RELAZIONE ESERCIZIO DI MODELLAZIONE CON GREATSPN E NUSMV E ESERCIZIO DI CONFRONTO CON ALGEBRA DEI PROCESSI

Laboratorio lungo 9 CFU

Marco Edoardo Santimaria
Matricola 912404

6 luglio 2023

Indice

1 Algoritmo 3.2 **3**
- 1.1 NuSMV . . . 3
- 1.2 Rete di Petri . . . 4
  - 1.2.1 Modello . . . 4
  - 1.2.2 Reachability graph . . . 5
  - 1.2.3 Analisi strutturale . . . 5
- 1.3 Model checking CTL . . . 6

2 Algoritmo 3.6 **8**
- 2.1 NuSMV . . . 8
- 2.2 Rete di Petri . . . 10
  - 2.2.1 Modello . . . 10
  - 2.2.2 Rechability Graph . . . 10
  - 2.2.3 Analisi strutturale . . . 11
  - 2.2.4 Riduzione della rete . . . 11
- 2.3 Model checking CTL . . . 11
- 2.4 Process Algebra . . . 14

3 Algoritmo 3.8 **15**
- 3.1 NuSMV . . . 15
- 3.2 Rete di Petri . . . 17
  - 3.2.1 Modello . . . 17
  - 3.2.2 Rechability graph . . . 17
  - 3.2.3 Analisi strutturale . . . 17
  - 3.2.4 Riduzione della rete . . . 18
- 3.3 Model checking CTL . . . 18
- 3.4 Process Algebra . . . 19

4 Calcolo bisimulazione tra algoritmi 3.6 e algoritmi 3.8 **20**

5 Algoritmo 3.10 **21**
- 5.1 NuSMV . . . 21
- 5.2 Rete di Petri . . . 24
  - 5.2.1 Modello . . . 24
  - 5.2.2 Rechability graph . . . 25
  - 5.2.3 Analisi strutturale . . . 25
- 5.3 Model checking CTL . . . 25

6 Confronti **25**
- 6.1 Tabella di confronti . . . 25
- 6.2 Commenti sui confronti . . . 26
  - 6.2.1 Fairness . . . 26
- 6.3 Modifica requisito progresso in sezione critica . . . 26

Disclaimer: per quanto abbia tentato di mettere le varie immagini nella posizione corretta mi e' stato quasi impossibile raggiungere il risultato sperato. Pertanto ho incluso dei link ipertestuali e referenze in modo che se l'immagine non si trovi dove ci si aspetti, si possa comunque recuperare l'immagine facilmente.

1 Algoritmo 3.2

1.1 NuSMV

Il listato che modella l'algoritmo 3.2 e' il seguente:

```
MODULE main
VAR
    turno: {p, q};
    proc_p: process proc(p, turno);
    proc_q: process proc(q, turno);
ASSIGN
    init(turno) := p;



 --mutua esclusione
SPEC AG(proc_p.stato=l3 -> proc_q.stato!=l3) & AG(proc_q.stato=l3 -> proc_p.stato!=l3);

--assenza di deadlock
SPEC AG EF proc_p.stato=l1 |  AF EG proc_q.stato=l1;

--assenza di deadlock come definita nelle slides
SPEC AG ((proc_p.stato=l2 | proc_q.stato=l2) -> AF (proc_p.stato=l3 | proc_q.stato=l3));

--template del mio processo in nusmv
MODULE proc(identita, turno)
VAR
    stato : {l1, l2, l3, l4}; --l1 come per dire linea 1 del codice originale
ASSIGN
    init(stato) := l1;
    next(stato) :=
        case
            stato = l1: {l1, l2};
            stato = l2 & turno=identita: l3;
            stato = l3: l4;
            stato = l4: l1;
            TRUE: stato;
        esac;

    next(turno):=
                case
                    stato=l4 & turno=identita & identita=p: q;
                    stato=l4 & turno=identita & identita=q: p;
                    TRUE: turno;
                esac;
FAIRNESS running;
```

```
--assenza di starvation individuale
SPEC AG(stato=l2 -> AF stato=l3);
```

Il modello dell'algoritmo e' stato realizzando cercando di modellare in modo da essere 1:1 con lo pseudo codice fornito dal testo dell'esercizio. Per questo motivo i nomi degli stati che il modello NuSMV può avere, iniziano con la lettera "l", dove l sta per indicare la linea di codice identificata dal numero successivo. Questo approccio, seppur magari di non semplice comprensione a prima vista, e' stato scelto in quanto in ogni momento si ha un confronto diretto con il punto di esecuzione raggiunto all'interno dello pseudo codice.
Un altro punto su cui voglio far prestare attenzione e' il fatto di aver modellato un singolo processo e non due. Questo e' stato possibile grazie ad aver aggiunto un parametro nel modulo chiamato identità. Questo parametro va a specificare che tipo di processo sia (se processo p o processo q). Questo accorgimento ha consentito di limitare la dimensione del codice NuSMV ottenendo un risultato uguale a una modellazione con due processi diversi.

Per quanto riguarda le clausole di fairness, richieste per la correttezza dell'algoritmo sono state modellate nel seguente modo:

1. **Assenza di starvation individuale**: Questa specifica e' stata modellata usando CTL nel seguente modo:

   ```
   SPEC AG(stato=l2 -> AF stato=l3);
   ```

   ed e' da interpretare nel seguente modo: "Ogni qualvolta un processo richiede di entrare nella sezione critica, vi entrerà in futuro". Essendo un vincolo che riguarda il singolo processo, ho deciso di inserirla all'interno del modulo proc().
   Facendo model checking su questa specifica, in questo modello, essa viene valutata come **FALSA**. Questo perché' e' possibile che un processo faccia richiesta di entrare nella sezione critica senza che sia il suo turno, e l'altro processo rimane in loop nella sezione non critica.

2. **Mutua esclusione**: Questa specifica e' stata modellata nel seguente modo:

   ```
   SPEC AG(proc_p.stato=l3 -> proc_q.stato!=l3)
   & AG(proc_q.stato=l3 -> proc_p.stato!=l3);
   ```

   Essendo una specifica che richiede di andare a verificare lo stato dei due processi e' stato richiesto di inserire la specifica all'interno del modulo main. Questa specifica viene modellata come **VERA**.

3. **Assenza di deadlock (definizione da slides)**. Questa specifica e' stata modellata come

   ```
   SPEC AG ((proc_p.stato=l2 | proc_q.stato=l2)
   -> AF (proc_p.stato=l3 | proc_q.stato=l3));
   ```

   Essendo una specifica che richiede di andare a verificare lo stato dei due processi e' stato richiesto di inserire la specifica all'interno del modulo main. Questa specifica viene modellata come **FALSA**. Questo perché è possibile che il processo P sia nello stato di richiesta di ingresso nella sezione critica, ma il turno sia q, e il processo q abbia una esecuzione infinita nella sezione non critica. Questa situazione rende appunto falsa la condizione.

4. **Assenza di deadlock**:Questa specifica e' stata modellata nel seguente modo:

   ```
   SPEC AG EF proc_p.stato=l1 | AG EF proc_q.stato=l1;
   ```

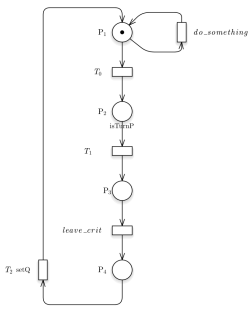
   Essendo una specifica che richiede di andare a verificare lo stato dei due processi e' stato richiesto di inserire la specifica all'interno del modulo main. Questa specifica viene modellata come **VERA**.

Per concludere, questa prima approssimazione dell'algoritmo di dekker, soffre di un problema, ovvero la presenza di **starvation individuale**.

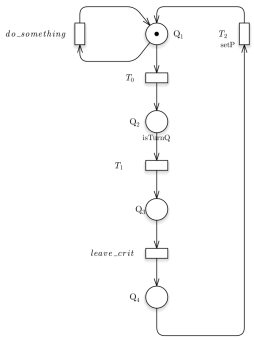
1.2 Rete di Petri

1.2.1 Modello

Il modello e' stato realizzato per composizione. In particolare, nell'immagine sotto, si possono vedere i tre componenti (producer, consumer, e variabile turn) oltre che alla rete risultante della composizione:



(a) Processo p algoritmo 3.2



(b) Processo q algoritmo 3.2

- process p 1a
- process q 1b
- turn 2
- rete completa3

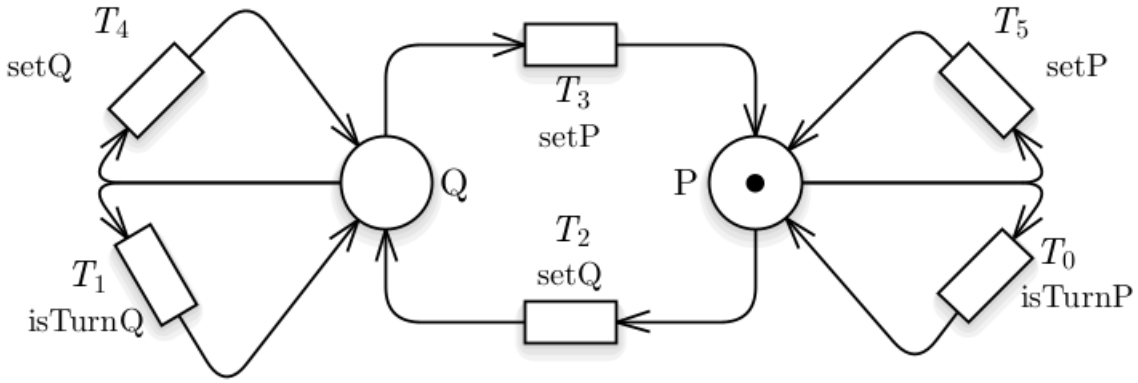


Figura 2: variabile booleana turn algoritmo 3.2

1.2.2 Reachability graph

Il reachability graph (4) ottenuto dalla rete e' qua sotto presentato. Ha 16 stati e non presenta situazioni di deadlock (come ci si voleva aspettare dalla modellazione in NuSMV). Il reachability graph presenta 16 marcature raggiungibili, e la marcatura iniziale m_0 e' un home state. La rete e' quindi reversibile.

1.2.3 Analisi strutturale

GreatSPN ha calcolato 3 semi-flussi minimali. Da questo si evincono due aspetti: la rete e' bounded e il bound di ogni posto corrisponde a 1 token.
GreatSPN ha calcolato 5 diversi t semiflow minimali. Da essi si può verificare che la rete e' reversibile (Dimostrato anche dalle proprietà calcolate da greatSPN, essendo m0 un home state).

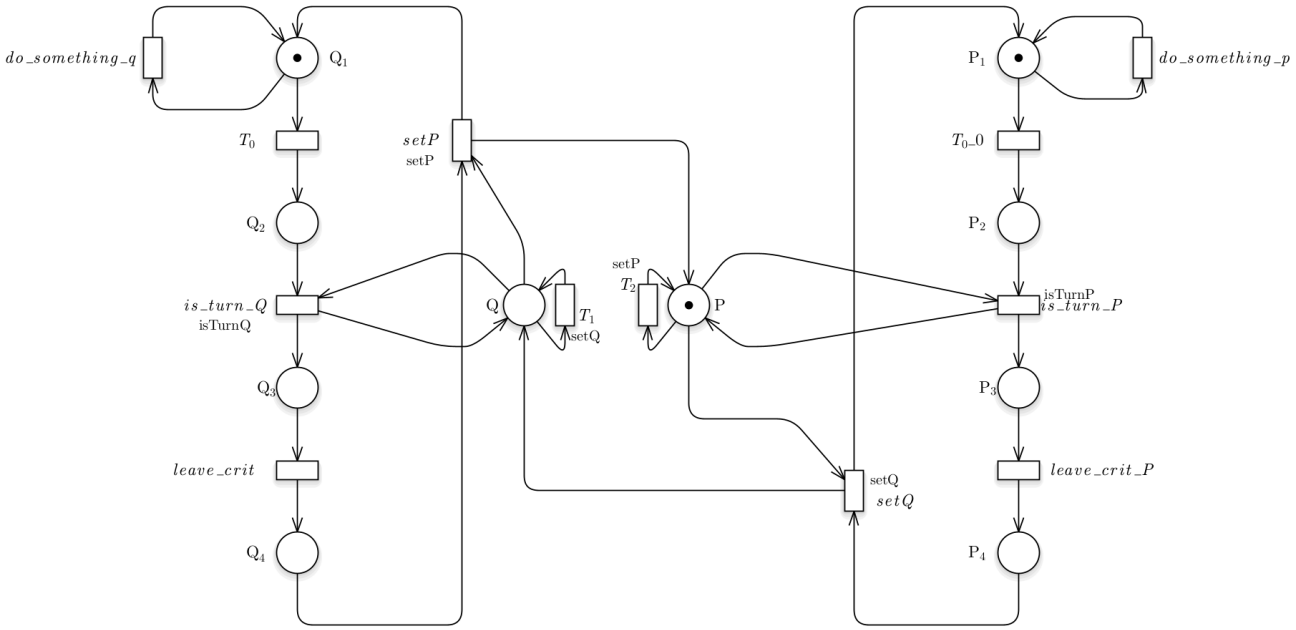


Figura 3: Rete algoritmo 3.2

1.3 Model checking CTL

La modellazione delle proprietà di correttezza dell'algoritmo, e' stata fatta in modo che sia il più possibile uguale a quella fatta in NuSMV. Anche greatSPN conferma il risultato che si ha ottenuto con NuSMV, ovvero che l'algoritmo soffre di starvation individuale, come mostrato dallo screenshot della pagina delle measures5.

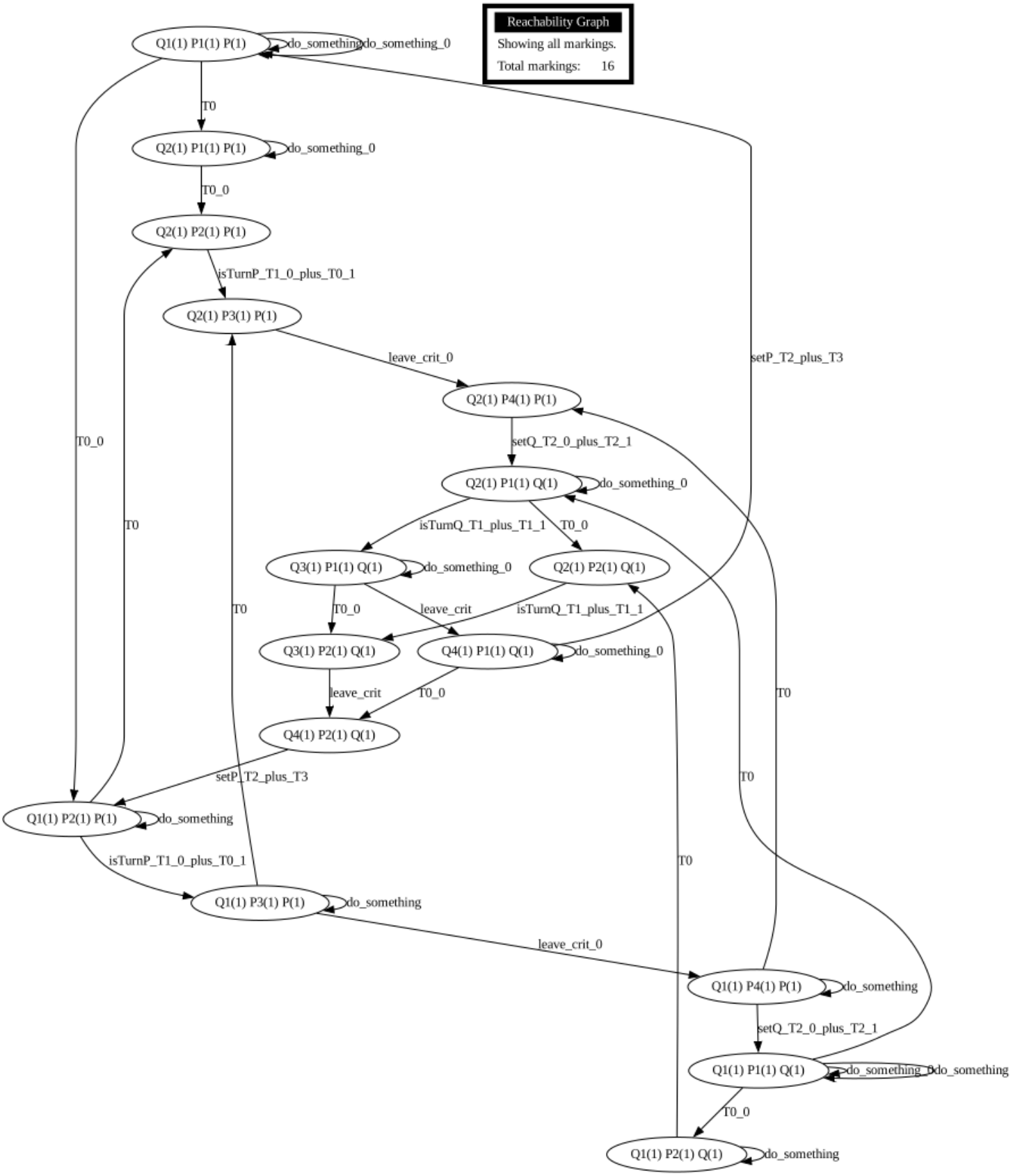


Figura 4: Reachability Graph algoritmo 3.2

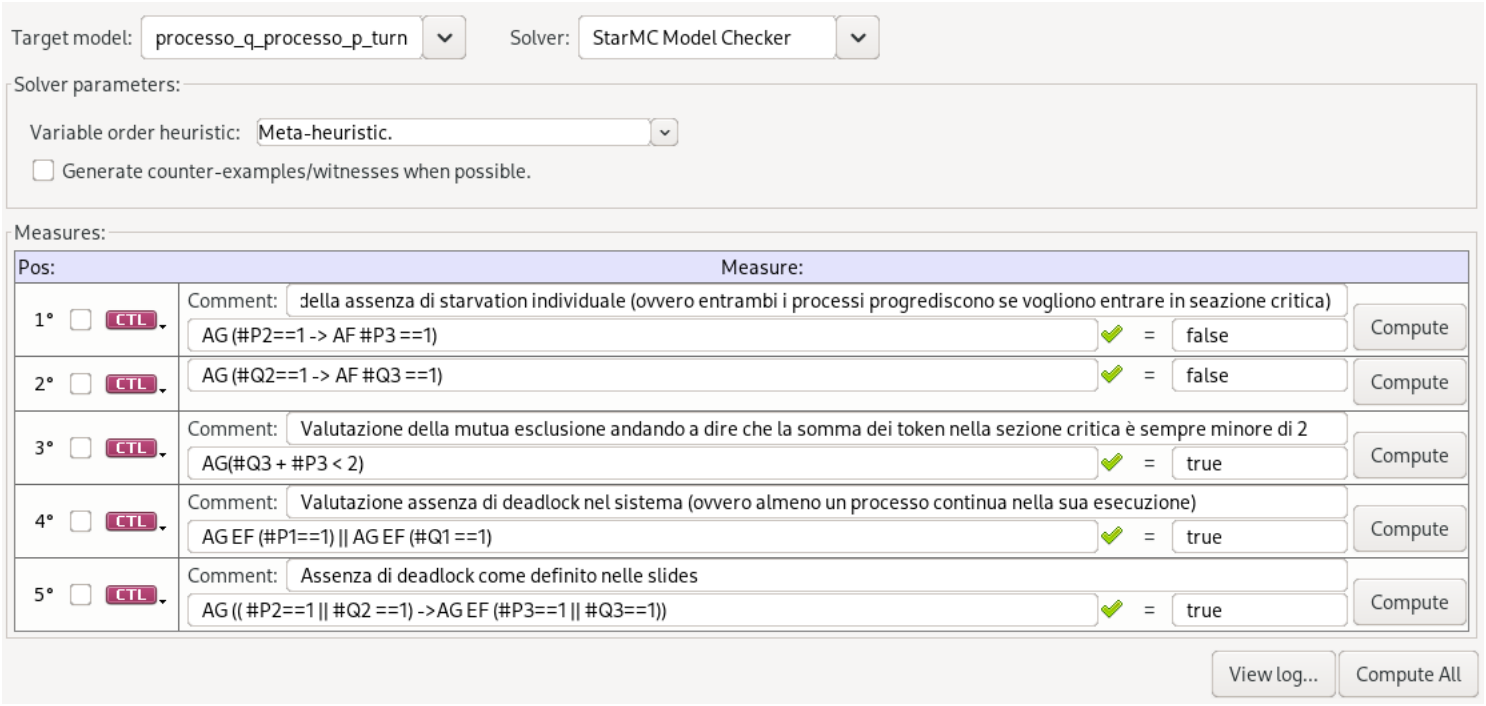


Figura 5: Verifica CTL algoritmo 3.2

2 Algoritmo 3.6

2.1 NuSMV

Il listato dell'algoritmo 3.6 e' il seguente:

```
 MODULE main
VAR
    want_p_var: boolean;
    want_q_var: boolean;


    proc_p: process proc(want_p_var, want_q_var);
    proc_q: process proc(want_q_var, want_p_var);

ASSIGN
    init(want_p_var):=FALSE;
    init(want_q_var):=FALSE;

 --mutua esclusione
SPEC AG(proc_p.stato=l3 -> proc_q.stato!=l3) & AG(proc_q.stato=l3 -> proc_p.stato!=l3);

--assenza di deadlock
SPEC AG EF proc_p.stato=l1 |  AG EF proc_q.stato=l1;

--assenza di deadlock come definita nelle slides
SPEC AG ((proc_p.stato=l2 | proc_q.stato=l2) ->
AF (proc_p.stato=l4 | proc_q.stato=l4));
```

```
MODULE proc( want_io, want_altro)
VAR
    stato : {l1, l2, l3, l4, l5};
ASSIGN
    init(stato) := l1;
    next(stato) :=
        case
            stato=l1: {l1, l2};
            stato=l2 & want_altro=FALSE:l3;
            stato=l3: l4;
            stato=l4: l5; --sezione critica
            stato=l5: l1;
            TRUE: stato;
        esac;

    next(want_io) :=
        case
            stato=l3:TRUE; -
            stato=l5: FALSE;
            TRUE:want_io;
        esac;


FAIRNESS running;
SPEC AG( stato=l2 -> AF stato=l3 );
```

Anche in questo listato vale la stessa considerazione per il nome assegnato agli stati del sistema e per l'aver modellato un singolo processo.
Per quanto riguarda le clausole di fairness, richieste per la correttezza dell'algoritmo sono state modellate nel seguente modo:

1. **Assenza di starvation individuale**: Questa specifica e' stata modellata usando CTL nel seguente modo:

   ```
   AG( stato=l2 -> AF stato=l3 );
   ```

   ed e' da interpretare nel seguente modo: "Ogni qualvolta un processo richiede di entrare nella sezione critica, vi entrerà in futuro". Essendo un vincolo che riguarda il singolo processo, ho deciso di inserirla all'interno del modulo proc().
   Facendo model checking su questa specifica, in questo modello, essa viene valutata come **FALSA**.

2. **Mutua esclusione**: Questa specifica e' stata modellata nel seguente modo:

   ```
   SPEC AG(proc_p.stato=l3 -> proc_q.stato!=l3) &
   AG(proc_q.stato=l3 -> proc_p.stato!=l3);
   ```

   Essendo una specifica che richiede di andare a verificare lo stato dei due processi e' stato richiesto di inserire la specifica all'interno del modulo main. Questa specifica viene valutata come **FALSA**. Questo perché' esiste una sequenza di esecuzione, che porta entrambi i processi a essere in sezione critica (NuSMV fornisce un esempio di esecuzione, ma con il tool greatSPN risulta più semplice notare la situazione anomala).

3. **Assenza di deadlock (definizione da slides)**. Questa specifica e' stata modellata come

   ```
   SPEC AG ((proc_p.stato=l2 | proc_q.stato=l2) -> AF (proc_p.stato=l4 | proc_q.stato=l4
   ```

   Essendo una specifica che richiede di andare a verificare lo stato dei due processi e' stato richiesto di inserire la specifica all'interno del modulo main. Questa specifica viene modellata come **VERA**.

4. **Assenza di deadlock**:Questa specifica e' stata modellata nel seguente modo:

```
SPEC AG EF proc_p.stato=l1 | AG EF proc_q.stato=l1;
```

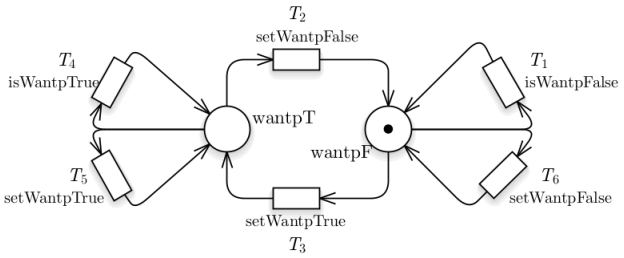
Essendo una specifica che richiede di andare a verificare lo stato dei due processi e' stato richiesto di inserire la specifica all'interno del modulo main. Questa specifica viene valutata come **VERA**.

2.2 Rete di Petri

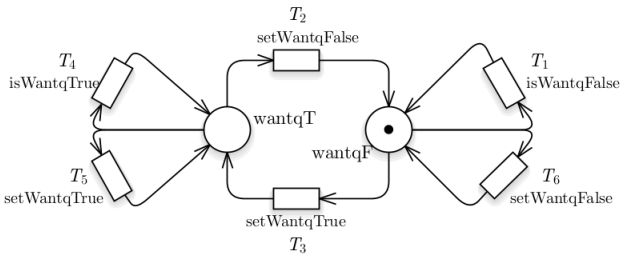
2.2.1 Modello

La rete è stata modellata con le seguenti sotto componenti:

- Variabile booleana want p6a
- Variabile booleana want q6b
- processo P7b
- processo q7a
- rete finale realizzata per composizione8



(a) Variabile booleana Want p algoritmo 3.6

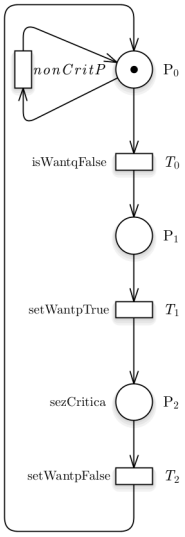


(b) Variabile booleana Want q algoritmo 3.6

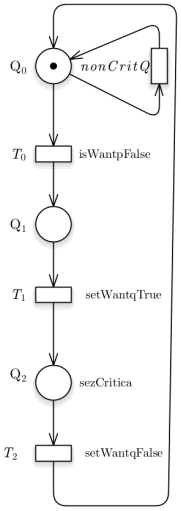
2.2.2 Rechability Graph

Il reachability graph9 ha 9 stati e non presenta situazioni di deadlock (come ci si voleva aspettare dalla modellazione in NuSMV). Il reachability graph presenta 9 marcature raggiungibili, e la marcatura iniziale m0 e' un home state. La rete e' quindi reversibile.

Anche in questo caso ci si trova in cui il reachability graph della rete di petri ha meno stati rispetto al modello in nusmv. Anche qua si tratta della stessa motivazione: il modello greatspn ha 3 posti, mentre il modello nusmv ha 5 stati.



(a) Processo p algoritmo 3.6



(b) Processo q algoritmo 3.6

2.2.3 Analisi strutturale

L'analisi e' stata effettuata sulla rete non ridotta. GreatSPN ha calcolato 8 semiflussi minimali. Da questo si evincono due aspetti: la rete e' bounded e il bound di ogni posto corrisponde a 1 token.
GreatSPN ha calcolato 8 diversi t semiflow minimali. Da essi si può verificare che la rete e' reversibile (Dimostrato anche dalle proprietà calcolate da greatSPN, essendo m0 un home state).
La motivazione per cui il modello greatspn abbia meno stati del modello in nusmv è semplice: in greatspn il processo generico è dotato di 3 posti, mentre il modello nusmv ha 5 stati. GreatSPN ha meno stati per la scelta di codificare le "linee" di codice sia con posti che con transizioni, invece di usare solamente dei posti.

2.2.4 Riduzione della rete

La riduzione della rete finale (figura 10) ha revisto la rimozione delle seguenti azioni:

- Rimozione self loop sezione non critica per processo P e per processo Q
- Rimozione dei self loop sulle varibili booleane want p e want

Per quanto riguarda i posti sono stati rimossi:

- Rimozione dei posti wantpT e wantqT in quanto non usati

La riduzione ha comunque mantenuto la valutazione delle proprietà della rete. Il reachability graph11 della rete è composto di 9 marcature. La marcatura iniziale m0 è un home state e quindi la rete è reversibile.

2.3 Model checking CTL

Anche per questo algoritmo la modellazione delle proprietà CTL e' stata fatta cercando di mantenerle il più uguali possibili alla versione in NuSMV. In questo caso si può vedere12 che anche in greatSPN l'algoritmo soffre si mancanza di mutua esclusione e di starvation individuale

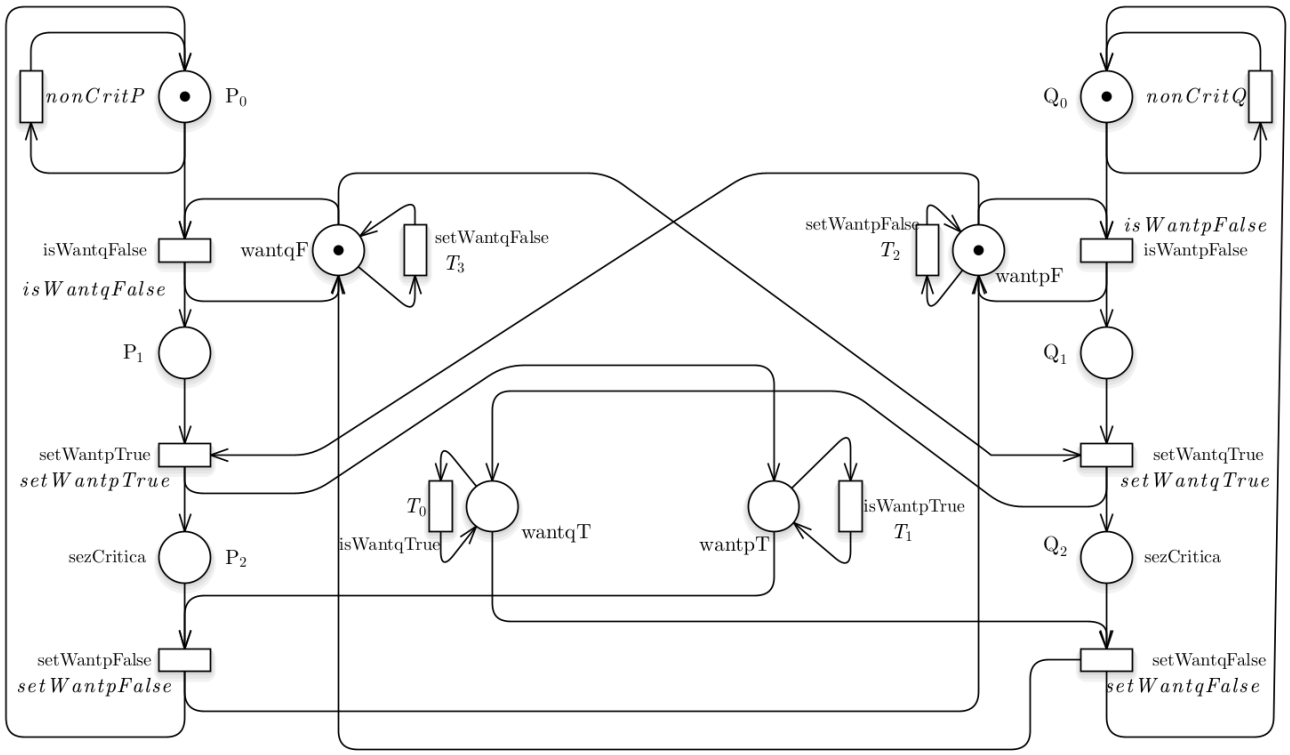


Figura 8: Rete finale algoritmo 3.6

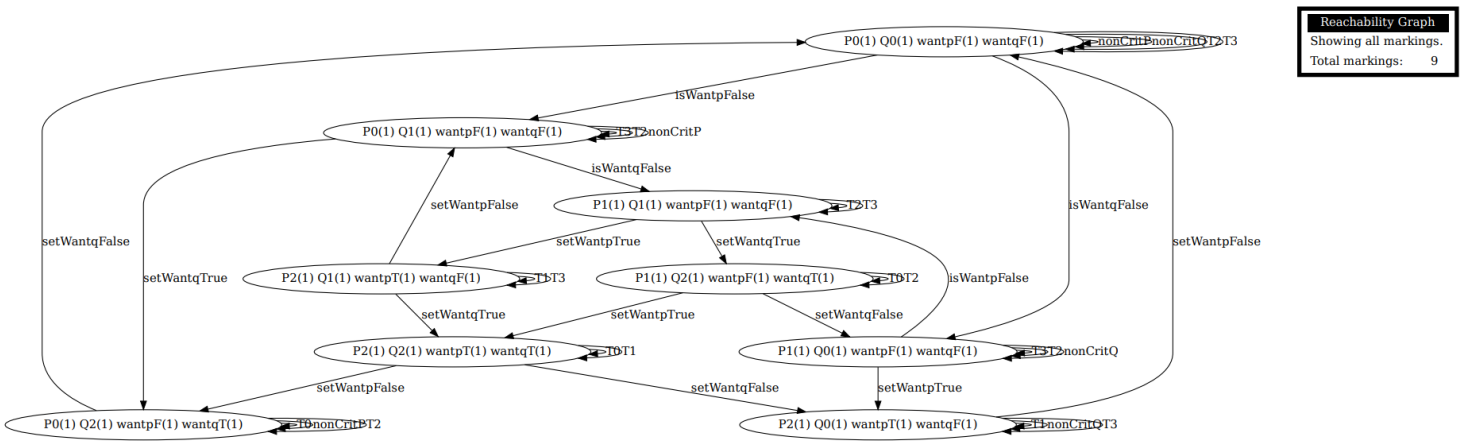


Figura 9: Rete finale algoritmo 3.6

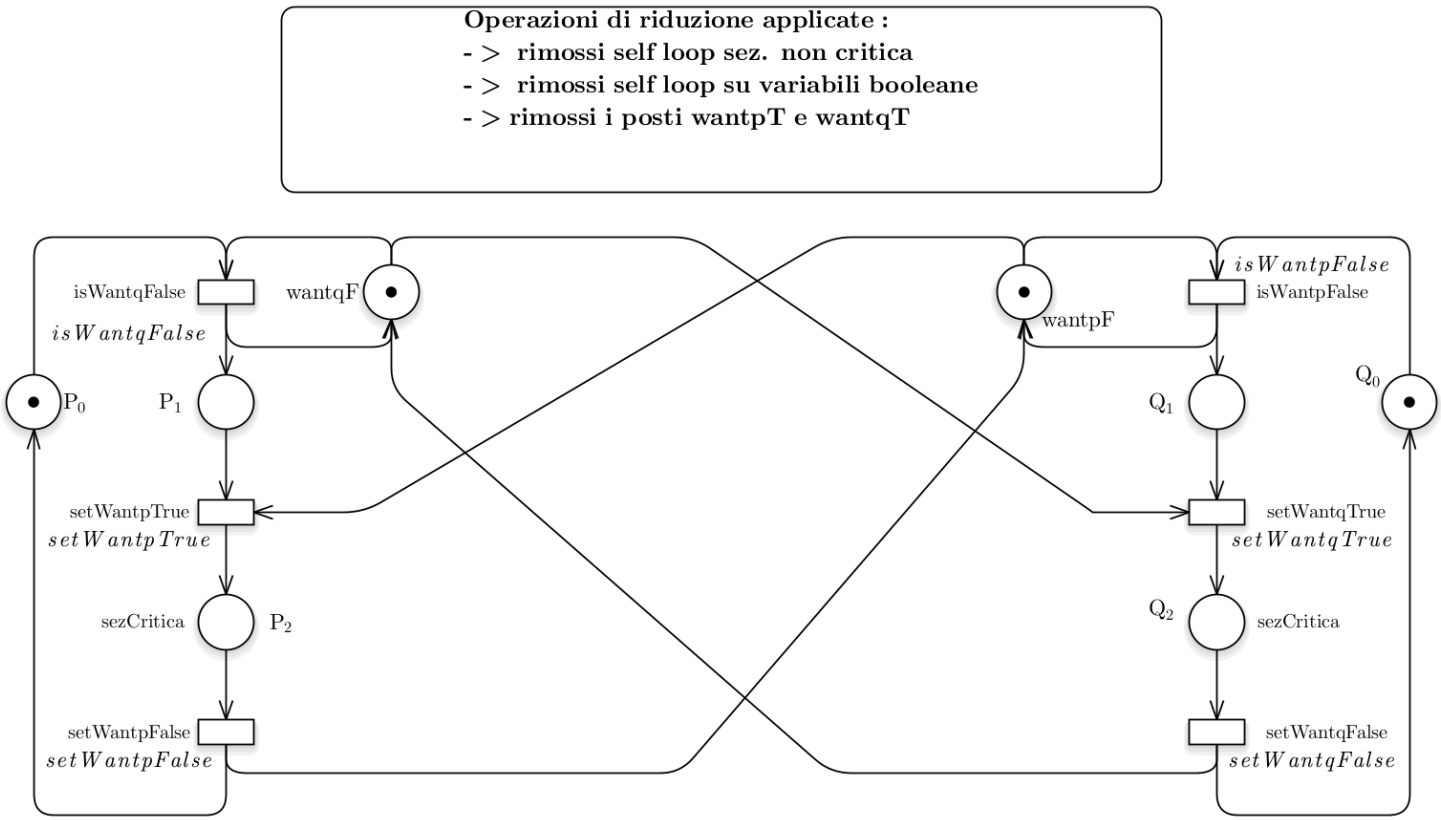


Figura 10: Rete finale ridotta algoritmo 3.6

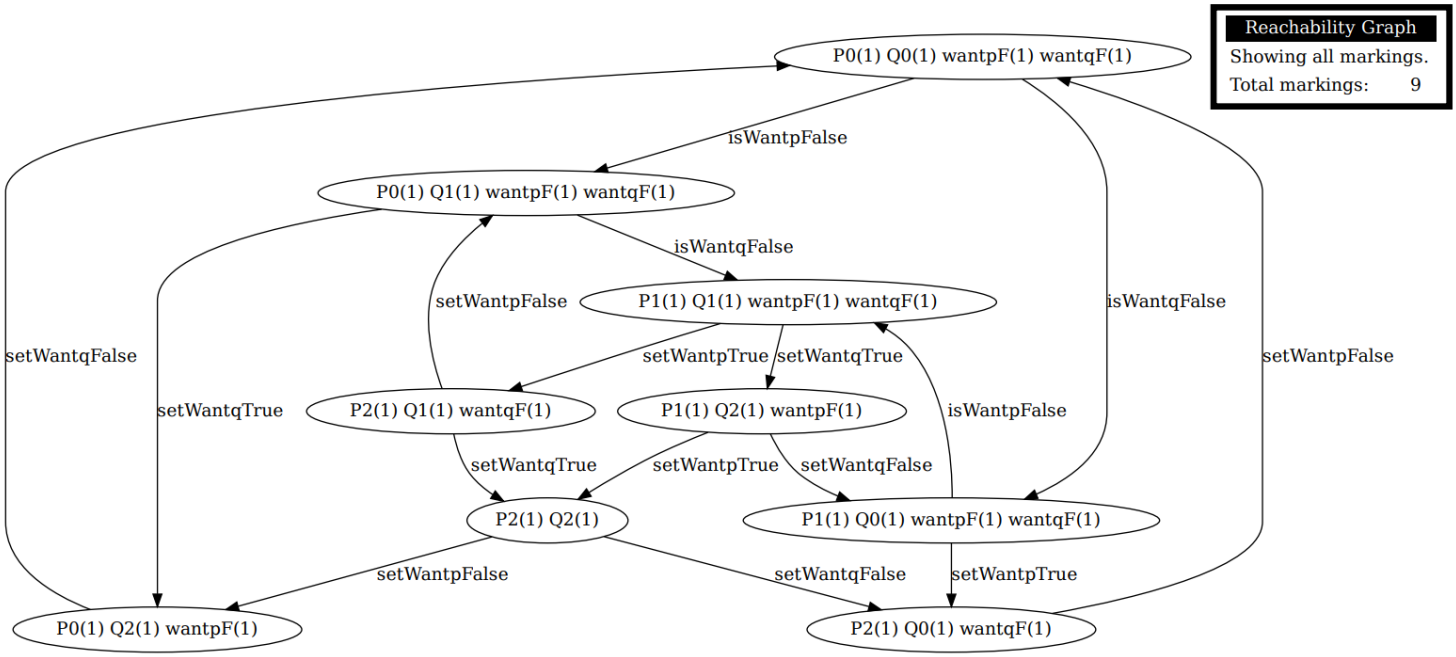


Figura 11: Reachability graph per la rete finale ridotta algoritmo 3.6

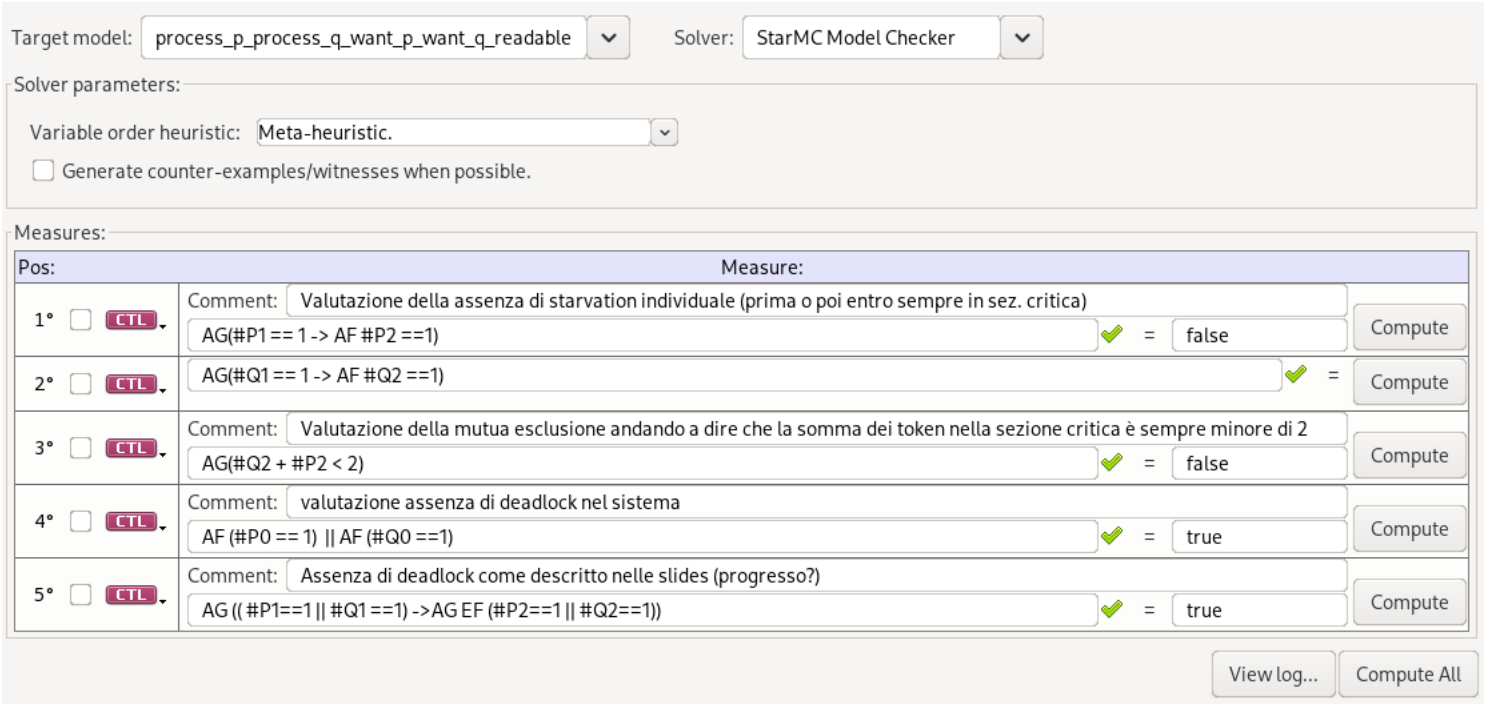


Figura 12: Model checking CTL algoritmo 3.6

2.4 Process Algebra

I processi e le variabili sono state modellati nel seguente modo:
PROCESS P:
P $\triangleq$ `nonCritP.P` + `is_wantq_false.P1`
P1 $\triangleq$ `set_wantp_true.P2`
P2 $\triangleq$ `critical.P3`
P3 $\triangleq$ `set_wantp_false.P`

PROCESS Q:
Q $\triangleq$ `nonCritQ.Q` + `is_wantp_false.Q1`
Q1 $\triangleq$ `set_wantq_true.Q2`
Q2 $\triangleq$ `critical.Q3`
Q3 $\triangleq$ `set_wantq_false.P`

wantP:
WP $\triangleq \overline{is\_wantp\_false.WP} + \overline{set\_wantp\_true.WP1}$
WP1 $\triangleq \overline{is\_wantp\_true.WP1} + \overline{set\_wantp\_false.WP}$

wantQ:
WQ $\triangleq \overline{is\_wantq\_false.WP} + \overline{set\_wantq\_true.WP1}$
WQ1 $\triangleq \overline{is\_wantq\_true.WP1} + \overline{set\_wantq\_false.WP}$

Il derivation graph risultante è il seguente(13)

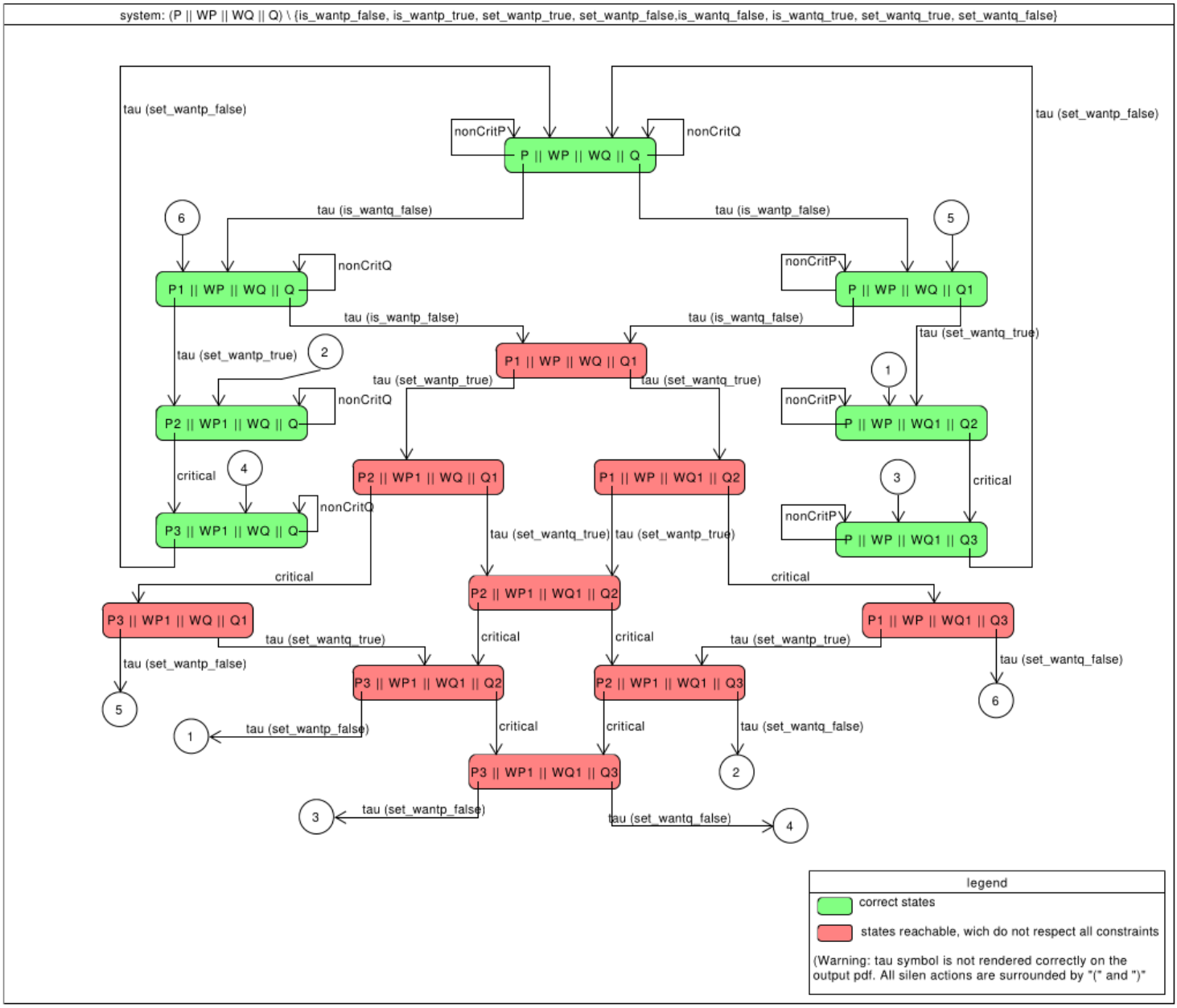


Figura 13: Process algebra per algoritmo 3.6

3 Algoritmo 3.8

3.1 NuSMV

Il listato dell'algoritmo 3.8 e' il seguente:

```
MODULE main
VAR
    want_p_var: boolean;
    want_q_var: boolean;

    proc_p : process proc(want_p_var, want_q_var);
    proc_q : process proc(want_q_var, want_p_var);

ASSIGN
    init(want_p_var) := FALSE;
```

```
    init(want_q_var) := FALSE;

 --mutua esclusione
SPEC AG(proc_p.stato=l4 -> proc_q.stato!=l4) &
AG(proc_q.stato=l4 -> proc_p.stato!=l4);

--assenza di deadlock
SPEC AG EF (proc_p.stato=l1) | AG EF (proc_q.stato=l1);

--assenza di deadlock come definita nelle slides
SPEC AG ((proc_p.stato=l2 | proc_q.stato=l2) ->
AF (proc_p.stato=l4 | proc_q.stato=l4));


MODULE proc(want_io, want_altro)
VAR
    stato : {l1, l2, l3, l4, l5};
ASSIGN
    init(stato) := l1;
    next(stato) :=
        case
            stato=l1 : {l1, l2};
            stato=l2 : l3;
            stato=l3 & want_altro=FALSE: l4;
            stato=l4: l5;
            stato=l5: l1;
            TRUE:stato;
        esac;

    next(want_io) :=
        case
            stato=l2: TRUE;
            stato=l5: FALSE;
            TRUE: want_io;
        esac;



FAIRNESS running;
SPEC AG( stato=l3 -> AF stato=l4 );
```

Anche in questo listato vale la stessa considerazione per il nome assegnato agli stati del sistema e per l'aver modellato un singolo processo.
Per quanto riguarda le clausole di fairness, richieste per la correttezza dell'algoritmo sono state modellate nel seguente modo:

1. **Assenza di starvation individuale**: Questa specifica e' stata modellata usando CTL nel seguente modo:

   ```
   SPEC AG( stato=l3 -> AF stato=l4 );
   ```

   Questa specifica e' valutata come **FALSA** in quanto vi e' deadlock

2. **Mutua esclusione**: Questa specifica e' stata modellata nel seguente modo:

```
        SPEC AG(proc_p.stato=l4 -> proc_q.stato!=l4) &
AG(proc_q.stato=l4 -> proc_p.stato!=l4);
```

Essendo una specifica che richiede di andare a verificare lo stato dei due processi e' stato richiesto di inserire la specifica all'interno del modulo main. Questa specifica viene valutata come **VERA**.

3. **Assenza di deadlock (definizione da slides)**. Questa specifica e' stata modellata come

```
SPEC AG ((proc_p.stato=l2 | proc_q.stato=l2) ->
AF (proc_p.stato=l4 | proc_q.stato=l4));
```

Essendo una specifica che richiede di andare a verificare lo stato dei due processi e' stato richiesto di inserire la specifica all'interno del modulo main. Questa specifica viene modellata come **FALSA** in quanto vi è deadlock.

4. **Assenza di deadlock**:Questa specifica e' stata modellata nel seguente modo:

```
SPEC AG EF (proc_p.stato=l1) | AG EF (proc_q.stato=l1);
```

Essendo una specifica che richiede di andare a verificare lo stato dei due processi e' stato richiesto di inserire la specifica all'interno del modulo main. Questa specifica viene valutata come **FALSA**.

3.2 Rete di Petri

3.2.1 Modello

La rete è stata modellata con le seguenti sotto componenti:

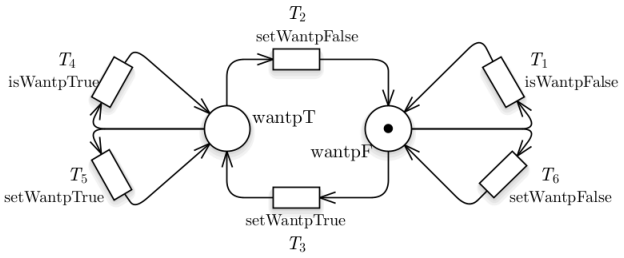
- Variabile booleana want p14a
- Variabile booleana want q14b
- processo P15b
- processo q15a
- rete finale realizzata per composizione16

3.2.2 Rechability graph

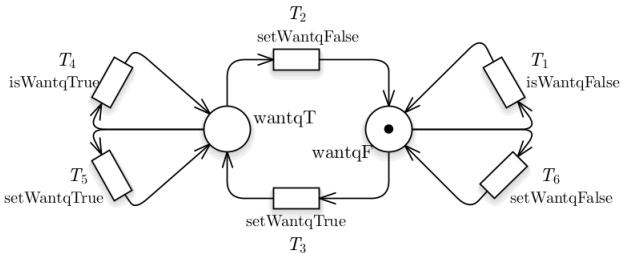
Il reachability graph 17 della rete presenta 8 stati senza pero mostrare apparentemente situazioni di deadlock. Questa situazione e' causata dal modo in cui le variabili booleane sono state modellate: nella marcatura $m = P_1 + wantpT + wantqT + Q_1$ (raggiunte con la firing sequence $\sigma = setWantpTrue, setWantqTrue$), il sistema e' ancora in grado di evolvere in quanto le azioni isWantpTrue, setWantpTrue, isWantqTrue, setWantqTrue sono abilitate. Questa situazione che non rappresenta lo stato del sistema che ci si aspetta, viene risolta applicando le regole di riduzione, che vanno a mostrare effettivamente la presenza di deadlock.
E' comunque possibile individuare due botttom component del RG che si comportano come stati pozzo, ovvero una volta raggiunto quello stato, l'unica azione che posso intraprendere e' una serie di self loop che pero' non mi consentono di spostarmi in un altra marcatura, avendo di fatto una situazione in cui viene violata la assenza di starvation individuale.
Anche in questo caso ci si trova in cui il reachability graph della rete di petri ha meno stati rispetto al modello in nusmv. Anche qua si tratta della stessa motivazione: il modello del processo greatspn ha 3 posti, mentre il modello nusmv ha 5 stati.

3.2.3 Analisi strutturale

Dalla analisi dei p semiflussi (greatSPN ne ha calcolati 8), si evince che la rete sia bounded, e il bound di ogni posto sia 1. Dall'analisi dei t semifluissi (greatSPN ne ha calcolati 10) e dalle statistiche calcolate, parrebbe che la marcatura m0 sia un home state (e quindi che la rete sia reversibile.
Io non sono d'accordo con questa affermazione per il seguente motivo: Citando il capitolo 6 del match book (pagina 4)



(a) Variabile booleana Want p algoritmo 3.8



(b) Variabile booleana Want q algoritmo 3.8

A marking is a home state if it is reachable from any other reachable marking.

Eppure osservando il reachability graph 17 si può notare chiaramente che la marcatura $m^{'} = P_1 + Q_1 + wantpT + wantqT$, non ha alcuna firing sequence che possa portare alla marcatura iniziale m0 (secondo il tool basic statistic di greatSPN, m0 e' un home state**??**). Di conseguenza mi sento di affermare che la marcatura iniziale non sia un home state e che la rete non sia reversibile. Infine, non esistendo un t semiflow che abbbia supporto uguale all'insieme delle transizioni, non si può affermare che la rete sia consistente, cosi come non e' possibile affermare che la rete sia conservativa in quanto non esiste un p semiflow che abbia supporto uguale all'insieme dei posti.

3.2.4 Riduzione della rete

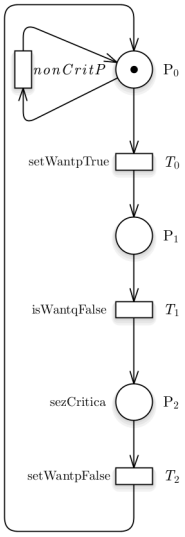
La rete ridotta risultate e' la seguente 19. Per arrivare a questo risultato sono state applicate le seguenti riduzioni:

- Rimosse le transazioni self loop sulla sezione non critica
- Rimosse tutte le transizioni self loop sulle variabili booleane
- Eliminati i posti wantqT e wantpT

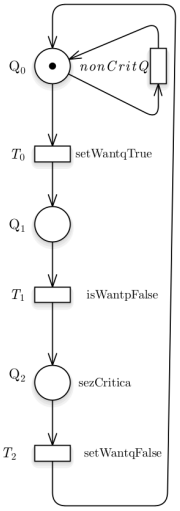
Successivamente aver completato la riduzione, ho provveduto a generare il reachability graph della rete ridotta, ottenendo il grafo20. Da questo rechability graph si può verificare che effettivamente vi e' una situazione di deadlock (come ci si voleva aspettare dalla modellazione con NuSMV). Inoltre il reachability graph ha 9 stati (di cui uno marcato come deadlock). Le proprietà CTL sono state mantenute con la stessa valutazione anche nella rete ridotta.

3.3 Model checking CTL

Anche in questo caso si e' tentato di mantenere la modellazione delle proprietà il più simile al modello NuSMV. In particolare dalla modellazione21 si può osservare che l'algoritmo soffra di mancanza di liveness e di assenza di starvation individuale a causa del deadlock presente in NuSMV, e anche nella rete di petri (a condizione che pero vengano eliminati i self loop che vanno a "nascondere" la presenza di deadlock.



(a) Processo p algoritmo 3.8



(b) Processo q algoritmo 3.8

3.4 Process Algebra

I processi e le variabili sono state modellati nel seguente modo:
PROCESS P:
P $\triangleq$ `nonCritP.P` + `set_wantp_true.P1`
P1 $\triangleq$ `is_wantq_false.P2`
P2 $\triangleq$ `critical.P3`
P3 $\triangleq$ `set_wantp_false.P`

PROCESS Q:
Q $\triangleq$ `nonCritQ.Q` + `set_wantq_true.Q1`
Q1 $\triangleq$ `is_wantp_false.Q2`
Q2 $\triangleq$ `critical.Q3`
Q3 $\triangleq$ `set_wantq_false.P`

wantP:
WP $\triangleq \overline{is\_wantp\_false.WP} + \overline{set\_wantp\_true.WP1}$
WP1 $\triangleq \overline{is\_wantp\_true.WP1} + \overline{set\_wantp\_false.WP}$

wantQ:
WQ $\triangleq \overline{is\_wantq\_false.WP} + \overline{set\_wantq\_true.WP1}$
WQ1 $\triangleq \overline{is\_wantq\_true.WP1} + \overline{set\_wantq\_false.WP}$

Il derivation graph risultante è il seguente(22)

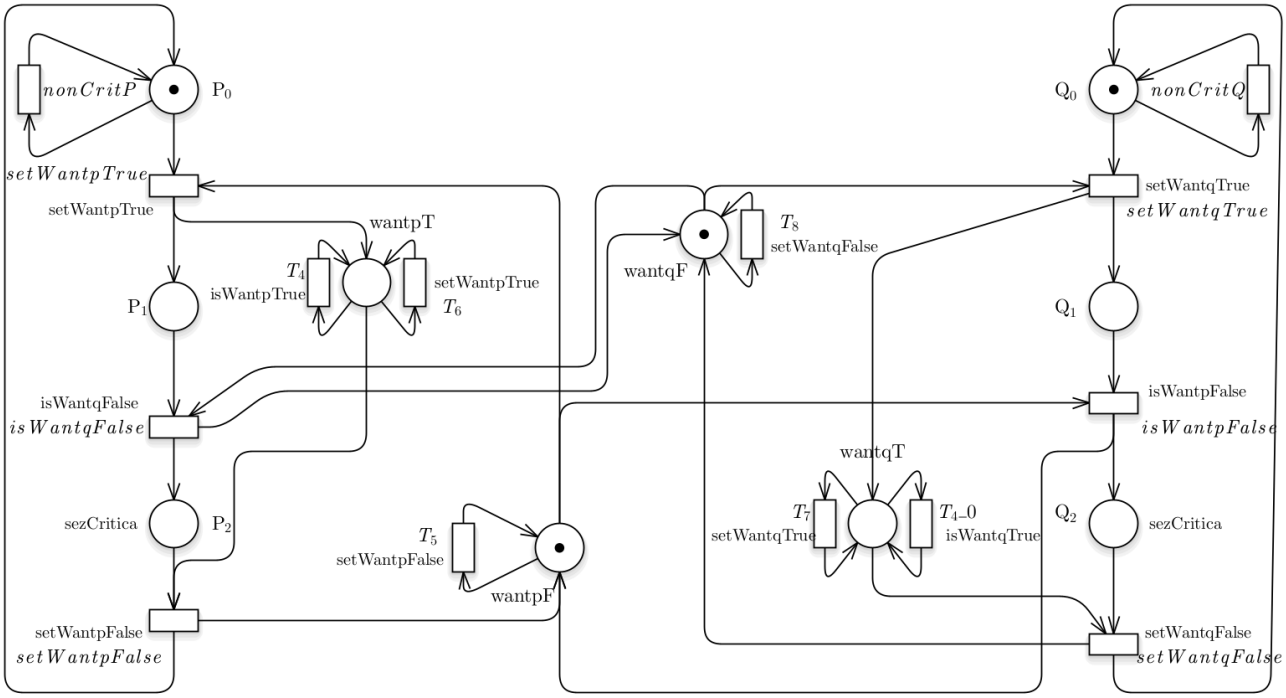


Figura 16: Rete finale algoritmo 3.8

4 Calcolo bisimulazione tra algoritmi 3.6 e algoritmi 3.8

Per praticità ho proceduto a rinonimare gli stati ottenuti dai derivation graph dei due algoritmi, come illustrato nelle immagini 23 e 24.

Poiché ho usato CCS per modellare, ho proceduto a calcolare una weak bisimulation, per andare a tenere in considerazione le azioni silenti tau. Il risultato del calcolo e' il seguente

1. **partizione iniziale** = {X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11,X12, Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Y9, Y10, Y11, Y12, Y13, Y14, Y15, Y16}
2. **Split su nonCritP** P1 = {Y1, Y3, Y5, Y9, X1, X3, X5},{Y2,Y4, Y6, Y7, Y8, Y10, Y11, Y12, Y13, Y14, Y15, Y16, X2, X4, X7, X8, X9, X10, X11, X12 }
3. **Split su nonCritQ** P2 = {Y1, X1} , {Y3, Y5, Y9, X3, X5} , {Y2, Y4, Y10, X2, X4, X7} , {Y6, Y7, Y8, Y11, Y12,Y13, Y14, Y15, Y16, X8, X9, X10, X11, X12}
4. **Split su crititical** P3 = {Y1, X1} , {Y5,X5} , {Y3, Y9, X3} , {Y4, X4} , {Y2,Y10,X2,X7}, {Y7,Y8,Y12,Y14,Y15,X9, X10} , {Y6,Y11,Y13,Y16,X8,X11,X12}
5. **Split su** τ P3 = {Y1,X1} , {Y5} , {X5} , {Y3, Y9,X3} , {X4} , {Y4} , {Y2,Y10,X2,X7} , {Y7, Y8, Y14, Y15}, {Y12, X9, X10} , {Y6, Y11, Y13, Y16, X11, X12} , {X8}

Poiché dopo la valutazione dello split su tau (ultimo step) mi trovo ad avere delle partizioni che contengono un solo elemento, posso affermare che i due sistemi non sono una weak bisimulation dell'altro, in quanto, per esempio, nella partizione che contiene solo X8, non vi e' uno stato di Y (algoritmo 3.6) che possa simulare X8.
In realtà non sarebbe stato necessario fare il calcolo della bisimulazione ma sarebbe stato possibile affermare a priori che non si può costruire una bisimulazione, ne una bisimulazione debole tra i due sistemi, in quanto l'algoritmo 3.8 va in deadlock e il 3.6 no (l'algoritmo 3.6 non e' quindi in grado di simulare l'algoritmo 3.8).

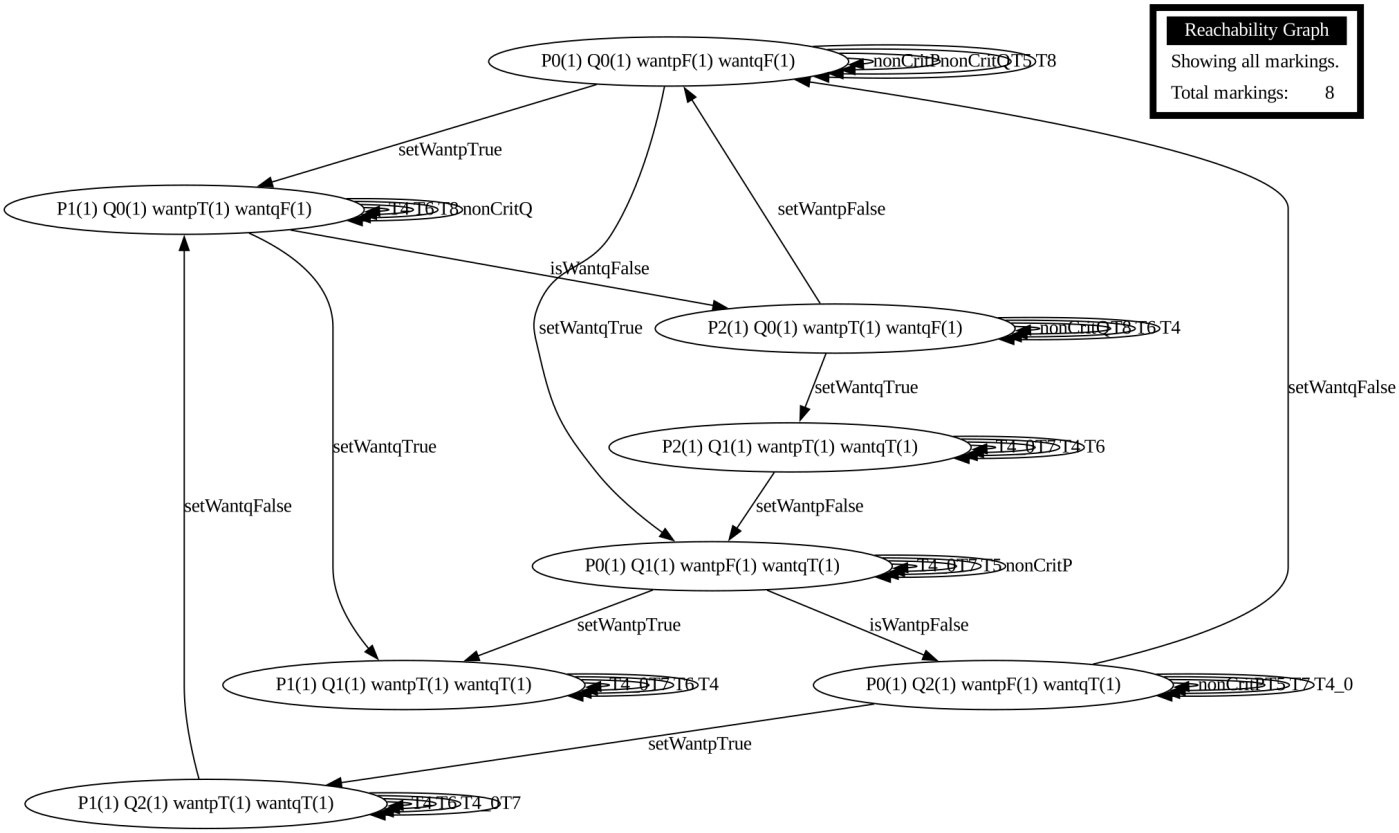


Figura 17: Rechability graph algoritmo 3.8

```
 ***** Reachability Graph *****

TANGIBLE MARKINGS   : 8
VANISHING MARKINGS  : 0
DEAD MARKINGS       : 0

TOTAL MARKINGS      : 8

 ************************************

The initial marking is a home state
Time required ----------> 0

************************************
```

Figura 18: Calcolo delle statistiche di base algoritmo 3.8 greatSPN

5 Algoritmo 3.10

5.1 NuSMV

Il listato dell'algoritmo 3.10 e' il seguente:

```
MODULE main
VAR
    turno: {p,q};
    want_p_var: boolean;
```

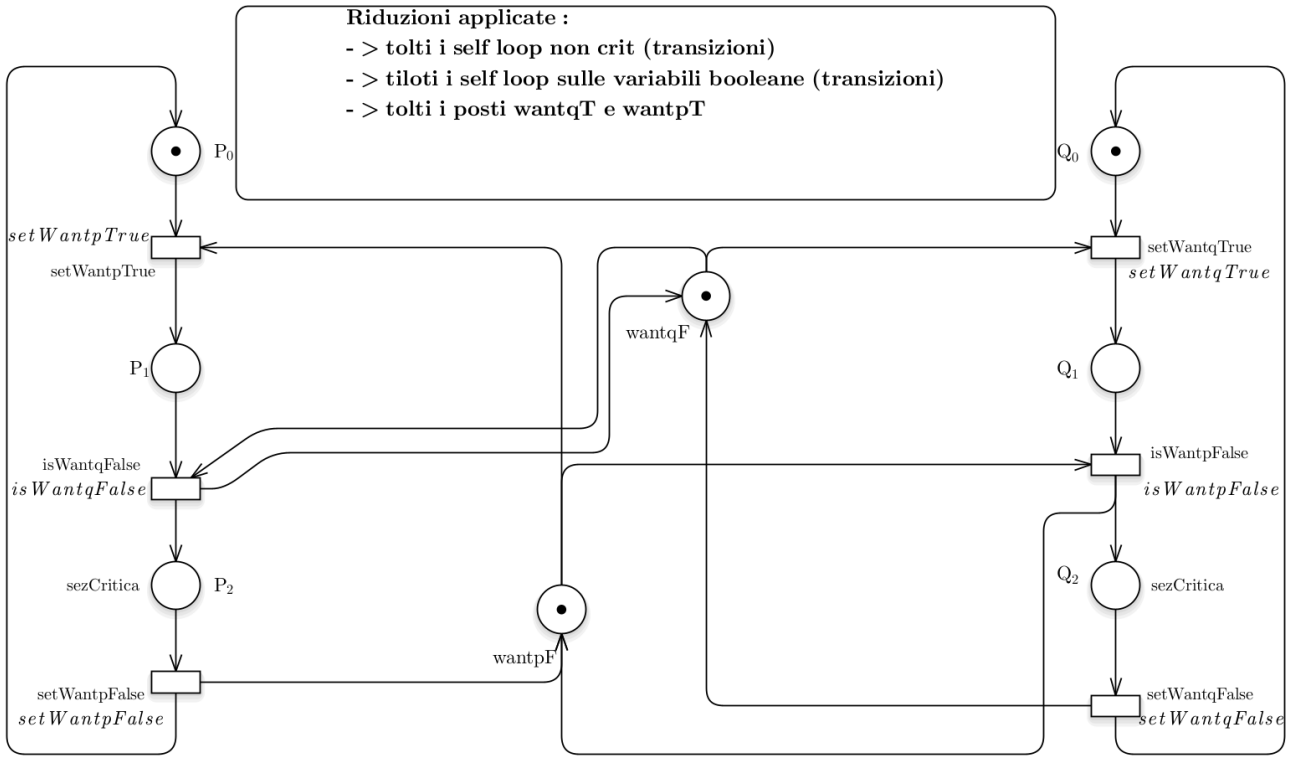


Figura 19: Rete ridotta algoritmo 3.8

```
    want_q_var: boolean;

    proc_p: process proc(p, want_p_var, want_q_var, turno);
    proc_q: process proc(q, want_p_var, want_q_var, turno);
ASSIGN
    init(want_p_var) := FALSE;
    init(want_q_var) := FALSE;
    init(turno) := p;

--assenza di deadlock (
SPEC AG EF (proc_p.stato=l1) | AG EF (proc_q.stato=l1)

--mutua esclusione
SPEC AG ((proc_p.stato=l8 -> proc_q.stato!=l8) &
(proc_q.stato=l8 -> proc_p.stato!=l8));

--assenza di deadlock come definita nelle slides
SPEC AG ( (proc_p.stato=l2 -> AF proc_p.stato=l8 ) |
(proc_q.stato=l2 -> AF proc_q.stato=l8 ))


MODULE proc(identita, want_p, want_q, turno)
VAR
    stato : {l1, l2, l3, l4, l5, l6, l7, l8, l9, l10};
```

```
ASSIGN
    init(stato) := l1;
    next(stato) :=
        case
            stato=l1 : {l1, l2};
            stato=l2: l3; -- set want_identita a true
            stato=l3:
                case
                    identita=p & want_q=TRUE: l4;
                    identita=q & want_p=TRUE: l4;
                    TRUE: l8;
                esac;
            stato=l4:
                case
                    identita=p & turno=q: l5;
                    identita=q & turno=p: l5;
                    TRUE: l3;
                esac;
            stato=l5: l6; --set di want_idendita a false
            stato=l6:
                case
                    turno=identita: l7;
                    TRUE: stato; --attendo il turno uguale alla mia identita'
                esac;

            stato=l7: l3; --torno alle testa del ciclo
            stato=l8:l9; --sezione critica
            stato=l9:l10; --aggiornamento di turno alla altra variabile
            stato=l10: l1; --ricomincio da capo
            TRUE: stato;
        esac;

    next(want_p) :=
        case
            identita=p:
                case
                    stato=l2 | stato=l7: TRUE;
                    stato=l5 | stato=l10: FALSE;
                    TRUE: want_p;
                esac;
            TRUE:want_p;
        esac;

    next(want_q) :=
        case
            identita=q:
                case
                    stato=l2 | stato=l7: TRUE;
                    stato=l5 | stato=l10: FALSE;
                    TRUE: want_q;
```

```
                esac;
            TRUE: want_q;
        esac;

    next(turno) :=
        case
            stato=l9:
            case
                identita=p: q;
                identita=q: p;
                TRUE: turno;
            esac;
            TRUE: turno;
        esac;
FAIRNESS running;

SPEC AG(stato=l2 -> AF stato=l8);
```

Anche in questo listato vale la stessa considerazione per il nome assegnato agli stati del sistema e per l'aver modellato un singolo processo.
Per quanto riguarda le clausole di fairness, richieste per la correttezza dell'algoritmo sono state modellate nel seguente modo:

1. **Assenza di starvation individuale**: Questa specifica e' stata modellata usando CTL nel seguente modo:

   ```
   AG( stato=l3 -> AF stato=l4 );
   ```

   Questa specifica e' valutata come **VERA**

2. **Mutua esclusione**: Questa specifica e' stata modellata nel seguente modo:

   ```
   SPEC AG ((proc_p.stato=l8 -> proc_q.stato!=l8) &
   (proc_q.stato=l8 -> proc_p.stato!=l8));
   ```

   Essendo una specifica che richiede di andare a verificare lo stato dei due processi e' stato richiesto di inserire la specifica all'interno del modulo main. Questa specifica viene valutata come **VERA**.

3. **Assenza di deadlock (definizione da slides)**. Questa specifica e' stata modellata come

   ```
   SPEC AG ( (proc_p.stato=l2 -> AF proc_p.stato=l8 ) |
   (proc_q.stato=l2 -> AF proc_q.stato=l8 ))
   ```

   Essendo una specifica che richiede di andare a verificare lo stato dei due processi e' stato richiesto di inserire la specifica all'interno del modulo main. Questa specifica viene modellata come **VERA**.

4. **Assenza di deadlock**:Questa specifica e' stata modellata nel seguente modo:

   ```
   SPEC AG EF (proc_p.stato=l1) & AG EF (proc_q.stato=l1)
   ```

   Essendo una specifica che richiede di andare a verificare lo stato dei due processi e' stato richiesto di inserire la specifica all'interno del modulo main. Questa specifica viene valutata come **VERA**.

Essendo modellato effettivamente l'algoritmo di dekker, la valutazione delle proprieta' e' coerente con quello che ci si aspetterebbe.

5.2 Rete di Petri

5.2.1 Modello

La rete è stata modellata con le seguenti sotto componenti:

- Variabile booleana want p25a
- Variabile booleana want q25b
- Variabile booleana turn25c
- processo P26a
- processo q26b
- rete finale realizzata per composizione27

Una nota sulla variabile turn: ho dovuto modellare tre diverse variabili per il test del turno, per evitare di avere sincronizzazioni non volute (nel caso che si utilizzi la medesima transizione in tutta la rete).
La rete finale ha usato archi broken essendo troppi per poter tentare una visualizzazione che producesse senso. pertanto l'immagine non e' molto rappresentativa della situazione e per capire cosa stia succedendo e' necessario rifarsi al simulatore del modello o al reachability graph della rete.

5.2.2 Rechability graph

Il reachability graph?? della rete e' composto di 70 stati, non vi sono deadlock e la marcatura iniziale e' un home state. Sfortunatamente essendo molto grande non rende bene in immagine, pertanto viene comunque allegata la rete originale in greatSPN.
Anche in questo caso ci si trova in cui il reachability graph della rete di petri ha meno stati rispetto al modello in nusmv. Anche qua si tratta della stessa motivazione: il modello del processo greatspn ha 8 posti, mentre il modello nusmv ha 10 stati.

5.2.3 Analisi strutturale

GreatSPN ha calcolato 0 p semiflussi minimali tutti con bound a 1, pertanto possiamo affermare che la rete sia bounded. Ha anche calcolato 15 T semiflussi minimali, di cui effettivamente di interesse sono 3. Con le informazioni calcolate, si può affermare che la marcatura iniziale sia un home state e che la rete sia reversibile. Non esistendo un t semiflow che abbia supporto uguale all'insieme delle transizioni, non si può affermare che la rete sia consistente, cosi come non e' possibile affermare che la rete sia conservativa in quanto non esiste un p semiflow che abbia supporto uguale all'insieme dei posti.

5.3 Model checking CTL

Anche in questo caso si e' tentato di mantenere la modellazione delle proprietà il più simile al modello NuSMV. In particolare dalla modellazione29 si può osservare che l'algoritmo verifichi come vere tutte le proprietà definite, come ci si voleva aspettare dall'algoritmo corretto.

6 Confronti

6.1 Tabella di confronti

Per la lettura della tabella: se in una cella vi è un si, vuole dire che la proprietà indicata dalla colonna è verificata come true per l'algoritmo indicato nella linea.

Algo	$\|RS\|$ PN	$\|RS\|$ NuSMV	Mutua esclusione	deadlock (slides)	assenza starvation individuale	assenza di deadlock
3.2	16	16	SI	SI	NO	SI
3.6	9	25	NO	SI	NO	SI
3.8	8	21	SI	NO	NO	NO
3.10	70	134	SI	SI	SI	SI

6.2 Commenti sui confronti

Innanzitutto si nota una discrepanza nella dimensione dei rechability set: in particolare NuSMV ha più stati rispetto a greatSPN. Questo fatto è causato da una differenza nella modellazione in NuSMV e in greatSPN. In particolare, in nusmv è stata modellata ogni linea di codice come uno stato, mentre in greatspn non tutte le linee di codice corrispondono a una transazione (in particolare la sezione critica è stata modellata come un posto e non come una transizione).

Le valutazione dei model checker delle proprietà non e' inattesa, anzi era anche intuibile guardando lo pseudo codice dell'algoritmo. L'avere avuto una conferma anche dai model checker delle mie supposizioni ha dato un riscontro positivo sulla presunta corretta modellazione degli algoritmi.

6.2.1 Fairness

Per tutti i modelli NuSMV è stato inserita come condizione fairness "running" in tutti i moduli, per garantire che infinitamente spesso essi vengano eseguiti.
Su greatSPN invece non è stato possibile andare a inserire condizioni di fairness in quanto avrei avuto la necessità di andare a esprimere che delle transizioni "scattano nel futuro" ma questo, a quanto so, non è possibile in greatSPN.

6.3 Modifica requisito progresso in sezione critica

Ho verificato questa modifica andando a modificare un modello in NuSMV, andando a inserire una scelta non deterministica nello stato di richiesta di accesso, ovvero:

```
...
   stato=l2: {l2,l3}; -- set want_identita a true
...
```

Le modifiche sono state effettuate sulla base dell'algoritmo 3.10 (ovvero dekker) e sono visibili nel file `listing-no-progresso-richiesta-accesso.smv`. Questa modifica significa che un processo può ora esprimere il volere di entrare in sezione critica, senza pero' effettivamente continuare a eseguire il protocollo di accesso. Il risultato di questa modifica e' che ora la proprietà di assenza di starvation individuale non e' più verificata. Risulta anche che viene invalidata la proprietà di assenza di deadlock (con la definizione delle slides). Restano valide invece mutua esclusione e la assenza di deadlock nel sistema. Sulla mutua esclusione non vi e' da stupirsi se viene mantenuta in quanto tutti gli steps del protocollo di accesso devono essere effettuati per entrare in regione critica. Il deadlock ovviamente non c'è in quanto il sistema ha comunque che o un processo va in loop sulla sezione critica oppure un processo va in loop infinito sul punto in cui si esprime l'intenzione di entrare in sezione critica (senza mai effettivamente entrarci).
Per scelta e' stato modellato come scelta non deterministica di rimanere in loop sulla espressione della volontà di entrare in sezione critica, ma sarebbe stato uguale (anche se forse intuitivamente meglio), effettuare un salto alla sezione non critica cosi' da modellare un concetto di "voglio entrare in futuro in sezione critica".

Figura 20: Rechability graph per rete ridotta algoritmo 3.8

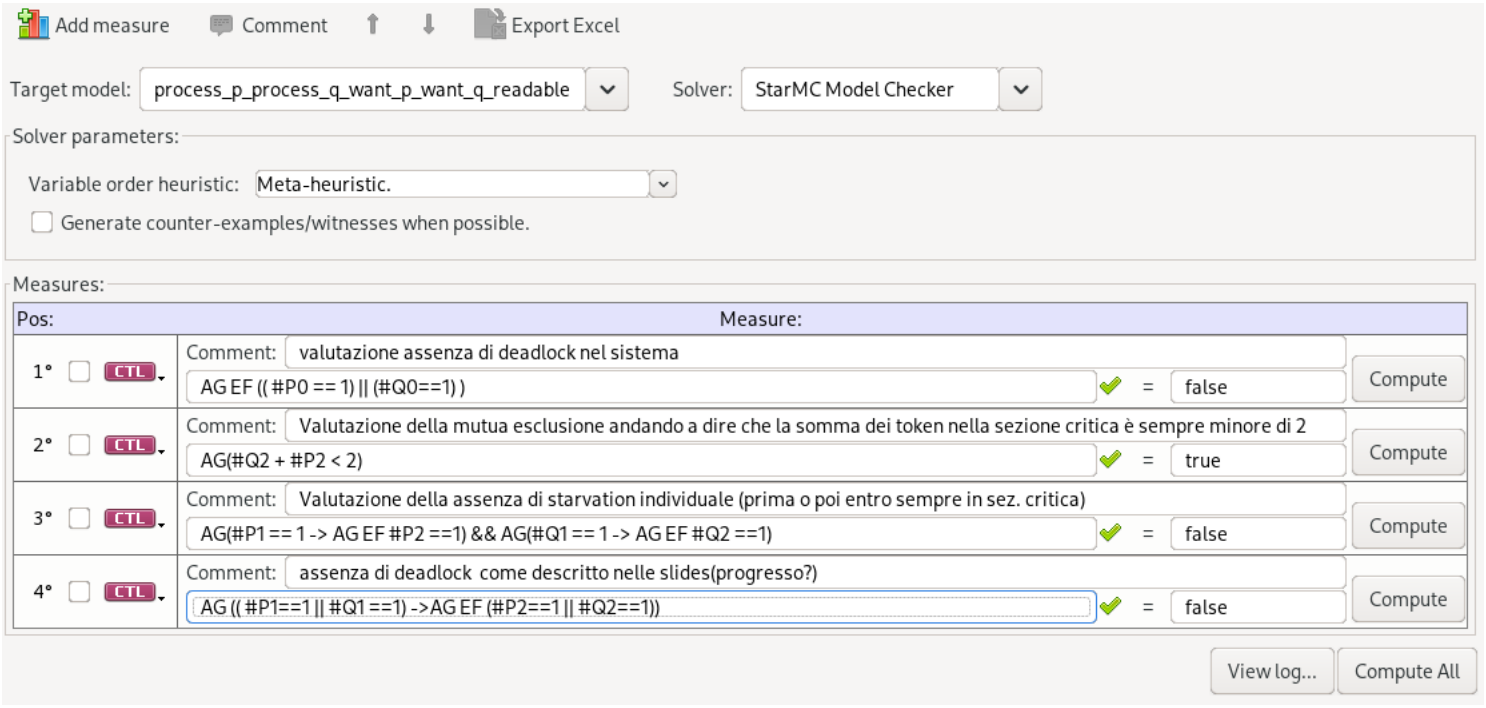


Figura 21: CTL model checking per algoritmo 3.8

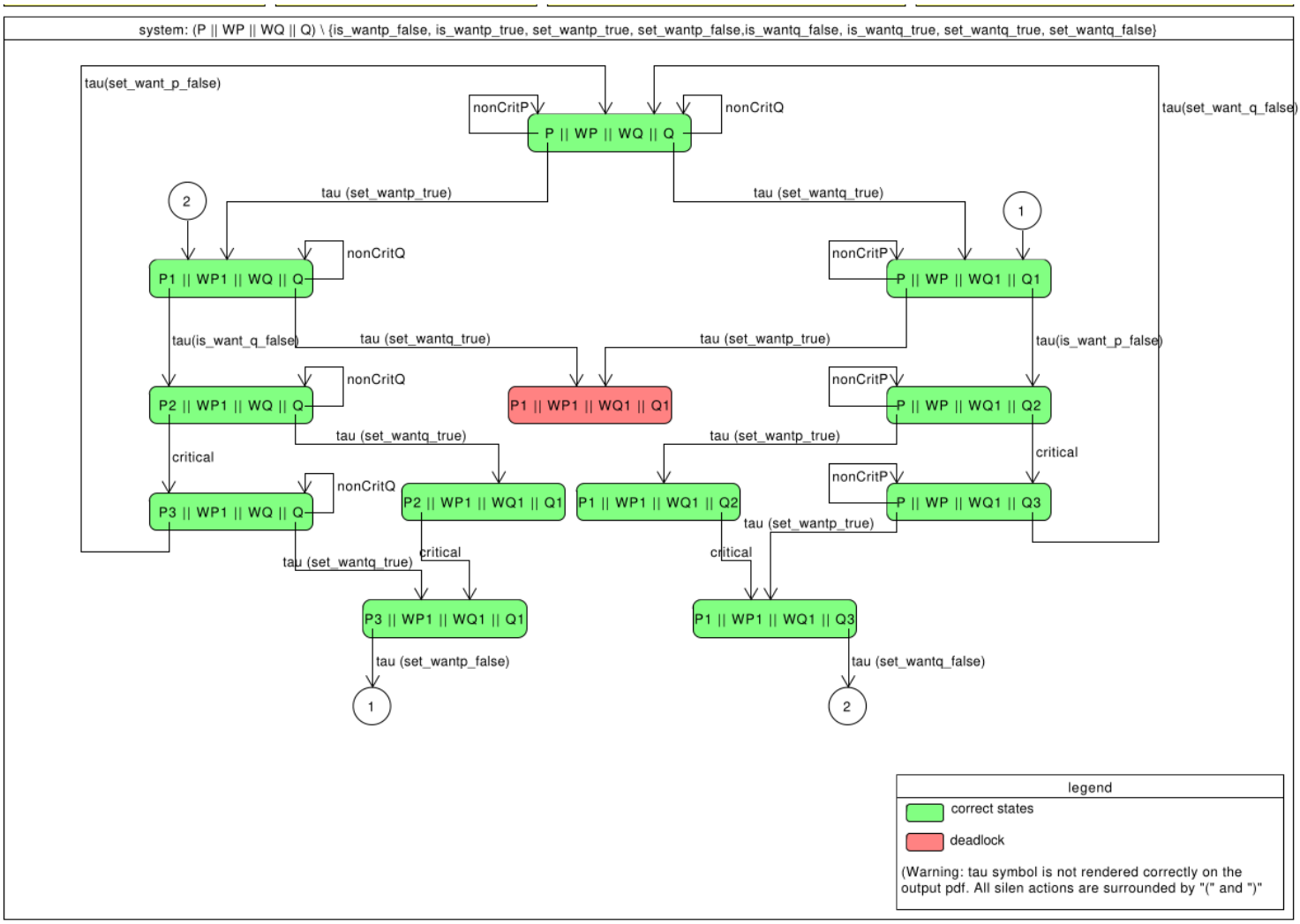


Figura 22: Process algebra per algoritmo 3.8

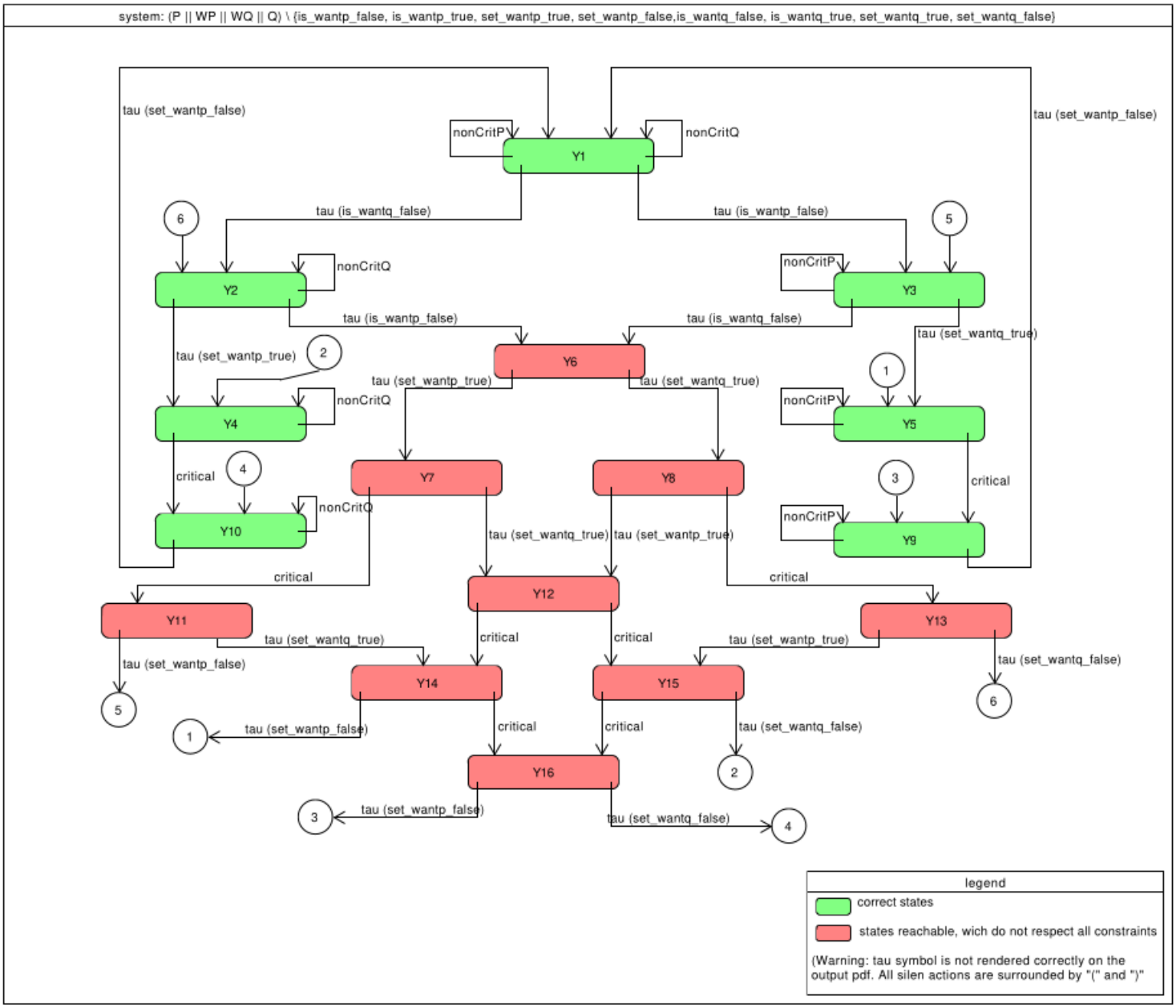


Figura 23: Process algebra per bisimulazione di algoritmo 3.6

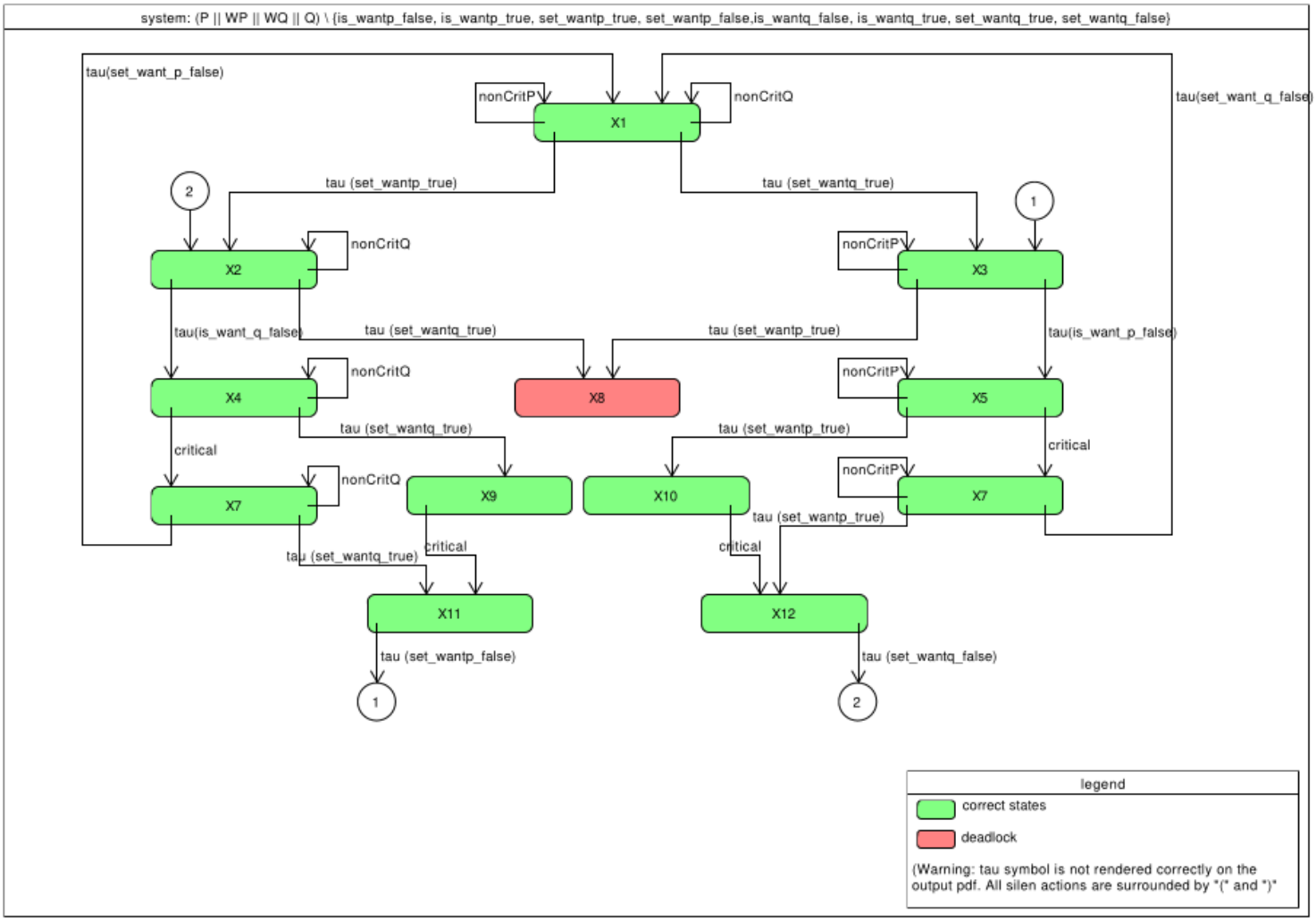
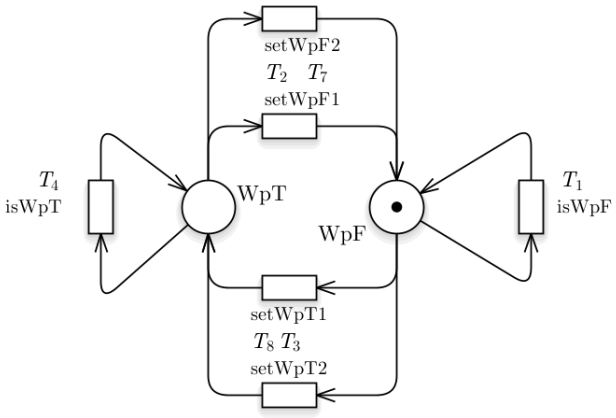
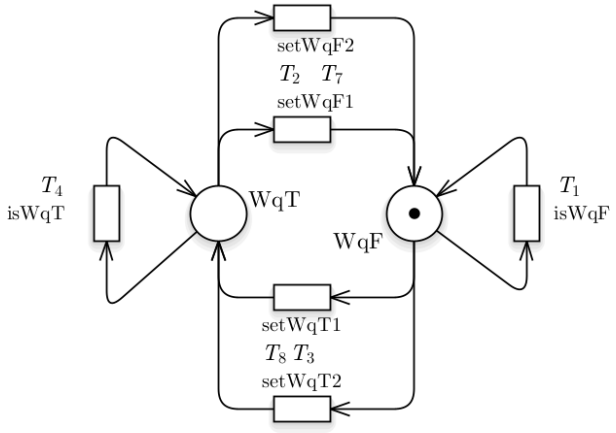


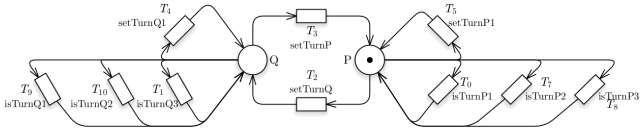
Figura 24: Process algebra per bisimulazione di algoritmo 3.8



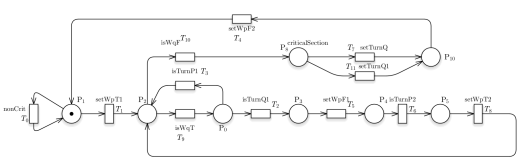
(a) Variabile booleana Want p algoritmo 3.10



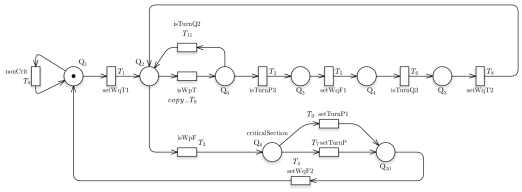
(b) Variabile booleana Want q algoritmo 3.10



(c) Variabile booleana turn algoritmo 3.10



(a) Processo p algoritmo 3.10



(b) Processo q algoritmo 3.10

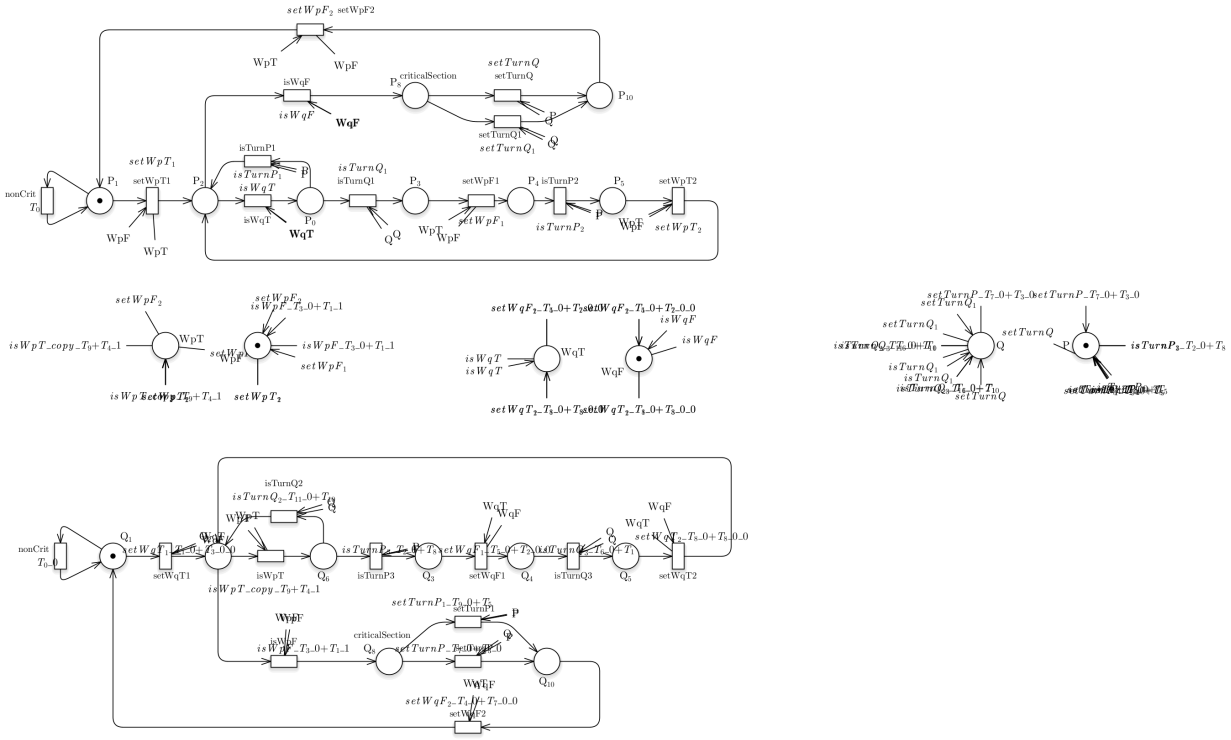


Figura 27: Rete finale algoritmo 3.10

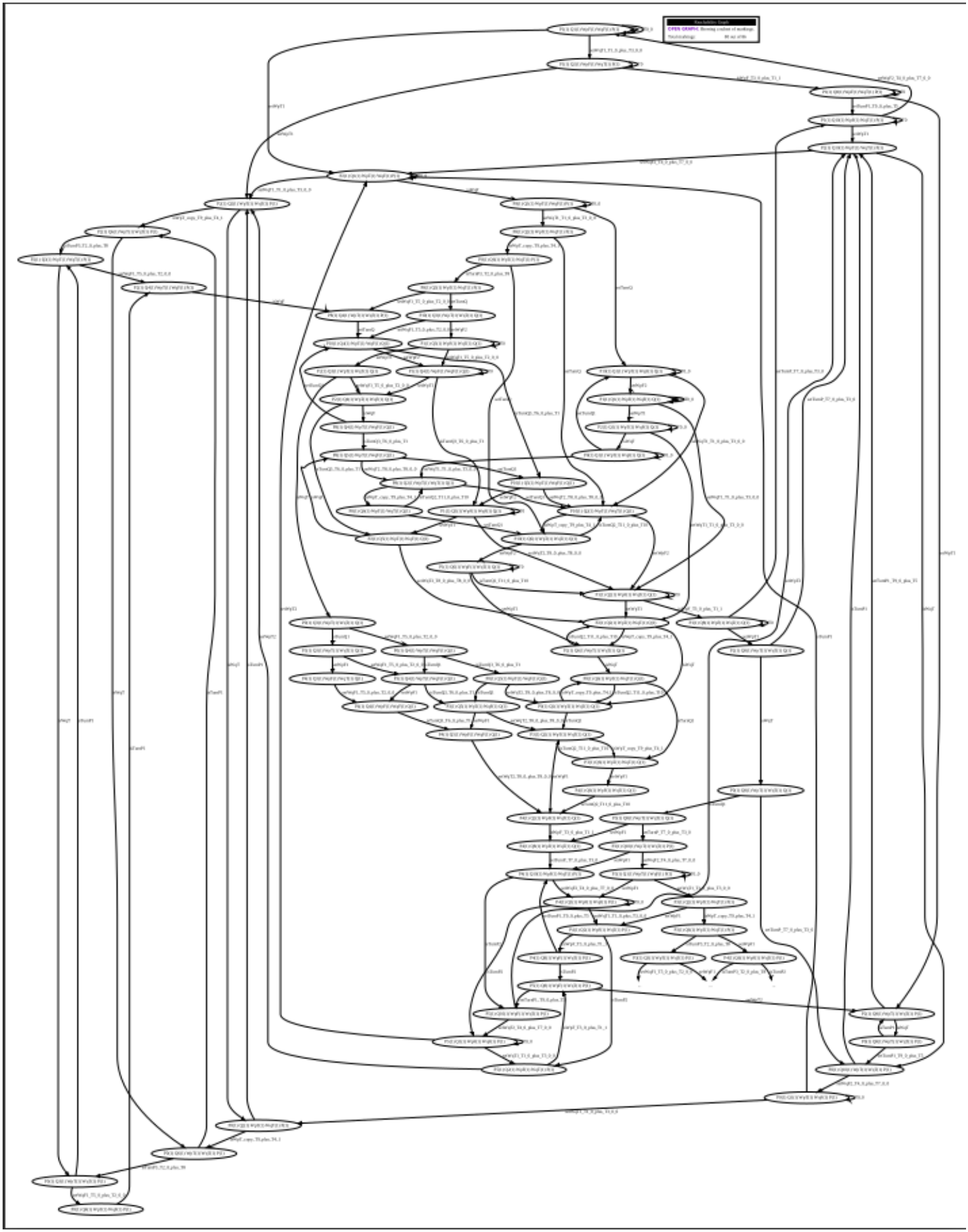


Figura 28: Rechability graph algoritmo 3.10

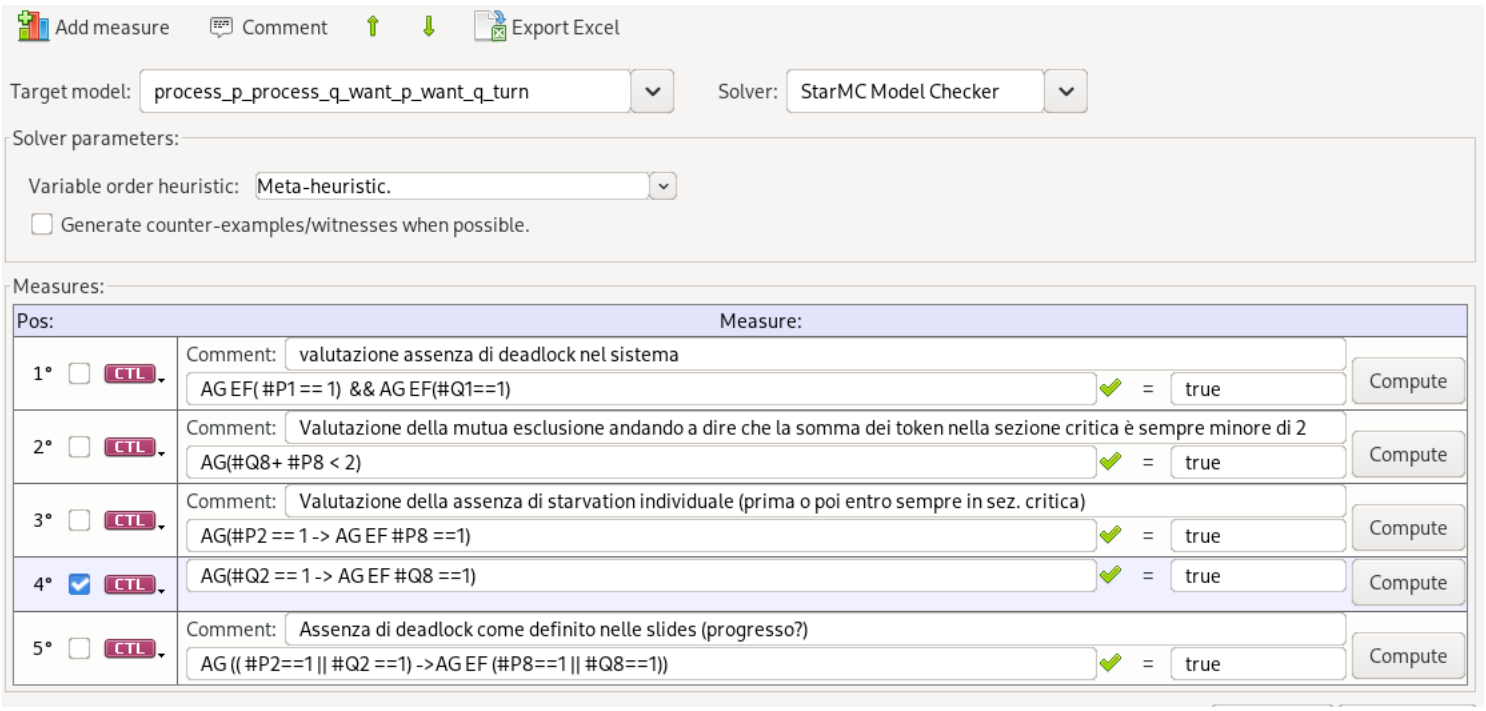


Figura 29: CTL model checking per algoritmo 3.10