Produttori e consumatori in PA e Nusmv

Lorenzo Dentis, lorenzo.dentis@edu.unito.it

14 dicembre 2022

1 Process Algebra

1.1 1 produttore, 1 consumatore, buffer a N posizioni

Il sistema è organizzato come segue:

$SYS = (Prod||Cons||(Buffer))/_{\{put,get,passa\}}$
$Prod = think.produce.\overline{PUT}.Prod$
$Cons = think.produce.\overline{GET}.Cons$
funzioni di Relabeling:

$$f_E\begin{bmatrix} put \to put \\ get \to passa_1 \end{bmatrix}$$

$$f_F\begin{bmatrix} put \to \overline{passa}_{n-1} \\ get \to passa_n \end{bmatrix} \forall n \in [1, N]$$

$$f_E\begin{bmatrix} put \to put \\ get \to \overline{passa}_n \end{bmatrix}$$

$$Buffer = \underbrace{E_{[f_E]}||...||E_{[f_M]}||...||E_{[f_F]}}_{N \text{ volte}}$$

$E = PUT.F$
$F = GET.E$

Le operazioni put, get sono inserite in una restrizione, in modo che sia possibile inserire e rimuovere oggetti dal buffer solamente tramite τ-operazioni che coinvolgono una "cella" del buffer ed uno tra il consumatore ed il produttore.
Non è possibile indicare un buffer a N posizioni in maniera generica quindi ho costruito il buffer come una composizione di N buffer ad 1 posizione, dato che l'esercizio richiede che non il buffer sia un unico oggetto e non sia possibile inserire elementi in differenti "celle" di questo ho definito tre funzioni di relabeling ed una nuova operazione $passa_n$ anche essa inserita nella restrizione.

- f_E: operazione effettuabile solo da una cella del buffer, è l'unica cella che può effettuare operazioni di *put*, obbligando il produttore ad inserire dati solo in quella cella.
- f_F: operazione effettuabile solo da una cella del buffer, è l'unica cella che può effettuare operazioni di *get*, obbligando il consumatore a prelevare dati solo da quella cella.
- f_M: questa funzione di relabeling permette alle celle del buffer differenti da quella iniziale e quella finale di passare il dato verso il fondo del buffer.

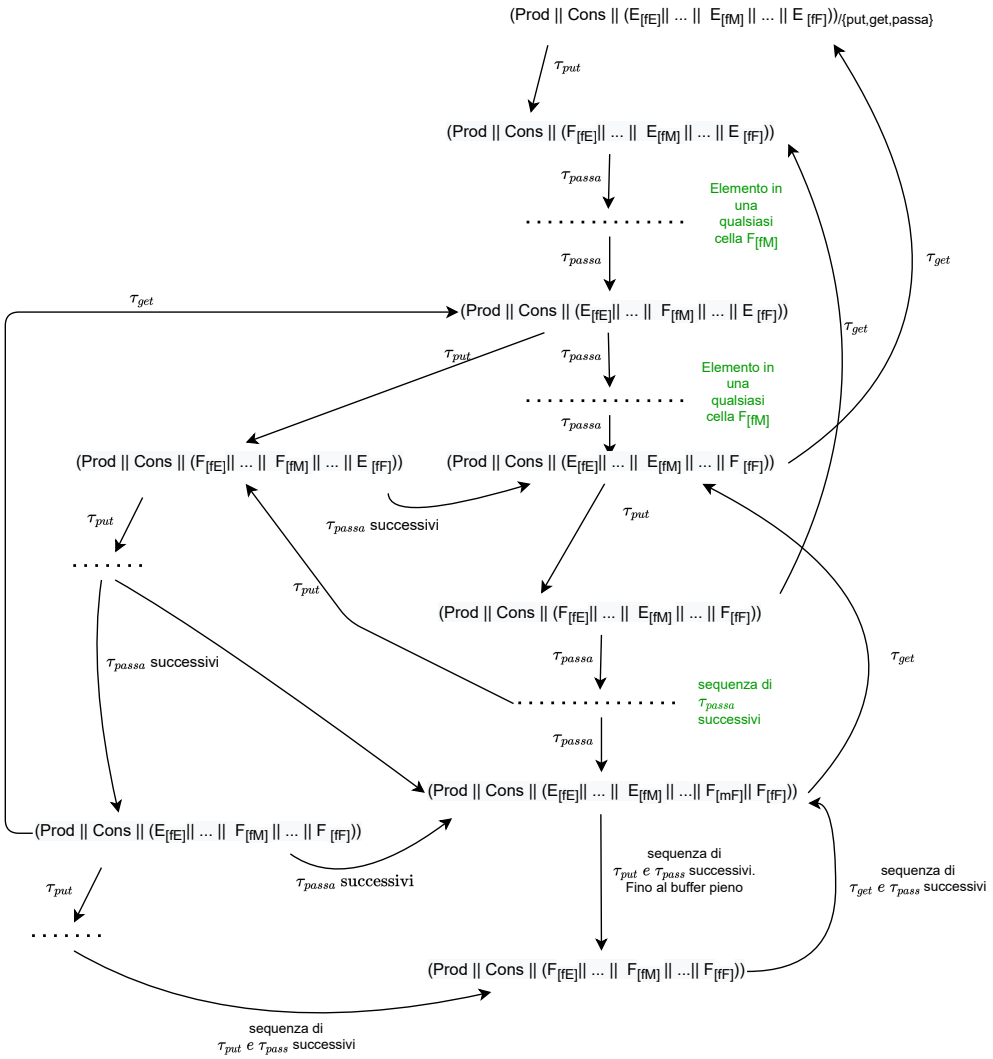

Figura 1: Setting1

Lo scopo principale di questo *derivation graph* è mostrare il funzionamento del buffer a N posizioni, l'operazione di *put* può essere effettuata solo sulla prima cella del buffer e l'operazione di *get* solo sull'ultima. Il buffer può liberamente compiere operazioni interne $passa_n$ che causano un "effetto cascata". Quando un elemento viene inserito nella prima cella questa lo passa alla seconda, che lo passa alla terza e via così fino all'ultima, dalla quale può essere rimosso.
Le operazioni interne al buffer sono state rappresentate informalmente con il termine $\tau_{passa[n]}$ per distinguerle chiaramente dalle operazioni svolte dal produttore e dal consumatore, rispettivamente τ_{get} e τ_{put}.
Notiamo che le operazioni *passa* non sono obbligate, un elemento potrebbe rima-

nere in una cella differente dall'ultima. Questa proprietà ha una conseguenza, un produttore potrebbe voler effettuare una *put* quando gli elementi non sono per forza accodati verso il fondo, e ciò gli sarebbe permesso in quanto il la prima cella del buffer è libera. Tale operazione rende impossibile la rappresentazione del *derivation graph* di un buffer a N posizioni dato che genera infinite possibili τ_{get} operazioni, tutte le possibili interfogliazioni tra le operazioni svolte dal produttore e le operazioni del buffer sono infinite. É interessante notare come questo problema non si pone nel caso opposto, se un consumatore vuole estrarre un elemento deve attendere che il buffer abbia concluso tutte le sue operazioni interne, altrimenti non vi è alcun elemento nell' ultima cella. Nelle implementazioni successive assumeremo per semplicità che il buffer sia a 3 posizioni per ottenere un numero di τ_{passa} finito pur continuando a mostrare la situazione in cui un elemento viene inserito o rimosso dal buffer prima che le operazioni di "spostamento" siano state concluse.

1.2 1 produttore, 2 consumatori, buffer a N posizioni

Il sistema in questo caso è molto simile al sitema del caso precedente, l'unica aggiunta è la concatenazione di un ulteriore Cons.

$SYS = (Prod||Cons||Cons||(Buffer))/_{\{put,get,passa\}}$

$Prod = think.produce.\overline{PUT}.Prod$

$Cons = think.produce.\overline{GET}.Cons$

funzioni di Relabeling:

$$f_E\begin{bmatrix} put \to put \\ get \to passa_1 \end{bmatrix}$$

$$f_F\begin{bmatrix} put \to \overline{passa}_{n-1} \\ get \to passa_n \end{bmatrix} \forall n \in [1, N]$$

$$f_E\begin{bmatrix} put \to put \\ get \to \overline{passa}_n \end{bmatrix}$$

$$Buffer = \underbrace{E_{[f_E]}||...||E_{[f_M]}||...||E_{[f_F]}}_{N \text{ volte}}$$

$E = PUT.F$

$F = GET.E$

In questo *derivation graph* si è considerata solo la situazione in cui il buffer è limitato a 3 posizioni, dato che tale assunto semplifica la lettura ed in ogni caso la situazione con buffer ad N posizioni è stata già discussa nella sezione precendente (sezione 1.1). La presenza di un ulteriore consumatore non modifica molto il grafo, infatti aggiunge solo ad ogni stato (escluso lo stato con il buffer completamente vuoto) la possibilità di effettuare due distinte operazioni τ_{get}, una per ogni *Consumer*. Entrambe le operazioni τ_{get}, nonostante siano svolte da due *Consumer* differenti, conducono a due stati indistinguibili che ho quindi accorpato in uno stato solo.

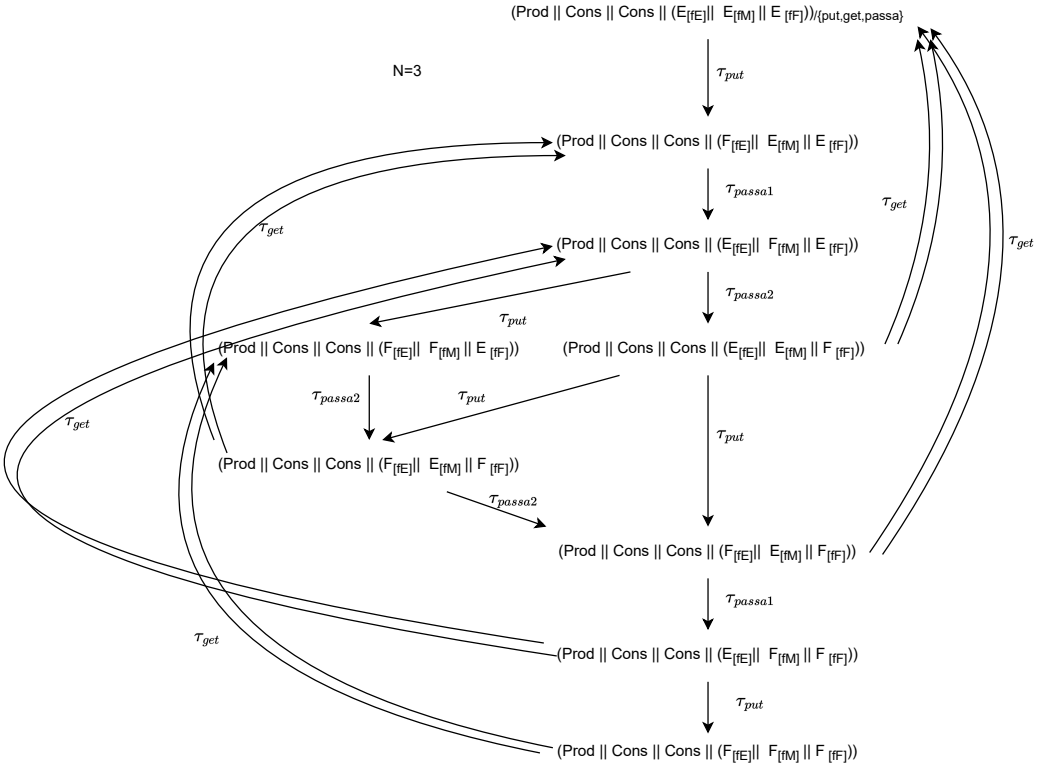

Figura 2: Setting2

1.3 P produttori, C consumatori, buffer a N posizioni

Questo caso comporta lo stesso problema di implementazione (raffigurare un buffer a N posizioni), anche per i *Produttori* ed i *consumatori*.

$$SYS = (\underbrace{Prod||...||Prod}_{P\ volte}||\underbrace{Cons||...||Cons}_{C\ volte}||(Buffer))/_{\{put,get,passa\}}$$

$$Prod = think.produce.\overline{PUT}.Prod$$

$$Cons = think.produce.\overline{GET}.Cons$$

funzioni di Relabeling:

$$f_E\begin{bmatrix} put \to put \\ get \to passa_1 \end{bmatrix}$$

$$f_F\begin{bmatrix} put \to \overline{passa}_{n-1} \\ get \to passa_n \end{bmatrix} \forall n \in [1, N]$$

$$f_E\begin{bmatrix} put \to put \\ get \to \overline{passa}_n \end{bmatrix}$$

$$Buffer = \underbrace{E_{[f_E]}||...||E_{[f_M]}||...||E_{[f_F]}}_{N \text{ volte}}$$

$$E = PUT.F$$

$$F = GET.E$$

Fortunatamente (come nel caso di sezione 1.2) tutte le operazioni svolte dai molteplici *Produttori e Consumatori* conducono a stati tra loro indistinguibili. L'unica differenza rilevante sta nel numero di τ-operazioni eseguibili, in figura rappresentate tramite frecce con righe composte da puntini.Da ogni stato avente "la prima" cella del buffer vuota sarà possibile effettuare P operazioni τ_{put} e da ogni stato avente "l' ultima" cella piena è sarà possibile effettuare C operazioni

τ_{get}.
Come nel caso 1.2 da ogni stato avente almeno un elemento nel buffer è possibile effettuare $N - elements - 1$ τ_{passa}-transizioni, ove $elements$ è il numero di celle del buffer occupate.

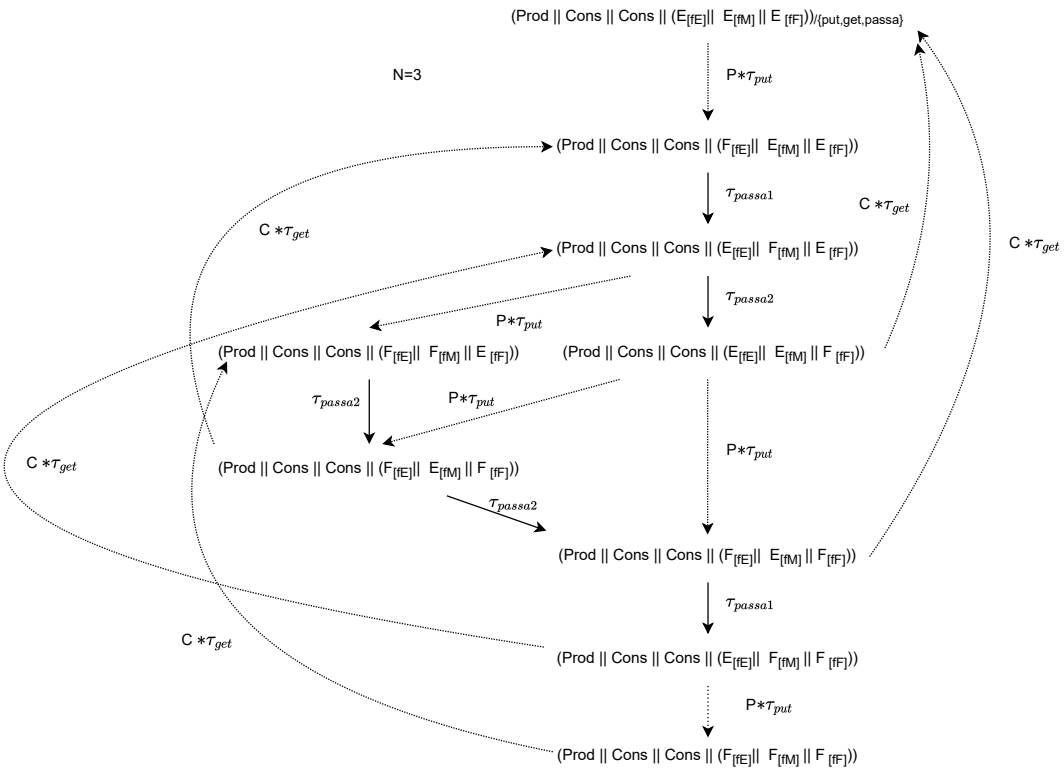

Figura 3: Setting3

2 NuSMV

2.1 1 produttore, 1 consumatore, buffer a N posizioni

```
1  MODULE main
2  VAR
3          buffer : 0 .. 3;
4          prod : process producer(buffer);
5          cons : process consumer(buffer);
6  ASSIGN
7          init(buffer) := 0;
8
9
10 MODULE producer(buffer)
11 VAR
12         state : {think, produce, place};
13 ASSIGN
14         init(state) := think;
15         next(state) :=
16           case
17                 state = think : produce;
18                 state = produce : place;
19                 state = place & !(buffer = 3): think;
20                 TRUE : state;
21           esac;
22         next(buffer) :=
23           case
24                 state = place & !(buffer = 3): buffer + 1;
25                 TRUE : buffer;
26           esac;
27 MODULE consumer(buffer)
28 VAR
29         state : {think, consume, take};
30 ASSIGN
31         init(state) := think;
32         next(state) :=
33           case
34                 state = take & !(buffer = 0): consume;
35                 state = think : take;
36                 state = consume : think;
37                 TRUE : state;
38           esac;
39         next(buffer) :=
40           case
41                 state = take & !(buffer = 0): buffer -1;
42                 TRUE : buffer;
43           esac;
```

In questo programma il buffer è identificato da una variabile intera che può assumere valori compresi tra 0 e 3, volendo incrementare la dimensione del buffer bisogna modificare il valore 3 inserendo il valore desiderato. Non c'è quindi la possibilità di impostare il valore del buffer in maniera interattiva ma quantomeno lo si può gestire parametricamente.
I due processi ricevono come parametro il buffer ed hanno una semplice implementazione dei passaggi di stato: l'unica limitazione inserita è nell'uscita dallo stato `place`/`take` che corrisponde all'operazione di aggiunta/rimozione di un elemento dal buffer.
Il *Consumer* non può effettuare l'operazione *get* se il buffer è vuoto (riga 36), così come il *producer* non puà effettuare una *put* se il buffer è già pieno (riga 19).

Invece il buffer viene incrementato o decrementato solo quando un processo effettua una operazione su di esso (righe 24 e 41), la sintassi di *NuSMV* obbliga l'inserimento di una clausola che impedisca al buffer di superare il valore 3 o scendere sotto il valore 0.

Lanciando il comando `print_reachable_states` risulta che il numero di stati raggiungibili è 36 *out of* 36, e varia a seconda della dimensione del buffer in

maniera linearmente proporzionale, in accordo con quanto ottenuto dallo studio del reachability graph nelle PN. Ci sono invece molti più stati di quanti siano i marking del *Reachability Graph*, probabilmente perchè il numero di posti non è equivalente al numero di "stati" dei processi producer,consumer e buffer. Ad esempio il processo producer ha 3 stati: "think, produce, place" ma la mia implementazione in PN ha solo 2 posti "ThinkP e PlaceInBuffer", se avessi mantenuto una implementazione cooerente tra i due strumenti avrei ottenuto lo stesso valore sia calcolando gli stati raggiungibili in NuSMV che calcolando il numero di markings nel reachability graph.

2.2 1 produttore, 2 consumatori, buffer a N posizioni

```
MODULE main
VAR
        buffer : 0 .. 3;
        prod1 : process producer(buffer);
        prod2 : process producer(buffer);
        cons1 : process consumer(buffer);
        cons2 : process consumer(buffer);
ASSIGN
        init(buffer) := 0;


MODULE producer(buffer)
VAR
        state : {think, produce, place};
ASSIGN
        init(state) := think;
        next(state) :=
          case
                state = think : produce;
                state = produce : place;
                state = place & !(buffer = 3): think;
                TRUE : state;
          esac;
        next(buffer) :=
          case
                state = place & !(buffer = 3): buffer + 1;
                TRUE : buffer;
          esac;
MODULE consumer(buffer)
VAR
        state : {think, consume, take};
ASSIGN
        init(state) := think;
        next(state) :=
          case
                state = take & !(buffer = 0): consume;
                state = think : take;
                state = consume : think;
                TRUE : state;
          esac;
        next(buffer) :=
          case
                state = take & !(buffer = 0): buffer -1;
                TRUE : buffer;
          esac;
```

Come nelle altre implementazioni il setting 2 è praticamente identico al setting 1. In questo caso la differenza sta nell'introduzione di una nuova variabile Cons2, istanza di `process consumer`. Si può però notare che non è necessaria alcuna forma di mutua esclusione esplicita in quanto le operazioni di `place` e `take` sono intese come operazioni atomiche.

Come nel caso precendente il comando `print_reachable_states` fornisce un valore che non corrisponde al valore ottenuto calcolando i markings della corrispondente Petri Net, per lo stesso motivo sicusso prima. In questo caso otteniamo 108 stati raggiungibili mentre il reachability graph della PN corrispondente identifica 32 markings.Se io avessi aggiunto un posto alla rete del consumatore ed un posto alla rete del produttore e poi effettuato una scalatura per ripetizioni di sottoreti avrei ottenuto un RG avente esattamente 108 markings.

2.3 P produttori, C consumatori, buffer a N posizioni

Questa implementazione non è possibile, in quanto *NuSMV* non permette la creazione di vettori di processi, l'unico modo di generare P produttori e C consumatori è quindi scrivere P variabili di tipo `process prod` e C variabili di tipo `process cons`.