



# Linguaggi per il global computing

## Esercizi B e D + Barbershop

Federico Perin - 2029215 - Ottobre 2021

# Indice

<b>1</b>	<b>Esercizi</b>	<b>2</b>
1.1	Esercizio B	2
1.1.1	Sintassi	2
1.1.2	Dimostrazione per somme finite	2
1.1.2.1	Punto 1	2
1.1.2.1.1	Dimostrazione	3
1.1.2.2	Punto 2	4
1.1.2.2.1	Dimostrazione	5
1.1.2.3	Conclusione prima parte	6
1.1.3	Dimostrazione per somme infinite	6
1.1.4	Conclusione	8
1.2	Esercizio D	9
1.2.1	Dimostrazione	9
1.2.1.1	Prefisso $C[] = \alpha$ .	9
1.2.1.2	Contesto non deterministico $C[] = ( \quad + R )$	9
1.2.1.3	Contesto parallelo $C[] = ( \quad   R )$	10
1.2.1.4	Contesto restrizione $C[] = \backslash L$	13
1.2.1.5	Contesto relabelling $C[] = [f]$	14
1.2.2	Conclusione	15
<b>2</b>	<b>Barbershop</b>	<b>16</b>
2.1	Una possibile soluzione	16
2.2	Modellazione in CCS	17
2.2.1	Codifica Mutex e contattore clienti	18
2.2.2	Codifica Customer	18
2.2.3	Codifica Barber	19
2.2.4	Codifica del sistema	19
2.3	Verifica della correttezza attraverso CWB	19
2.3.1	Weak Bisimulation and Trace Equivalence	19
2.3.2	Assenza di deadlock	19
2.3.3	Mutua esclusione contattore	19
2.3.4	Mutua esclusione nell'esecuzione del taglio	19
2.3.5	Verifica comportamento del cliente che entra quando il negozio pieno	19

# 1 Esercizi

## 1.1 Esercizio B

Dimostrare che ogni processo CCS finito termina in un numero finito di passi.

### 1.1.1 Sintassi

$$P, Q ::= \alpha.P \mid (P \mid Q) \mid \sum_{i \in I} P_i \mid P \setminus L \mid P[f] \mid \mathbf{0}$$

#### Note:

Nel CCS finito non sono previste le costanti  $\mathcal{K}$ .

Il processo  $\mathbf{0}$  ha un solo stato e non ha interazioni con altri processi.

La dimostrazione viene divisa in due casi:

- Nel primo caso si hanno somme finite, cioè l'insieme  $I$  contenente le scelte della somma non deterministica, è finito;
- Nel secondo caso l'insieme  $I$  sarà infinito.

### 1.1.2 Dimostrazione per somme finite

La dimostrazione prevede i seguenti punti:

1. Ogni processo del CCS finito termina con un numero finito di passi;
2. Ogni processo del CCS finito ha un numero finito di stati.

#### 1.1.2.1 Punto 1

Dato che non esistono costanti  $\mathcal{K}$ , non è possibile rigenerare passi eseguiti in precedenza, perciò dopo un certo numero finito di passi ogni processo  $P$  in CCS finito terminerà perché non avrà più passi da eseguire. Si deduce perciò che ogni processo ha un numero di passi limitato da un limite superiore è quindi tale numero è finito.

Per dimostrare quanto scritto si procede attraverso una dimostrazione induttiva sull'altezza di derivazione di un processo  $P$ , con ipotesi induttiva:  $P \xrightarrow{\alpha} P' \Rightarrow \text{Size}_p(P') < \text{Size}_p(P)$ , dove  $\text{Size}_p(P')$  si intende il numero di passi del processo  $P$  dopo aver fatto l'azione  $\alpha$ .

Definiamo  $\text{Size}_p(P)$ :

$$\text{Size}_p(P) = \begin{cases} P = \alpha.R, & 1 + \text{Size}_p(R) \\ P = \sum_{i \in I} P_i, & \max(\text{Size}_p(P_i)) \\ P = R \mid Q, & \text{Size}_p(R) + \text{Size}_p(Q) \\ P = R \setminus L, & \text{Size}_p(R) \\ P = R[f], & \text{Size}_p(R) \end{cases}$$

Si dimostrerà che  $\text{Size}_p(P)$  indica il limite superiore del numero di passi eseguiti dal processo  $P$  per terminare.

### 1.1.2.1.1 Dimostrazione

#### Caso Base:

$$\frac{\alpha.P \xrightarrow{\alpha} P}{\text{ACT}} \quad \text{Size}_p(\alpha.P) = 1 + \text{Size}_p(P) > \text{Size}_p(P)$$

Si ha che l'azione  $\alpha$  concatenata al processo  $P$  aggiunge un passo in più, perciò risulta essere corretto il limite superiore  $\text{Size}_p(\alpha.P) = 1 + \text{Size}_p(P)$  passi.

#### Caso Induttivo:

$$* \frac{P_j \xrightarrow{\alpha} P'}{\sum_{i \in I} P_i \xrightarrow{\alpha} P'} \text{SUM } j \in I$$

Si osserva che nell'esecuzione di un passo  $\alpha$ , per ipotesi induttiva vale che:  $\text{Size}_p(P_j) > \text{Size}_p(P')$ .

Perciò  $\text{Size}_p(\sum_{i \in I} P_i) > \text{Size}_p(P_j) > \text{Size}_p(P')$  allora  $\max(\text{Size}_p(P_i)) > \text{Size}_p(P')$

Il limite superiore  $\text{Size}_p(\sum_{i \in I} P_i) = \max(\text{Size}_p(P_i))$  passi, risulta essere corretto.

$$* \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \text{PAR-L}$$

Si osserva che nell'esecuzione di un passo  $\alpha$ , per ipotesi induttiva vale che:  $\text{Size}_p(P) > \text{Size}_p(P')$ .

E quindi dato che  $\text{Size}_p(P|Q) = \text{Size}_p(P) + \text{Size}_p(Q)$  mentre  $\text{Size}_p(P'|Q) = \text{Size}_p(P') + \text{Size}_p(Q)$ , per ipotesi induttiva  $\text{Size}_p(P|Q) > \text{Size}_p(P'|Q)$

$$* \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'} \text{PAR-R}$$

Si osserva che nell'esecuzione di un passo  $\alpha$ , per ipotesi induttiva vale che:  $\text{Size}_p(Q) > \text{Size}_p(Q')$

E quindi dato che  $\text{Size}_p(P|Q) = \text{Size}_p(P) + \text{Size}_p(Q)$  mentre  $\text{Size}_p(P|Q') = \text{Size}_p(P) + \text{Size}_p(Q')$ , per ipotesi induttiva  $\text{Size}_p(P|Q) > \text{Size}_p(P|Q')$

$$* \frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{\bar{\alpha}} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \text{PAR-}\tau$$

Si osserva che nell'esecuzione di un passo di sincronizzazione, per ipotesi induttiva vale che:  $\text{Size}_p(P) > \text{Size}_p(P')$  e  $\text{Size}_p(Q) > \text{Size}_p(Q')$

E quindi dato che  $\text{Size}_p(P|Q) = \text{Size}_p(P) + \text{Size}_p(Q)$  mentre  $\text{Size}_p(P'|Q') = \text{Size}_p(P') + \text{Size}_p(Q')$ , per ipotesi induttiva  $\text{Size}_p(P|Q) > \text{Size}_p(P'|Q')$

Perciò si è dimostrato che il limite superiore  $Size_p(P|Q) = Size_p(P) + Size_p(Q)$  passi, risulta essere corretto in tutti i tre casi PAR-L, PAR-R e PAR- $\tau$ .

$$* \frac{P \xrightarrow{\alpha} P'}{P \setminus L \xrightarrow{\alpha} P' \setminus L} RES \text{ se } \alpha, \bar{\alpha} \notin L$$

Si osserva che nell'esecuzione di un passo  $\alpha$ , per ipotesi induttiva vale che:  $Size_p(P) > Size_p(P')$

Applicare una restrizione ad un processo non fa aumentare il numero massimo di passi di esecuzione, ma avere una possibile variazione delle possibili interazioni con altri processi in parallelo, vale che:

$Size_p(P \setminus L) = Size_p(P)$  mentre  $Size_p(P' \setminus L) = Size_p(P')$ , per ipotesi induttiva  $Size_p(P \setminus L) > Size_p(P' \setminus L)$

Perciò si è dimostrato che il limite superiore  $Size_p(P \setminus L) = Size_p(P)$  passi, risulta essere corretto.

$$* \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]} REL$$

Si osserva che nell'esecuzione di un passo  $\alpha$ , per ipotesi induttiva vale che:  $Size_p(P) > Size_p(P')$

Applicare un relabelling ad un processo non fa aumentare il numero massimo di passi di esecuzione, ma avere una possibile variazione delle possibili interazioni con altri processi in parallelo, vale che:

$Size_p(P[f]) = Size_p(P)$  mentre  $Size_p(P'[f]) = Size_p(P')$ , per ipotesi induttiva  $Size_p(P[f]) > Size_p(P'[f])$

Perciò si è dimostrato che il limite superiore  $Size_p(P[f]) = Size_p(P)$  passi, risulta essere corretto.

### 1.1.2.2 Punto 2

Si vuole dimostrare che ogni processo CCS ha un numero finito di stati, cioè esiste un limite superiore del numero di stati di esecuzione per un processo P.

Definiamo  $Size_s(P)$ :

$$Size_s(P) \begin{cases} P = \mathbf{0}, & 1 \\ P = \alpha.R, & 1 + Size_s R \\ P = \sum_{i \in I} P_i, & \sum_{i \in I} Size_s P_i \\ P = R \mid Q, & Size_s R * Size_s Q \\ P = R \setminus L, & Size_s R \\ P = R[f], & Size_s R \end{cases}$$

Si dimostrerà di seguito che  $Size_s(P)$  calcola il limite superiore del numero di stati dell'esecuzione del processo P.

### 1.1.2.2.1 Dimostrazione

Tramite dimostrazione per induzione sull'esecuzione di un processo  $P$  si dimostra che i limiti di  $Size_s(P)$  sono corretti.

#### Caso Base:

**0**

Il processo **0** ha un solo stato, quello di partenza, per definizione quindi  $Size_s(\mathbf{0}) = 1$  stato.

#### Caso Induttivo:

In un passo si raggiunge uno sotto processo dal quale poi in  $n$  passi finiti si raggiungerà un stato terminante. Si utilizza la seguente ipotesi induttiva:

Un processo composto da sotto processi con un limite superiore ai stati di esecuzione ha anch'esso un limite superiore di stati di esecuzione.

Tale ipotesi scritta verrà utilizzata per dimostrare che il sotto processo raggiunto, da lui allo stato terminante, ci saranno un numero finiti di stati perché limitati da un limiti superiore. Si ricorda inoltre che il numero di stati raggiunti è finito se gli  $n$  passi sono finiti. Grazie alla dimostrazione del punto 1 si sa che  $n$  è finito.

#### \* ACT

Con l'azione  $\alpha$  concatenata al processo  $P$  viene aggiunto a  $P$  uno stato in più, infatti  $\alpha.P \xrightarrow{\alpha} P$ .  $P$  per ipotesi induttiva ha al più  $Size_s(P)$  stati, quindi  $\alpha.P$  avrà al più  $Size_s(\alpha.P) = 1 + Size_s(P)$  stati.

#### \* SUM

Si ha la seguente esecuzione:  $\sum_{i \in I} P_i \xrightarrow{\alpha} P'$ , dove qualunque  $P_i$  sia scelto per ipotesi induttiva ha un numero finito di stati d'esecuzione. Quindi tutti i processi che compongono la somma non deterministica hanno un numero finito di stati. Perciò  $\sum_{i \in I} P_i$  ha al più  $Size_s(\sum_{i \in I} P_i) = \sum_{i \in I} Size_s(P_i)$  stati inoltre, dato che il numero di processi  $P_i$  è finito, anche la somma dei stati lo sarà.

È corretto il limite superiore presentato precedentemente perché potrebbero esserci stati non condivisi tra i vari processi  $P_i$ , quindi tutti i  $Size_s(P_i)$ .

#### \* PAR-L

Si ha la seguente esecuzione:  $P|Q \xrightarrow{\alpha} P'|Q$ , dove  $P|Q$  raggiunge lo stato  $P'|Q$ .

Per ipotesi induttiva  $P'|Q$  ha un numero finito di stati d'esecuzione, cioè ha al più  $Size_s(P'|Q) = Size_s(P') * Size_s(Q)$  stati, quindi  $P|Q$  ha al più  $Size_s(P|Q) = (Size_s(P') + 1) * Size_s(Q)$  stati. Dato che è possibile che non ci sia alcun stato condiviso durante l'esecuzione dei due processi, il limite superiore ad essi è uguale al totale delle combinazioni possibili tra gli stati dei due singoli processi.

#### \* PAR-R

Si ha la seguente esecuzione:  $P|Q \xrightarrow{\alpha} P|Q'$ , dove  $P|Q$  raggiunge lo stato  $P|Q'$ .

Per ipotesi induttiva  $P|Q'$  ha un numero finito di stati d'esecuzione, cioè ha al più  $Size_s(P|Q') = Size_s(P) * Size_s(Q')$  stati, quindi  $P|Q$  ha al più  $Size_s(P|Q) = Size_s(P) * (Size_s(Q') + 1)$  stati. Dato che è possibile che non ci sia alcun stato condiviso durante l'esecuzione dei due processi, il limite superiore ad essi è uguale al totale delle combinazioni possibili tra gli stati dei due singoli processi.

\* **PAR- $\tau$**

Si ha la seguente esecuzione:  $P|Q \xrightarrow{\tau} P'|Q'$ , dove  $P|Q$  raggiunge lo stato  $P'|Q'$ .

Per ipotesi induttiva  $P'|Q'$  ha un numero finito di stati d'esecuzione, cioè ha al più  $Size_s(P'|Q') = Size_s(P') * Size_s(Q')$  stati, quindi  $P|Q$  ha al più  $Size_s(P|Q) = Size_s(P') * Size_s(Q') + 1$  stati. Dato che è possibile che non ci sia alcun stato condiviso durante l'esecuzione dei due processi, il limite superiore ad essi è uguale al totale delle combinazioni possibili tra gli stati dei due singoli processi.

Dunque  $Size_s(P|Q) = Size_s(P) * Size_s(Q)$  stati, risulta essere corretto.

\* **RES**

Si ha la seguente esecuzione:  $P \setminus L \xrightarrow{\alpha} P' \setminus L$ .

Per ipotesi induttiva  $P' \setminus L$  ha al più  $Size_s(P' \setminus L) = Size_s(P')$  stati.

Dato che  $Size_s(P' \setminus L)$  è finito, allora  $P \setminus L$  avrà al più:

$Size_s(P \setminus L) = 1 + Size_s(P' \setminus L) = 1 + Size_s(P') = Size_s(P)$  stati, perché applicando una funzione di restrizione non si aumentano il numero massimo di stati d'esecuzione, vale quindi il limite superiore  $Size_s(P \setminus L) = Size_s(P)$ .

\* **REL**

Si ha la seguente esecuzione:  $P[f] \xrightarrow{f(\alpha)} P'[f]$ .

Per ipotesi induttiva  $P'[f]$  ha al più  $Size_s(P'[f]) = Size_s(P')$  stati.

Dato che  $Size_s(P'[f])$  è finito, allora  $P[f]$  avrà al più:

$Size_s(P[f]) = 1 + Size_s(P'[f]) = 1 + Size_s(P') = Size_s(P)$  stati, perché applicando una funzione di relabelling non si aumentano il numero massimo di stati d'esecuzione ma si cambiano solo le possibili interazioni con altri processi in parallelo, vale quindi il limite superiore  $Size_s(P[f]) = Size_s(P)$ .

### 1.1.2.3 Conclusione prima parte

Si è dimostrato nei punti precedenti che i processi definiti attraverso il CCS finito hanno sempre un limite superiore sia per il numero di passi e sia per il numero di stati d'esecuzione, di conseguenza il numero di passi e stati sono entrambi finiti.

### 1.1.3 Dimostrazione per somme infinite

Ora per quanto riguarda la somma non deterministica  $\sum_{i \in I} P_i$ , l'insieme  $I$  sarà infinito.

Infatti i processi CCS che fanno parte della somma non deterministica, saranno infiniti e quindi hanno un numero infinito di stati ma hanno tutti una derivazione finita.

La dimostrazione sarà simile alla precedente per tutti i vari punti, tranne per il punto riguardante la somma non deterministica, infatti per dimostrare la finitezza dell'esecuzione si utilizzerà un'astrazione basata sulle generazioni della grammatica di CCS.

Si dimostra perciò per induzione sulla lunghezza di derivazione che ogni processo CCS termina in numero finito di passi.

**Caso Base:**

**0**

Il processo **0** termina in 0 passi per definizione.

**Caso Induttivo:**

La generazione della sequenza di interazioni avanza attraverso uno dei termini della grammatica e come ipotesi induttiva si ha che tale sequenza ha derivazione finita.

\* **ACT, RES e REL**

Con l'azione  $\alpha$  concatenata al processo  $P$  si ha che, la derivazione produrrà un processo del tipo,  $\alpha^{n+1}.P$  con  $n \geq 0$ , dove esiste un nuovo passo e uno nuovo stato nell'esecuzione di  $P$ . Perciò il numero di stati e di passi d'esecuzione rimangono finiti.

\* **PAR-L, PAR-R, PAR- $\tau$**

Come dimostrato per il caso del CCS finito con somme finite, nell'esecuzione parallela la derivazione produrrà dei sotto processi che per ipotesi induttiva sono finiti e quindi il processo di cui fanno parte sarà anch'esso finito.

\* **RES**

Con **RES** si avrà che la derivazione produrrà un certo numero finito di restrizioni. Perciò si avranno sempre passi e stati in numero finito e quindi vale quanto dimostrato nel CCS finito con somme finite.

\* **REL**

Con **REL** si avrà che la derivazione produrrà un certo numero finito di relabelling. Perciò si avranno sempre passi e stati in numero finito e quindi vale quanto dimostrato nel CCS finito con somme finite.

\* **SUM** In questo caso il processo derivato sarà del tipo  $P_1 + P_2 + \dots$ . Per ipotesi induttiva ogni  $P_i$  ha una derivazione di lunghezza finita, perciò si può dimostrare che la derivazione del processo  $P$  è finita per induzione sul numero di sotto processi che vengono usati nella somma.

**Caso Base:**

Processo **0** per definizione è finito.

**Caso Induttivo:**

L'ipotesi induttiva afferma che la scelta non deterministica tra i processi ha una derivazione di lunghezza finita. Inoltre si deve tener conto che nella scelta non deterministica, viene scelto un solo processo da eseguire tra tutti quelli presenti.

Quindi aggiungendo il processo  $P_{n+1}$  alla scelta, che per ipotesi induttiva anch'esso ha derivazione finita; la scelta diventerà tra l'esecuzione di uno



dei processi già presenti e il processo appena aggiunto  $P_{n+1}$ . Solo uno dei processi verrà eseguito e quindi la derivazione avrà una lunghezza superiormente limitata dalla massima lunghezza di derivazione dei sotto processi. Si è perciò dimostrato che l'insieme dei processi della scelta non deterministica può essere illimitato ma finito senza perdere la finitezza di esecuzione. Questo perché la derivazione esegue solo uno dei processi della scelta come scritto in precedenza e in più si sa che l'altezza della derivazione di  $P$  è limitata superiormente dall'altezza della derivazione del processo con l'altezza maggiore + 1, ovvero  $\max_h(P_i) + 1$ .

Purtroppo però il numero di stati che si ha durante l'esecuzione non è più limitato superiormente, perché essendo il limite superiore dato dalla  $\sum_{i \in I} P_i$  con  $I$  infinito, il numero di processi CCS risulta essere infinito, e quindi la somma dei stati d'esecuzione sarà anch'essa infinita. Perciò il numero di stati non è limitato superiormente.

#### 1.1.4 Conclusione

È stato dimostrato che ogni processo CCS finito termina in un numero finito di passi, indipendente dal fatto si utilizzi una grammatica che permette scelte non deterministiche in un insieme o infinito o finito. Si sottolinea che scelta di un insieme infinito o finito determinerà, se il numero di stati d'esecuzione raggiungibili sarà finito o infinito.

## 1.2 Esercizio D

Dimostrare che la trace equivalence è una congruenza per il CCS.

Prima di illustrare la dimostrazione si definisce che cosa si intende con i concetti di trace equivalence e congruenza.

Innanzitutto per traces di un processo  $P$  che di seguito verrà indicata con  $\text{Tr}(P)$  si intende, le sequenze di interazioni  $\alpha_1 \dots \alpha_n \in \text{Act}$  con  $n \geq 0$  tale che esiste una sequenza di transizioni  $P \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} P_n$ , e quindi rappresentata tutte le possibili interazioni con un processo. Più formalmente  $\text{Tr}(P) = \{ \alpha_1 \dots \alpha_n \mid P \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} P_n \}$ . Quindi due processi  $P$  e  $Q$  si dicono trace equivalence  $P \sim_t Q$  se  $\text{Tr}(P) = \text{Tr}(Q)$ .

Per congruenza si intende, dati due processi  $P$  e  $Q$  in relazione tra loro ( $P \ R \ Q$ ), allora per ogni contesto  $C[\ ]$ ,  $C[P] \ R \ C[Q]$ .

Perciò si dimostrerà che se  $P \sim_t Q \Rightarrow \forall C[\ ] \ C[P] \sim_t C[Q]$ .

### 1.2.1 Dimostrazione

Siano  $P, Q$  e  $R$  processi CCS con  $P \sim_t Q$ , allora

1.  $\alpha.P \sim_t \alpha.Q$
2.  $P + R \sim_t Q + R$
3.  $P|R \sim_t Q|R$
4.  $P \setminus L \sim_t Q \setminus L$
5.  $P[f] \sim_t Q[f]$

#### 1.2.1.1 Prefisso $C[\ ] = \alpha.$

Si ha che  $\text{Tr}(C[P]) = \text{Tr}(\alpha.P) = \alpha.\text{Tr}(P)$ , dato che con il contesto  $C[\ ]$  si è aggiunto l'interazione  $\alpha$  alle  $\text{Tr}(P)$ . Per ipotesi  $\text{Tr}(P) = \text{Tr}(Q)$ , inoltre aggiungendo l'interazione  $\alpha$  sia a  $\text{Tr}(P)$  e sia a  $\text{Tr}(Q)$ , si ha che  $\alpha.\text{Tr}(P) = \alpha.\text{Tr}(Q) = \text{Tr}(\alpha.Q)$ .

Perciò vale  $\alpha.P \sim_t \alpha.Q$ .

#### 1.2.1.2 Contesto non deterministico $C[\ ] = ( \ + \ R )$

Nel caso del contesto non deterministico tra i processi  $P$  e  $R$  le  $\text{Tr}(P + R) = \text{Tr}(P) \cup \text{Tr}(R)$ . Se questo è vero, dato che per ipotesi  $\text{Tr}(P) = \text{Tr}(Q)$  e  $\text{Tr}(Q + R) = \text{Tr}(Q) \cup \text{Tr}(R)$ , allora  $\text{Tr}(P + R) = \text{Tr}(Q + R)$ .

Perciò si deve dimostrare che il contesto non deterministico tra i processi  $P$  e  $R$  è uguale alla unione delle traccie dei due processi. Dimostrato questo ne consegue la veridicità di  $P + R \sim_t Q + R$ .

( $\subseteq$ )

Sia  $t \in \text{Tr}(P + R) \Rightarrow t \in (\text{Tr}(P) \cup \text{Tr}(R))$  Per induzione su  $|t|$ :

**Caso Base**  $|t| = 0$

Allora  $t = \varepsilon \in (\text{Tr}(P) \cup \text{Tr}(R))$

### Caso Induttivo $|t| = n + 1$

$t$  ha una prima interazione seguita poi dalla traccia  $t'$ , quindi  $P + R \xrightarrow{\alpha_1} X' \xrightarrow{t'}$  ovvero viene applicata una transizione secondo la regola della somma non deterministica arrivando in certo processo  $X'$ , ci sono perciò due possibilità:

- $P+R$  ha effettuato una transizione usando la regola SUM-L:

$$\frac{P \xrightarrow{\alpha_1} P'}{P + R \xrightarrow{\alpha_1} P' \xrightarrow{t'}} \text{ SUM-L}$$

$t = \alpha_1.t'$  con  $|t'| = n$ , per ipotesi induttiva  $t' \in \text{Tr}(P')$ , quindi  $\alpha_1.t' \in \alpha_1.\text{Tr}(P') \subseteq \text{Tr}(P)$  allora  $t \in \text{Tr}(P) \cup \text{Tr}(R)$

- $P+R$  ha effettuato una transizione usando la regola SUM-R:

$$\frac{R \xrightarrow{\alpha_1} R'}{P + R \xrightarrow{\alpha_1} R' \xrightarrow{t'}} \text{ SUM-R}$$

$t = \alpha_1.t'$  con  $|t'| = n$ , per ipotesi induttiva  $t' \in \text{Tr}(R')$ , quindi  $\alpha_1.t' \in \alpha_1.\text{Tr}(R') \subseteq \text{Tr}(R)$  allora  $t \in \text{Tr}(R) \cup \text{Tr}(P)$

( $\supseteq$ )

Sia  $t \in (\text{Tr}(P) \cup \text{Tr}(R)) \Rightarrow t \in \text{Tr}(P + R)$

Perciò  $t$  può essere una traccia sia di  $P$  e sia di  $R$  oppure solo uno dei due, quindi:

Se  $t \in \text{Tr}(P)$ ,  $P + R$  può scegliere di fare una transizione attraverso la regola SUM-L, e allora vale che  $t \in \text{Tr}(P + R)$ .

Se  $t \in \text{Tr}(R)$ ,  $P + R$  può scegliere di fare una transizione attraverso la regola SUM-R, e allora vale che  $t \in \text{Tr}(P + R)$ .

Se  $t$  appartiene sia a  $P$  che  $R$ , qualsiasi regola venga applicata per fare la transizione vale sempre  $t \in \text{Tr}(P + R)$ .

Perciò si è dimostrato che  $\text{Tr}(P + R) = \text{Tr}(P) \cup \text{Tr}(R)$  e quindi con  $\text{Tr}(P) = \text{Tr}(Q)$ ,  $P + R \sim_t Q + R$  come si voleva dimostrare.

#### 1.2.1.3 Contesto parallelo $C[ ] = ( \quad | R )$

Intuitivamente le tracce  $\text{Tr}(P|R)$  sono tutte le possibili combinazioni tra  $\text{Tr}(P)$  e  $\text{Tr}(R)$ , cioè quindi tutte le loro interazioni e sincronizzazioni. Se tale intuizione è vera allora dato che  $\text{Tr}(P) = \text{Tr}(Q)$ , si potrebbe sostituire  $P$  con  $Q$  nelle  $\text{Tr}(P|R)$  ed ottenere le stesse combinazioni della versione precedente, quindi varrebbe che  $\text{Tr}(P|R) = \text{Tr}(Q|R)$  e di conseguenza  $P|R \sim_t Q|R$ .

Per dimostrare ciò, si deve prima dimostrare il seguente lemma che sarà utilizzato nella dimostrazione:

Siano  $A$  e  $B$  due processi CSS, se  $\text{Tr}(A) = \text{Tr}(B) \Rightarrow \forall \alpha \text{Tr}(A') \subseteq \text{Tr}(\sum \{B' | B \xrightarrow{\alpha} B'\})$

con  $A \xrightarrow{\alpha} A'$

Cioè se i processi  $A$  e  $B$  hanno le stesse tracce allora le tracce del sotto processo di  $A$ ,

A' sono incluse nell'insieme delle tracce relative ai sotto processi B', raggiunti con una transizione  $\alpha$  dal processo B.

Si dimostra di seguito tale lemma:

$$\text{Tr}(\sum \{B'|B \xrightarrow{\alpha} B'\}) = \{t'|t = \alpha.t' \in \text{Tr}(B)\}, \text{ dato che } \text{Tr}(A) = \text{Tr}(B)$$

allora posso sostituire le  $\text{Tr}(B)$  con  $\text{Tr}(A)$  quindi,  $\{t'|t = \alpha.t' \in \text{Tr}(A)\}$ .

Perciò  $\text{Tr}(A') \subseteq \{t'|t = \alpha.t' \in \text{Tr}(A)\} = \text{Tr}(\sum \{B'|B \xrightarrow{\alpha} B'\})$  in accordo con quanto scritto precedentemente.

Si procede con la dimostrazione  $\text{Tr}(P|R) = \text{Tr}(Q|R)$ :

( $\subseteq$ )

Sia  $t \in \text{Tr}(P|R) \Rightarrow t \in (Q|R)$  Per induzione su  $|t|$ :

**Caso Base**  $|t| = 0$

Allora  $t = \varepsilon \in \text{Tr}(P|R) \Rightarrow \varepsilon \in \text{Tr}(Q|R)$

**Caso Induttivo**  $|t| = n + 1$

$t$  ha una prima interazione seguita poi dalla traccia  $t'$ , quindi  $P|R \xrightarrow{\alpha_1} X' \xrightarrow{t'}$  ovvero viene applicata una transizione secondo la regola del parallelo arrivando in un certo processo  $X'$ ; ci sono perciò tre possibilità:

- $P|R$  ha effettuato una transizione usando la regola PAR-L:

$$\frac{P \xrightarrow{\alpha_1} P'}{P|R \xrightarrow{\alpha_1} P'|R \xrightarrow{t'}} \text{ PAR-L}$$

$t = \alpha_1.t'$  con  $|t'| = n$ , per ipotesi induttiva  $t' \in \text{Tr}(P'|R)$ ,  
si applica il lemma:

$$\text{Tr}(P') \subseteq \text{Tr}(\sum \{Q'|Q \xrightarrow{\alpha_1} Q'\}) \text{ e quindi } t' \in \text{Tr}(P'|R) \Rightarrow t' \in \text{Tr}(\sum_{Q \xrightarrow{\alpha_1} Q'} (Q'|R))$$

Di conseguenza  $t \in \alpha_1.\text{Tr}(\sum_{Q \xrightarrow{\alpha_1} Q'} (Q'|R)) = \bigcup_{Q \xrightarrow{\alpha_1} Q'} \text{Tr}(\alpha_1.(Q'|R))$  e quindi, dato che

il processo  $Q|R$  attraverso una transizione  $\alpha_1$  può arrivare al processo  $Q'|R$ , si dimostra che se  $t \in (P|R) \Rightarrow t \in (Q|R)$ .

- $P|R$  ha effettuato una transizione usando la regola PAR-R:

$$\frac{R \xrightarrow{\alpha_1} R'}{P|R \xrightarrow{\alpha_1} P|R' \xrightarrow{t'}} \text{ PAR-R}$$

$t = \alpha_1.t'$  con  $|t'| = n$ , per ipotesi induttiva  $t' \in \text{Tr}(P|R')$ ,  
si applica il lemma:

$$t' \in \text{Tr}(P|R') \Rightarrow t' \in \text{Tr}(Q|R') \text{ e quindi } t \in \text{Tr}(\alpha_1.(P|R')) \Rightarrow t \in \text{Tr}(\alpha_1.(Q|R')).$$

Dato che il processo  $Q|R$  attraverso una transizione  $\alpha_1$  può arrivare al processo

$Q|R'$ , si dimostra che se  $t \in (P|R) \Rightarrow t \in (Q|R)$ .

- $P|R$  ha effettuato una transizione usando la regola PAR- $\tau$ :

$$\frac{P \xrightarrow{\alpha_1} P' \quad R \xrightarrow{\overline{\alpha_1}} R'}{P|R \xrightarrow{\tau} P'|R' \xrightarrow{t'}} \text{ PAR-}\tau$$

$t = \tau_1.t'$  con  $|t'| = n$ , per ipotesi induttiva  $t' \in \text{Tr}(P'|R')$ ,

si applica il lemma:

$$\text{Tr}(P') \subseteq \text{Tr}(\sum \{Q'|Q \xrightarrow{\alpha_1} Q'\}) \text{ e quindi } t' \in \text{Tr}(P'|R') \Rightarrow t' \in \text{Tr}(\sum_{Q \xrightarrow{\alpha_1} Q'} (Q'|R'))$$

Di conseguenza  $t \in \tau_1.\text{Tr}(\sum_{Q \xrightarrow{\alpha_1} Q'} (Q'|R')) = \bigcup_{Q \xrightarrow{\alpha_1} Q'} \text{Tr}(\tau_1.(Q'|R'))$  e quindi, dato che

il processo  $Q|R$  attraverso una transizione  $\tau_1$  effettua la sincronizzazione tra  $Q$  e  $R$  per arrivare al processo  $Q'|R'$ , si dimostra che se  $t \in (P|R) \Rightarrow t \in (Q|R)$ .

( $\supseteq$ )

Sia  $t \in \text{Tr}(Q|R) \Rightarrow t \in (P|R)$  Per induzione su  $|t|$ :

**Caso Base**  $|t| = 0$

Allora  $t = \varepsilon \in \text{Tr}(Q|R) \Rightarrow \varepsilon \in \text{Tr}(P|R)$

**Caso Induttivo**  $|t| = n + 1$

$t$  ha una prima interazione seguita poi dalla traccia  $t'$ , quindi  $Q|R \xrightarrow{\alpha_1} X' \xrightarrow{t'}$  ovvero viene applicata una transizione secondo la regola del parallelo arrivando in un processo stato  $X'$ ; ci sono perciò tre possibilità:

- $Q|R$  ha effettuato una transizione usando la regola PAR-L:

$$\frac{Q \xrightarrow{\alpha_1} Q'}{Q|R \xrightarrow{\alpha_1} Q'|R \xrightarrow{t'}} \text{ PAR-L}$$

$t = \alpha_1.t'$  con  $|t'| = n$ , per ipotesi induttiva  $t' \in \text{Tr}(Q'|R)$ ,

si applica il lemma:

$$\text{Tr}(Q') \subseteq \text{Tr}(\sum \{P'|P \xrightarrow{\alpha_1} P'\}) \text{ e quindi } t' \in \text{Tr}(Q'|R) \Rightarrow t' \in \text{Tr}(\sum_{P \xrightarrow{\alpha_1} P'} (P'|R))$$

Di conseguenza  $t \in \alpha_1.\text{Tr}(\sum_{P \xrightarrow{\alpha_1} P'} (P'|R)) = \bigcup_{P \xrightarrow{\alpha_1} P'} \text{Tr}(\alpha_1.(P'|R))$  e quindi, dato che

il processo  $P|R$  attraverso una transizione  $\alpha_1$  può arrivare al processo  $P'|R$ , si dimostra che se  $t \in (Q|R) \Rightarrow t \in (P|R)$ .

- $P|R$  ha effettuato una transizione usando la regola PAR-R:

$$\frac{R \xrightarrow{\alpha_1} R'}{Q|R \xrightarrow{\alpha_1} Q|R' \xrightarrow{t'}} PAR-R$$

$t = \alpha_1.t'$  con  $|t'| = n$ , per ipotesi induttiva  $t' \in Tr(Q|R')$ ,

si applica il lemma:

$t' \in Tr(Q|R') \Rightarrow t' \in Tr(P|R')$  e quindi  $t \in Tr(\alpha_1.(Q|R')) \Rightarrow t \in Tr(\alpha_1.(P|R'))$ .

Dato che il processo  $P|R$  attraverso una transizione  $\alpha_1$  può arrivare al processo  $P|R'$ , si dimostra che se  $t \in (Q|R) \Rightarrow t \in (P|R)$ .

- $P|R$  ha effettuato una transizione usando la regola  $PAR-\tau$ :

$$\frac{Q \xrightarrow{\alpha_1} Q' \quad R \xrightarrow{\overline{\alpha_1}} R'}{Q|R \xrightarrow{\tau_1} Q'|R' \xrightarrow{t'}} PAR-\tau$$

$t = \tau_1.t'$  con  $|t'| = n$ , per ipotesi induttiva  $t' \in Tr(Q'|R') \Rightarrow t' \in Tr(P'|R')$ ,

si applica il lemma:

$Tr(Q') \subseteq Tr(\sum \{P'|P \xrightarrow{\alpha_1} P'\})$  e quindi  $t' \in Tr(Q'|R') \Rightarrow t' \in Tr(\sum_{P \xrightarrow{\alpha_1} P'} (P'|R'))$

Di conseguenza  $t \in \tau_1.Tr(\sum_{P \xrightarrow{\alpha_1} P'} (P'|R')) = \bigcup_{P \xrightarrow{\alpha_1} P'} Tr(\tau_1.(P'|R'))$  e quindi, dato che

il processo  $P|R$  attraverso una transizione  $\tau_1$  effettua la sincronizzazione tra  $P$  e  $R$  per arrivare al processo  $P'|R'$ , si dimostra che se  $t \in (Q|R) \Rightarrow t \in (P|R)$ .

Quindi con  $Tr(P) = Tr(Q)$ ,  $P|R \sim_t Q|R$  come si voleva dimostrare.

#### 1.2.1.4 Contesto restrizione $C[\ ] = \backslash L$

Il caso del contesto restrizione  $L$  sul processo  $P$  ha la seguente uguaglianza:

$Tr(P \backslash L) = Tr(P) \backslash \{t = \dots \alpha_x \dots | \alpha_x \in L\}$  cioè le tracce che stanno in  $Tr(P)$  non ci sono nell'insieme di restrizione definito precedentemente. Se questo è vero, dato che per ipotesi  $Tr(P) = Tr(Q)$  e quindi  $Tr(Q \backslash L) = Tr(Q) \backslash \{t = \dots \alpha_x \dots | \alpha_x \in L\}$ , allora  $Tr(P \backslash L) = Tr(Q \backslash L)$ .

Perciò si deve dimostrare che il contesto restrizione  $L$  sul processo  $P$  è uguale a

$Tr(P) \backslash \{t = \dots \alpha_x \dots | \alpha_x \in L\}$ . Dimostrato questo ne consegue la veridicità di  $P \backslash L \sim_t Q \backslash L$ .

( $\subseteq$ )

Sia  $t \in Tr(P \backslash L) \Rightarrow t \in Tr(P) \backslash \{t = \dots \alpha_x \dots | \alpha_x \in L\}$  Per induzione su  $|t|$ :

**Caso Base**  $|t| = 0$

Allora  $t = \varepsilon \in Tr(P) \backslash \{t = \dots \alpha_x \dots | \alpha_x \in L\}$

**Caso Induttivo**  $|t| = n + 1$

$t$  ha una prima interazione seguita poi dalla traccia  $t'$ , quindi  $P \backslash L \xrightarrow{\alpha_1} X' \xrightarrow{t'}$  ovvero viene applicata una transizione secondo la regola della restrizione arrivando in un certo processo  $X'$ , quindi:

$$\frac{P \xrightarrow{\alpha_1} P'}{P \setminus L \xrightarrow{\alpha_1} P' \setminus L \xrightarrow{t'}} RES \text{ se } \alpha_1, \overline{\alpha_1} \notin L$$

$t = \alpha_1.t'$  con  $|t'| = n$ , per ipotesi induttiva  $t' \in \text{Tr}(P' \setminus L)$ , quindi  $\alpha_1.t' \in \alpha_1.\text{Tr}(P' \setminus L) \subseteq \text{Tr}(P \setminus L)$ . Dato che  $\alpha_1, \overline{\alpha_1} \notin L$  quindi  $t = \alpha.t' \notin \{t = \dots \alpha_x \dots | \alpha_x \in L\}$  allora  $t \in \text{Tr}(P) \setminus \{t = \dots \alpha_x \dots | \alpha_x \in L\}$ .

( $\supseteq$ )

Sia  $t \in \text{Tr}(P) \setminus \{t = \dots \alpha_x \dots | \alpha_x \in L\} \Rightarrow t \in \text{Tr}(P \setminus L)$ . Per induzione su  $|t|$ :

**Caso Base  $|t| = 0$**

Allora  $t = \varepsilon \in \text{Tr}(P \setminus L)$

**Caso Induttivo  $|t| = n + 1$**

$t$  ha una prima interazione seguita poi dalla traccia  $t'$ , cioè  $t = \alpha_1.t'$ . Si ha quindi una transizione  $P \xrightarrow{\alpha_1} P'$ , ciò è permesso dalla regola della restrizione, quindi:

$$\frac{P \xrightarrow{\alpha_1} P'}{P \setminus L \xrightarrow{\alpha_1} P' \setminus L \xrightarrow{t'}} RES \text{ se } \alpha_1, \overline{\alpha_1} \notin \{t = \dots \alpha_x \dots | \alpha_x \in L\}$$

Questo dimostra che  $P \setminus L$  sa fare l'interazione  $\alpha_1$ , perciò per ipotesi induttiva  $t' \in P' \setminus L$  allora  $\alpha_1.t' \in \text{Tr}(P \setminus L)$ .

Quindi dato che  $\text{Tr}(P \setminus L) = \text{Tr}(P) \setminus \{t = \dots \alpha_x \dots | \alpha_x \in L\}$  con  $\text{Tr}(P) = \text{Tr}(Q)$  allora si è dimostrato che  $P \setminus L \sim_t Q \setminus L$ .

### 1.2.1.5 Contesto relabelling $C[\ ] = [f]$

Nel caso del contesto relabelling sul processo  $P$  si ha che:

Data la funzione  $f : \text{Act} \rightarrow \text{Act}$ , le traccie di  $P[f]$  sono:

$$f(\varepsilon) = \varepsilon$$

$$f(\alpha.t) = f(\alpha).f(t)$$

Quindi voglio dimostrare che  $\text{Tr}(P[f]) = \{f(t) | t \in \text{Tr}(P)\}$ . Se questo è vero, dato che  $\text{Tr}(P) = \text{Tr}(Q)$  si può sostituire  $\text{Tr}(P)$  con  $\text{Tr}(Q)$  scrivendo  $\{f(t) | t \in \text{Tr}(Q)\}$  e grazie alla uguaglianza scritta precedentemente, allora  $\text{Tr}(P[f]) = \text{Tr}(Q[f])$ .

( $\subseteq$ )

Sia  $t \in \text{Tr}(P[f]) \Rightarrow t \in \{f(t) | t \in \text{Tr}(P)\}$ . Per induzione su  $|t|$ :

**Caso Base  $|t| = 0$**

Allora  $t = \varepsilon \in \{f(\varepsilon) | \varepsilon \in \text{Tr}(P)\}$

### Caso Induttivo $|t| = n + 1$

$t$  ha una prima interazione seguita poi dalla traccia  $t'$ , quindi  $P[f] \xrightarrow{\alpha_1} X' \xrightarrow{t'}$  ovvero viene applicata una transizione secondo la regola del relabelling arrivando in un certo processo  $X'$ , quindi:

$$\frac{P \xrightarrow{\alpha_1} P'}{P[f] \xrightarrow{f(\alpha_1)} P'[f] \xrightarrow{f(t')}} REL$$

$t = f(\alpha_1).f(t)'$  con  $|t'| = n$ , per ipotesi induttiva si ha che  $f(t') \in \{f(t') \mid t' \in \text{Tr}(P')\}$ , allora  $f(\alpha).f(t') \in \{f(t) \mid t \in \text{Tr}(P)\}$ .

( $\supseteq$ )

Sia  $t \in \{f(t) \mid t \in \text{Tr}(P)\} \Rightarrow t \in \text{Tr}(P[f])$ . Per induzione su  $|t|$ :

### Caso Base $|t| = 0$

Allora  $t = \varepsilon \in \text{Tr}(P[f])$

### Caso Induttivo $|t| = n + 1$

$t$  ha una prima interazione seguita poi dalla traccia  $t'$ , cioè  $t = \alpha_1.t'$ . Si ha quindi una transizione  $P \xrightarrow{\alpha_1} P'$ , ciò è permesso dalla regola del relabelling, quindi:

$$\frac{P \xrightarrow{\alpha_1} P'}{P[f] \xrightarrow{f(\alpha_1)} P'[f] \xrightarrow{f(t')}} REL$$

Questo dimostra che  $P[f]$  sa fare l'interazione  $\alpha_1$ , perciò per ipotesi induttiva  $t' \in P'[f]$  allora  $\alpha_1.t' \in \text{Tr}(P[f])$ .

Quindi dato che  $\text{Tr}(P[f]) = \{f(t) \mid t \in \text{Tr}(P)\}$  con  $\text{Tr}(P) = \text{Tr}(Q)$  allora si è dimostrato che  $P[f] \sim_t Q[f]$ .

### 1.2.2 Conclusione

Si è dimostrato con i vari casi della dimostrazione precedente, che per ogni possibile contesto che può essere usato, la trace equivalence risulta essere una congruenza per il CCS.



## 2 Barbershop

Il problema del Barbiere formato da 2 tipi di processi, un processo barbiere che effettua tagli di barba/capelli a dei processi clienti, e un insieme di processi clienti che vogliono effettuare un taglio. Il negozio del barbiere prevede l'esistenza di una sala d'attesa con  $n$  sedie, e la stanza del barbiere con una sedia dove viene effettuato il taglio. Se non ci sono clienti che aspettano di essere serviti, il barbiere dorme. Se arriva un cliente nel negozio vi sono tre casi possibili; tutte le sedie sono occupate da altri clienti, e quindi il cliente se ne va dal negozio, oppure il barbiere è occupato e c'è almeno una sedia libera nella sala d'attesa, il cliente può sedersi ed aspettare che il barbiere si liberi ed effettui il taglio, infine se il barbiere sta dormendo, il barbiere si sveglierà e effettuerà il taglio al cliente.

È importante rispettare i seguenti vincoli:

- I processi cliente dovrebbero ricevere il taglio;
- Se un cliente arriva quando il negozio è pieno, se ne va;
- Il barbiere dovrebbe effettuare i tagli;
- Il barbiere serve i clienti uno per volta.

### 2.1 Una possibile soluzione

Il libro Little Book of Semaphores suggerisce la seguente soluzione:

Sia  $n = 2$ , quindi una sedia per la sala d'attesa e una per la barberia.

Si vuole utilizzare un mutex per proteggere la sezione critica al cui interno vi è un contatore di clienti presenti nel negozio. Quindi il contatore per essere modificato si dovrà garantire la mutua esclusione. Quando un cliente entra nel negozio dovrà entrare nella sezione critica per controllare che il contatore sia uguale a  $n$ , se lo è allora esce dal negozio, se invece non lo è incrementa il contatore ed esce dalla sezione critica. Una volta incrementato il cliente segnala attraverso un semaforo Customer la sua presenza al barbiere e si mette in attesa nel semaforo Barber per attendere il servizio del barbiere.

Una volta che il barbiere segnala la presa in carico del cliente sul semaforo Barber, viene effettuato il taglio e il cliente e il barbiere si sincronizzano sui semafori CustomerDone e BarberDone per garantire che il taglio sia stato fatto e che il barbiere possa effettuare un taglio ad un altro cliente in attesa se c'è altrimenti torna a dormire. Il cliente dopo il taglio entra di nuovo nella sezione critica per decrementare il contatore in modo tale da uscire dal negozio.

Il barbiere nello specifico all'inizio rimane in attesa sul semaforo Customer aspettando l'arrivo di un cliente (simula il fatto che stia dormendo), una volta svegliato, segnala sul semaforo Barber di essere pronto con il taglio e prendere in carico un cliente (il primo che sincronizza con il segnale), effettua il taglio si sincronizza con il cliente che ha ricevuto il servizio. Se ci sono altri clienti in attesa passa direttamente al nuovo lavoro da effettuare senza mettersi a dormire, invece se non c'è nessuno torna a dormire e si ferma sul semaforo Customer.

Di seguito viene mostrata una possibile soluzione in pseudo-codice.

```
mutex.wait()
    if customers == n:
        mutex.signal()
        balk()
        customers += 1
mutex.signal()

customer.signal()
barber.wait()

# getHairCut()

customerDone.signal()
barberDone.wait()

mutex.wait()
    customers -= 1
mutex.signal()
```

Figura 1: Definizione processo cliente.

```
customer.wait()
barber.signal()

# cutHair()

customerDone.wait()
barberDone.signal()
```

Figura 2: Definizione processo barbiere.

## 2.2 Modellazione in CCS

Quindi le entità utilizzate nel programma CCS sono:

- $Customer_i$ : Processo cliente che riceve il taglio;
- $Count_i$ : Mutex con al suo interno il contattore del numero di clienti presenti nel negozio;
- Barber: Processo barbiere che effettua il taglio;
- Sys: Sistema.

Di seguito si mostra un esempio del sistema con  $n = 2$  e tre clienti.

```
Count1 = enter.incExit.Count2;
Count2 = enter.(incExit.CountB + decExit.Count1);
CountB = enter.(balk.CountB + decExit.Count2);
```

```
Customer1 = 'enter.('incExit.C1 + 'balk.Customer1);
C1 = semCustomer.'semBarber.semCustomerDone.'semBarberDone.'enter.'decExit.Customer1;
```

```
Customer2 = 'enter.('incExit.C2 + 'balk.Customer2);
C2 = semCustomer.'semBarber.semCustomerDone.'semBarberDone.'enter.'decExit.Customer2;
```

```
Customer3 = 'enter.('incExit.C3 + 'balk.Customer3);
C3 = semCustomer.'semBarber.semCustomerDone.'semBarberDone.'enter.'decExit.Customer3;
```

```
Barber = 'semCustomer.semBarber.'semCustomerDone.cutDone.semBarberDone.Barber;
```

set  $L = \{ \text{enter}, \text{incExit}, \text{decExit}, \text{balk}, \text{semCustomer}, \text{semCustomerDone}, \text{semBarber}, \text{semBarberDone} \};$

$\text{Sys} = (\text{Customer1} | \text{Customer2} | \text{Customer3} | \text{Count1} | \text{Barber}) \setminus L;$

### 2.2.1 Codifica Mutex e contatore clienti

Per codificare il contatore e il mutex si è deciso di utilizzare un unico processo, o meglio un insieme di processi che codificano il mutex e il contatore, perciò abbiamo:

*Count1 = enter.incExit.Count2;*

Codifica il contatore che passa da zero a uno gestendo il tutto in mutua esclusione. Per interagire con il contatore i processi clienti devono sincronizzarsi su *enter* che risulta essere un canale ristretto, ciò permette la mutua esclusione che sarà dimostrata in seguito. Per poter incrementare il contatore e uscire dalla sezione critica i processi cliente utilizzeranno il canale *incExit* anch'esso ristretto, mentre il processo *Count1* andrà nel processo *Count2* per mantenere il contatore a uno e per dare la possibilità di decrementare o incrementare un successivo processo cliente.

*Count2 = enter.(incExit.CountB + decExit.Count1);*

Codifica il contatore che passa da uno a due e/o da uno a zero, gestendo il tutto in mutua esclusione. Analogamente a *Count1* c'è *enter* per la sincronizzazione in mutua esclusione. Viene data una scelta in più su come proseguire l'esecuzione. A seconda di cosa richiede il processo cliente può essere data la possibilità di incrementare il contatore e uscire, sempre attraverso il canale *incExit*, oppure il decremento attraverso *decExit* (canale ristretto). La scelta dipende dello stato d'esecuzione in cui si trova il processo cliente. Nel caso incrementi, *Count2* va in *CountB* altrimenti torna in *Count1*.

*CountB = enter.(balk.CountB + decExit.Count2);*

Codifica il contatore che non può essere più incrementato perché uguale a *n*, e l'azione di decremento. Il funzionamento è uguale a *Count2* tranne per il fatto che *incExit* non esiste ma c'è *balk* (canale ristretto) usato per simulare l'uscita dal locale del cliente nel caso in cui voglia ricevere un taglio ma non ci sono sedie disponibili.

### 2.2.2 Codifica Customer

La codifica dei clienti avviene nel seguente modo:

*Customer1 = 'enter.('incExit.C1 + 'balk.Customer1);*

*C1 = semCustomer.'semBarber.semCustomerDone.'semBarberDone.'enter.'decExit.Customer1;*

Il cliente per poter controllare se può entrare si sincronizza sul canale *enter* rimando in attesa della possibilità di entrare nella sezione critica. Una volta entrato controlla se può incrementare oppure no il contatore, quindi sceglie se eseguire *incExit* o *balk*, tale scelta dipende da cosa offre il processo *count*. Nel caso sia *CountB*, l'unica azione possibile per il cliente è eseguire *balk* che lo fa uscire dalla sezione critica e non lo fa entrare nel negozio. Nel caso invece sia *Count1* o *Count2* il cliente esegue *incExit* così che incrementi il contatore e esca dalla sezione critica. Successivamente si sincronizza con il barbiere, che nel caso in cui il barbiere sia libero passa a ricevere il taglio. Finito il taglio si sincronizza subito con il barbiere per poi entrare nella sezione critica in mutua esclusione e decrementare il contatore per simulare la sua uscita dal negozio.

### 2.2.3 Codifica Barber

La codifica del Barbiere avviene nel seguente modo:

```
Barber = 'semCustomer.semBarber.'semCustomerDone.cutDone.semBarberDone.Barber;
```

Il barbiere rimane in attesa di un nuovo cliente da servire su `semCustomer` se non ci sono cliente in attesa. Una volta arrivato un cliente si sincronizza con esso ed esegue il taglio, successivamente si sincronizza con il cliente appena servito per garantire che sia avvenuto il taglio per poi tornare all'inizio della sua esecuzione per poter eseguire un nuovo cliente se c'è, altrimenti attende un nuovo cliente.

### 2.2.4 Codifica del sistema

L'intero sistema viene codificato nel seguente modo:

```
Sys = (Customer1|Customer2|Customer3|Count1|Barber) \L;  
set L = { enter, incExit, decExit, balk, semCustomer, semCustomerDone, semBarber,  
semBarberDone};
```

Il sistema viene rappresentato attraverso una composizione parallela dei processi Customer, Barber e Count1. Tutti i canali vengono ristretti per permettere la sincronizzazione dei processi, escluso `cutDone` usato per mostrare un comportamento all'esterno.

## 2.3 Verifica della correttezza attraverso CWB

Di seguito si verificano se il programma definito rispetta tutte le caratteristiche stabilite dalla definizione del problema, attraverso l'uso della Edinburgh Concurrency Workbench (CWB).

### 2.3.1 Weak Bisimulation and Trace Equivalence

```
Spec = cutDone.Spec;
```

### 2.3.2 Assenza di deadlock

### 2.3.3 Mutua esclusione contattore

### 2.3.4 Mutua esclusione nell'esecuzione del taglio

### 2.3.5 Verifica comportamento del cliente che entra quando il negozio pieno

### 2.3.6 Verifica comportamento del barbiere nell'attesa dell'arrivo di un nuovo cliente

### 2.3.7 Verifica comportamento del cliente nell'attesa di essere servito

### 2.3.8 Fairness