYN_BY_C(2) SYN_SENT(2)
S_CLOSED(2) C_CLOSED(12) SYN_BY_C(3) SYN_SENT(3)
S_CLOSED(2) C_CLOSED(11) SYN_BY_C(4) SYN_SENT(4)
S_CLOSED(2) C_CLOSED(10) SYN_BY_C(5) SYN_SENT(5)
S_CLOSED(2) C_CLOSED(9) SYN_BY_C(6) SYN_SENT(6)
S_CLOSED(2) C_CLOSED(8) SYN_BY_C(7) SYN_SENT(7)
S_CLOSED(2) C_CLOSED(7) SYN_BY_C(8) SYN_SENT(8)
S_CLOSED(2) C_CLOSED(6) SYN_BY_C(9) SYN_SENT(9)
S_CLOSED(2) C_CLOSED(5) SYN_BY_C(10) SYN_SENT(10)
S_CLOSED(2) C_CLOSED(4) SYN_BY_C(11) SYN_SENT(11)
S_CLOSED(2) C_CLOSED(3) SYN_BY_C(12) SYN_SENT(12)
S_CLOSED(2) C_CLOSED(2) SYN_BY_C(13) SYN_SENT(13)
S_CLOSED(2) C_CLOSED(1) SYN_BY_C(14) SYN_SENT(14)
S_CLOSED(2) SYN_BY_C(15) SYN_SENT(15)
....
....
```

Per comodità l'elenco di tutti gli stati non è stata riportato. Tramite il model cheker MC4CSL^{TA} sono stati effettuati i seguenti test di verifica:

./DSPN-Tool-Release -load /Users/chiaracontoli/Desktop/Magistrale/.../Model/casoTcp/tcp -cslta #C_ESTABLISHED>0 && #S_ESTABLISHED>0 -wrsat statfiletcp1.txt
il cui output con i valori di verità per ogni stato della catena di Markov sono visibili nel file statfiletcp1.txt
./DSPN-Tool-Release -load /Users/chiaracontoli/Desktop/Magistrale/.../Model/casoTcp/tcp -dta-path ../DTA/ -cslta PROB_TA > 0.10 until_0B (3 | | #SYN_SENT>0, #C_ESTABLISHED>0) -wrsat tcp_ex1.txt
il cui output è visibile nel file tcp_ex1.txt in cui, oltre al valore di verità è riportato anche la probabilità che essa assume in ogni stato; mentre in output viene visualizzato quanto segue:

```
THE RG IS A CONTINUOS TIME MARKOV CHAIN

MODEL CHECKING CSLTA EXPRESSION:
  PROB_TA > 0.1 "until_0B" (3 | | (#SYN_SENT > 0), (#C_ESTABLISHED > 0))


LOADING DTA FILE "../DTA//until_0B.dta" ...
DTA PARTITIONED INTO ZONES.
  <S,L,C> STATES IN MxA: 1
```

```
NUMBER OF TANGIBLE <S*L*C> STATES:
5527 (INCLUDING TOP, BOT)
NUMBER OF CTMC STATES:                  8571
NUMBER OF DTA LOCATIONS:                6
NUMBER OF TIMED CLOCK ZONES:            2
NUMBER OF DETERMINISTIC TRANSITIONS:    1
NONZERO ENTRIES IN Q:                   24798
NONZERO ENTRIES IN Qbar:                1095
NONZERO ENTRIES IN DELTA:               5525
NON-TRIVIAL INITIAL STATES:             5525

THERE ARE 2 REGENERATIVE STATES.

5525 SUBORDINATED MARKOV CHAINS WILL BE COMPUTED.
COMPUTING SMC 1/5525...
COMPUTING SMC 51/5525...
COMPUTING SMC 100/5525...
COMPUTING SMC 151/5525...
COMPUTING SMC 201/5525...
COMPUTING SMC 249/5525...
COMPUTING SMC 298/5525... 27s remained.
COMPUTING SMC 346/5525... 26s remained.
COMPUTING SMC 396/5525... 26s remained.
COMPUTING SMC 446/5525... 26s remained.
....
....
P MATRIX (5527 x 5527) HAS 11050 NONZERO ENTRIES.
C MATRIX (5527 x 5527) HAS 4990490 NONZERO ENTRIES.
EMC BUILD TIME: [User 29.707s, Sys 1.378s]

BACKWARD STEADY STATE SOLUTION.
FORWARD GAUSS-SEIDEL: iteration=1, err=3.65959e-04

FORWARD GAUSS-SEIDEL: convergence in 4 iterations, err=4.34587e-08

BACKWARD STEADY STATE SOLUTION OF THE MxA PROCESS COMPUTED.

RESULT: COMPUTED FOR ALL THE 8571 INTIAL STATES.

SOLUTION TIME: [User 30.395s, Sys 1.560s]
MODEL CHECKING COMPLETED.

MODEL CHECKING SOLUTION WRITTEN TO tcp_ex1.txt.
```

./DSPN-Tool-Release -load /Users/chiaracontoli/Desktop/Magistrale/.../Model/casoTcp/tcp -dta-path ../DTA/ -cslta PROB_TA > 0.10 until_0B (3 | | #SYN_SENT>0, #S_ESTABLISHED>0) -wrsat tcp_server_ex1.txt

./DSPN-Tool-Release -load /Users/chiaracontoli/Desktop/Magistrale/.../Model/casoTcp/tcp -dta-path ../DTA/ -cslta PROB_TA > 0.10 until_0B (3 | | #AC_FIN_WAIT_1>0, #AC_TIME_WAIT>0) -wrsat tcp_ex2.txt

A partire dal caso precedente è possibile pensare di realizzare una modifica per poter stimare il numero di richieste di connessione che non vengono soddisfatte; questo quando il numero di server è molto inferiore rispetto al numero di richieste che arrivano. Per ragioni di tempi di elaborazione, dovuti principalmente al problema dell'esplosione dello spazio degli stati da analizzare, è stata fatta un'analisi in steady state nel caso in cui vi sono 5 token che agiscono come client e un token che agisce come server; in figura 31 è riportata la rete, in cui ogni client termina non appena viene chiusa la connessione, mentre i server tornano disponibili non appena la chiudono la connessione precedente. A regime tutte le richieste verrebbero soddisfatte, poiché le richieste di connessione vengono accodate (virtualmente all'infinito) nello stato SYN_BY_C; per emulare un sistema in cui, dopo un certo tempo di attesa, a fronte di una richiesta di connessione questa non viene soddisfatta, si è scelto di aggiungere un'ulteriore transizione che scatta solamente quando tutti i client hanno emesso la richiesta di connessione e i server sono nello stato di CLOSED. In questo scenario, l'analisi delle transition invariants non è verificata, poiché la rete, essendovi un deadlock, non ritornerà mai nella configurazione iniziale.
La figura 32 mostra a regime il numero medio di token che identificano le richieste non soddisfatte, con la relativa probabilità. Per questo scenario sono state analizzate le medesime proprietà del caso precedente, con in più

E F (CLOSED > 0)
E F(C_CLOSED = 0 and SYN_SENT > 0 and S_CLOSED = 0)

La prima indica verifica il fatto che prima o poi i client che hanno in precedenza stabilito una connessione, poi la chiudono anche; mentre la secondo indica verifica il fatto che non tutte le richieste di connessione saranno soddisfatte.
Con il MC4CSL^{TA} è stato eseguito:
./DSPN-Tool-Release -load /Users/chiaracontoli/Desktop/.../Model/casoTcp/tcp -dta-path ../DTA/ -cslta PROB_TA > 0.10 until_0B (10 | | #C_CLOSED=0, #SYN_SENT>0) -wrsat tcp_extension_ex1.txt

5 MRMC e INFAMY: un piccolo confronto con GreatSPN ed MC4CSL^{TA}

MRMC è anch'esso un model checker per sistemi con possibilità di modeling per sistemi Discrete time Markov chains (DTMCs), Continuous time Markov chains (CTMCs), Discrete time Markov Reward models (DMRMs), Continuous time Markov Reward models (CMRMs), Continuous time Markov decision processes (CTMDPIs). Il modello viene scritto fornendo in ingresso cinque diversi tipi di file, anche se quelli obbligatori sono i file con estensione *.lab*, *.tra* e *.ctmdpi*

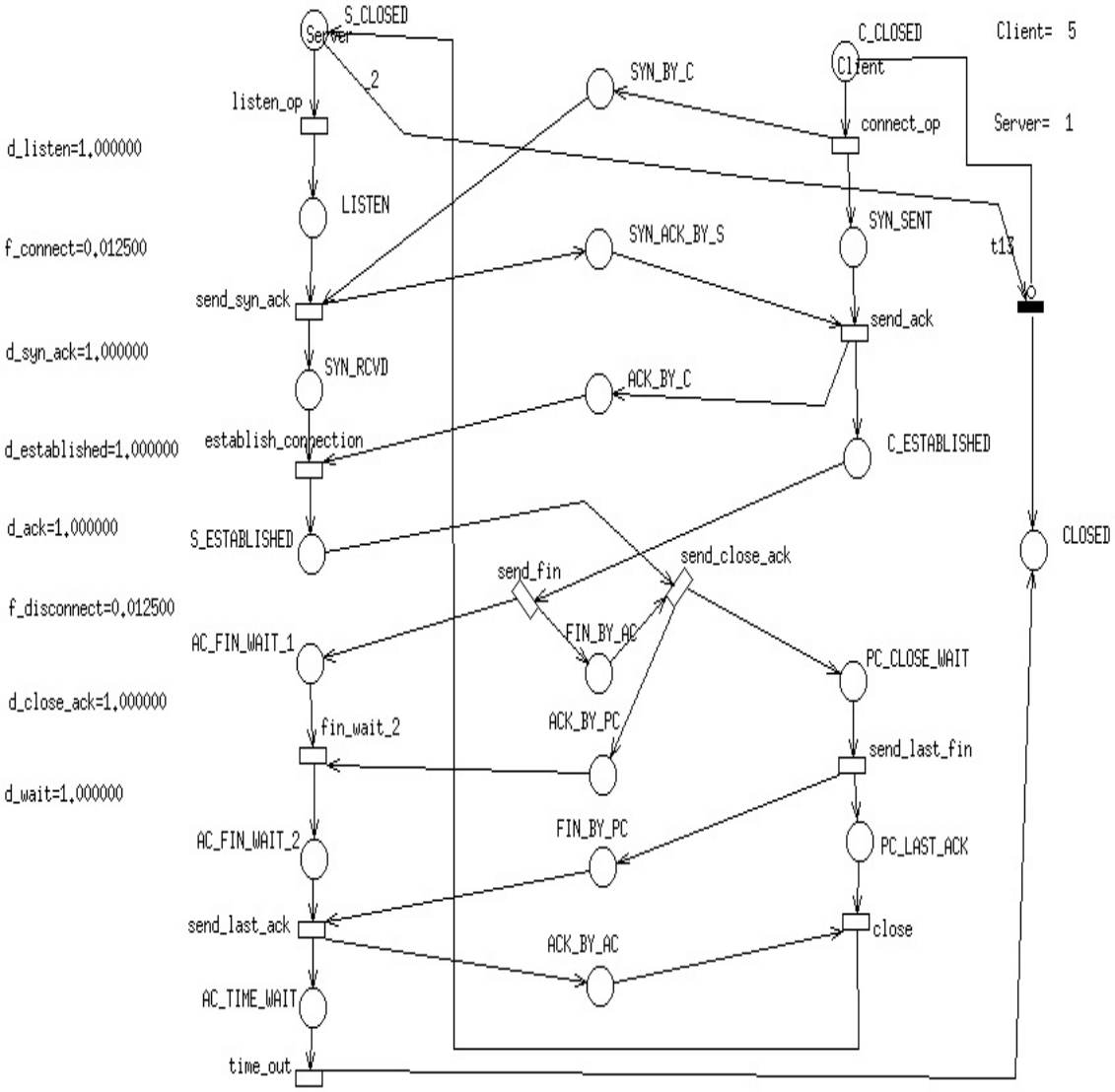


Figura 31. Protocollo TCP: caso con perdita 1

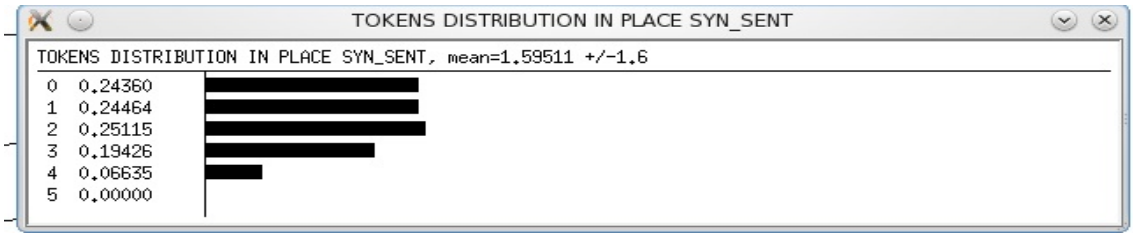


Figura 32. Protocollo TCP con perdita: analisi steady state

. Il file con estensione *.tra* descrive la matrice di probabilità o frequenze di un DTMC, CTMC o MRM, e il *.lab* contiene alcune label che identificano gli stati con proposizioni atomiche. Gli altri tre file che possono essere specificati hanno estensione *.ctmdpi*, file contenente la matrice dei rate e le label delle transizioni per un sistema CTMDPI; estensione *.rew*, associa agli stati un valore di "ricompensa" , e *.rewi* che specifica l'impulse-reward. Di seguito viene mostrato come deve essere il formato dei file, formato che no viene controllato da MRMC. Formato file .tra:

```
Tra_File =  Header Body
Header   =   'STATES' <number of states> \n
Body = 'TRANSITIONS' <number of transitions> \n
  <from state> <to state> <rate/probability> \n
   Body
| <from state> <to state> <rate/probability> \n
```

Formato file .lab:

```
Lab_File = Declaration Body
Declaration =  '#DECLARATION' \n
  Atomic_Prop_List \n
  '#END' \n
 Body =  <state> Atomic_Prop_List \n Body
| <state> Atomic_Prop_List \n
Atomic_Prop_List =   <atomic proposition> Atomic_Prop_List
| <atomic proposition>

<atomic proposition> =  {let}{alnum}*
let                  =  [_a-zA-Z]
alnum                =  [_a-zA-Z0-9<>_^*+-=]

<state> Atomic_Prop_Lists
```

Formato file .ctmdpi:

```
Ctmdpi_File =   Header Body_Int_Nondet
Header =   'STATES' <number of states> \n
    '#DECLARATION' \n
     Atomic_Prop_List \n
```

```
    '#END' \n
Body_Int_Nondet =   <from state> <label> \n
    * <to state> <rate> \n
    { * <to state> <rate> } \n
    Body_Int_Nondet
  | <from state> <label> \n
    * <to state> <rate> \n
    { * <to state> <rate> } \n
```

Definire tutti e cinque i file, o anche solo i tre obbligatori non è cosa semplice, poiché non è intuitivo pensare di modellare un sistema in questo modo. Se poi il sistema da modellare è particolarmente difficile, questo contribuisce ad aumentare la complessità della modellazione. É meglio avere tool che consentono una modellazione di più alto livello, più intuitiva che hanno un opportuno linguaggio di modellezione, sia esso grafico (come GreatSPN) o "commando-oriented" come ad esempio *INFAMY*. INFAMY è un tool per il model checking di formule CSL pensato per catene di Markov tempo continue infinite, specificate in una variante del linguaggio usato da PRISM, ovvero usando interi "unbounded", cioè a range infinito.
Una possibile rappresentazione del modello adottato negli esempi è riportata di seguito:

```
stochastic

const lambda1 = 0.5; //arrival frequency of UT1
const lambda2 = 0.5; //arrival frequency of UT2
const mu1 = 0.75; //service frequency of UT1
const mu2 = 0.25; //service frequency of UT2
const NU = 10;

module mm1_prio

        //places
        ut1: int; //user of type 1 (highest priority)
        ut2: int; //user of type 2
        server: int; //resource shared between ut1 and ut2
        wq1: int; //queue of ut1
        wq2: int; //queue of ut2
        servingT1: int; //ut1 is in service
        servingT2: int; //ut2 is in service

        //transitions
        [] ut1> 0 -> lambda1: (ut1'=ut1-1) & (wq1'=wq1+1);
        [] ut2> 0 -> lambda2: (ut2'=ut2-1) & (wq2'=wq2+1);
        [] wq1> 0 & server=1 -> 1.0: (wq1'=wq1-1) & (server'=server-1) &
        (servingT1'=servingT1+1);
        [] wq2> 0 & wq1=0 & server=1 -> 1.0: (wq2'=wq2-1) & (server'=server-1) &
         (servingT2'=servingT2+1);
        [] servingT1 > 0 -> mu1: (servingT1'=servingT1-1) & (server'=server+1) &
        (ut1'=ut1+1);
```

```
        [] servingT2 > 0 -> mu2: (servingT2'=servingT2-1) & (server'=server+1) &
        (ut2'=ut2+1);

endmodule

//marking M0
init
        ut1 = NU &
        ut2 = NU &
        server = 1 &
        wq1 = 0 &
        wq2 = 0 &
        servingT1 = 0 &
        servingT2 = 0
endinit
```

Il model checker è eseguibile da linea di comando tramite *infamy-32 nomefile.sm nome.csl*, dove il file con estensione *.sm* rappresenta il modello, e il file con estensione *.csl* contiene le formule che si vogliono verificare. L'output viene visualizzato su terminale, come mostra l'esempio di seguito. Il file .csl contiene le seguenti formule:

```
P=?[ wq1>=0 U<=3.0 servingT1=1 ]
P=?[ wq1>=0 U<=2.0 servingT1=1 ]
P=?[ wq2>=0 U<=3.0 servingT2=1 ]
P=?[ wq1>=0 & wq2 >= 0 U<=3.0 servingT1=1 & servingT2=1]
P > 0.1 [ wq1>=0 U<=10.0 servingT1=1 ]
```

il cui risultato è:

```
Handling property "Pmax=?[ (wq1 >= 0) U <=3 (servingT1 = 1) ]"
RESULT[0] = 0.537375

model construction = 0.508059 seconds
analysis time = 0.00200398 seconds
Handling property "Pmax=?[ (wq1 >= 0) U <=2 (servingT1 = 1) ]"
RESULT[0] = 0.370157

model construction = 0.502235 seconds
analysis time = 0.00567706 seconds
Handling property "Pmax=?[ (wq2 >= 0) U <=3 (servingT2 = 1) ]"
RESULT[0] = 0.351979

model construction = 0.512359 seconds
analysis time = 0.00176001 seconds
Handling property "Pmax=?[ ((wq1 >= 0) AND (wq2 >= 0)) U <=3 ((servingT1 = 1) AND (se
RESULT[0] = 0

model construction = 0.48122 seconds
analysis time = 0.00194299 seconds
Handling property "Pmax> 0.1[ (wq1 >= 0) U <=10 (servingT1 = 1) ]"
```

```
true

model construction = 0.528363 seconds
analysis time = 0.00187703 seconds
```

6 Conclusioni

GreatSPN è un tool grafico che permette la modellazione di sistemi, anche complessi, con il formalismo delle reti di Petri, comprese le estensioni di archi inibitori, reti temporizzate (reti stocastiche, cioè ritardi stabiliti probabilisticamente) e reti colorate (associano un simbolismo e regole computazionali alle transizioni che producono e consumano i tokens). Oltre alla modellazione fornisce anche strumenti di analisi, sia tramite GUI che da linea di comando. L'analisi è orientata sia alla valutazione delle performance, che alla verifica formale di importanti proprietà che il sistema deve soddisfare. GreatSPN viene accompagnato da un model checker a linea di comando, ord_rgMEDD, che consente la valutazione di proprietà secondo la Computational Tree Logic. Oltre a questo model checker, vi è anche MC4CSL^{TA}, tool a linea di comando che, pur non essendo ancora integrato stabilmente con GreatSPN, sta evolvendo e maturando per implementare le funzionalità che già sono a disposizione di GreatSPN. GreatSPN offre buona flessibilità nello scrivere il modello ed MC4CSL^{TA}, pur nel suo stato "embrionale" per certi aspetti, offre già buone possibilità di analisi per sistemi associabili a catene di Markov.

Riferimenti bibliografici

1. GreatSPN 2.0
   `http://www.di.unito.it/ greatspn/index.html`.
2. MC4CSLA^{TA}
   `http://www.di.unito.it/ amparore/mc4cslta/`.
3. Elvio Gilberto Amparore e Susanna Donatelli
   `Improving and Assessing the Efficiency of the MC4CSL`^{TA} `Model Checker`.
4. MRMC
   `http://www.mrmc-tool.org/trac/`
5. INFAMY
   `http://depend.cs.uni-saarland.de/tools/infamy/`
6. Ernst Moritz Hahn, Holger Hermanns, Bjorn Wachter, and Lijun Zhang
   `INFAMY: An Infinite-State Markov Model` $Checker^?$