Automatic System Verification
Exercices

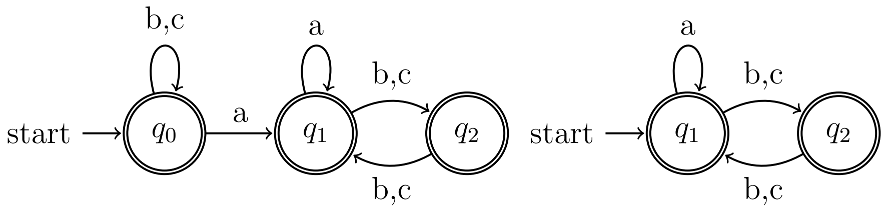
Cominato Enrico 137396
Department of Computer Science
University of Udine

March 9, 2021

1 Exercices on the automata's notes

Esercizio 2.3

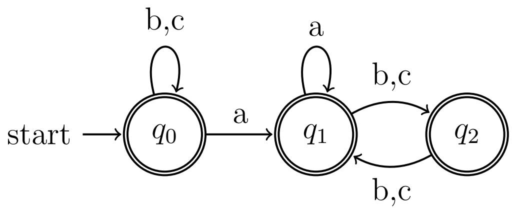
Sia $\mathcal{A}$ l'automa dell'Esempio 2.2. Si consideri l'automa $\mathcal{A}'$ ottenuto da $\mathcal{A}$ rimuovendo lo stato q_0, e le transizioni in esso entranti e da esso uscenti, e facendo diventare q_1 il nuovo stato iniziale. Si stabilisca se $\mathcal{A}$ e $\mathcal{A}'$ riconoscono o meno lo stesso linguaggio
Riporto di seguito i due grafi.



I due linguaggi non sono uguali. Per esempio la $\omega-parola$ *babcabca...* appartiene al primo dei due automi, ma non al secondo (da q_1 andiamo in q_2 ma da li possiamo leggere solo una b oppure una c)

Esercizio 2.4

Si costruisca l'automa $\mathcal{A}'$ che riconosce la variante finita (linguaggio di parole finite) dell'Esempio 2.2
Il linguaggio richiesto è il seguente:
L'insieme delle parole finite su $A = \{a, b, c\}$ tali che tra ogni coppia di occorrenze consecutive di a esiste un numero pari di occorrenze di simboli diversi da a.
Osservazione: una parola con una sola occorrenza di a deve essere sempre accettata.
L'automa risultante quindi è lo stesso dell'esempio 2.2, solo che le run su questo automa sono finite.



Esercizio 2.5

Sia W il linguaggio riconosciuto dall'automa $\mathcal{A}$' dell'Esercizio 2.4. Si caratterizzi il linguaggio $\overrightarrow{W}$.
Riprendo la definizione di $\overrightarrow{W}$:

$$\overrightarrow{W} = \{\alpha \in A^\omega \ t.c. \ \exists^\omega n \ \alpha(0, n) \in W\}$$

Sono quindi tutte quelle ω-parole di cui ogni prefisso finito appartiene a W. Analizziamo per casi:

- la parola non ha neppure una a: in questo caso ogni suo prefisso appartiene a W perchè le parole $(b|c), (b|c)^2, (b|c)^3, \dots, (b|c)^n, \dots \in W$
- la parola ha una sola occorrenza di a: anche in questo caso ogni suo prefisso appartiene a W perchè:
  - come visto nel punto precedente, fino a che non si incontra la lettera a le parole appartengono a W
  - dopo l'occorrenza di a, ancora tutti i prefissi appartengono a W, dato che tutte le parole finite con una sola occorrenza di a appartengono a W
- infine tutti gli altri casi sono parole con un più di una occorrenza di a: dove abbiamo che ogni prefisso, o ricade in uno dei precedenti casi, oppure è una parola che tra ogni coppia di occorrenze consecutive di a esiste un numero pari di occorrenze di simboli diversi da a e che quindi appartiene a W.

In questo caso abbiamo che $W^\omega = \overrightarrow{W}$.

Esercizio 2.7

Teorema 2.6

1. Se $V \subseteq A^*$ è regolare, allora V^ω è $\omega - regolare$
2. Se $V \subseteq A^*$ è regolare e $L \subseteq A^\omega$ è $\omega - regolare$, allora $V \cdot L$ è $\omega - regolare$
3. Se $L_1, L_2 \subseteq A^*$ sono $\omega - regolari$, allora $L_1 \cup L_2$ e $L_1 \cap L_2$sono $\omega - regolari$

Dimostrare le proprietà (2) e (3) del teorema
Dimostro *(2)*. Siano $\mathcal{A}, \mathcal{B}$ automi tali che $\mathcal{A}$ accetta V e $\mathcal{B}$ accetta L, allora possiamo costruire $\mathcal{C}$ unendo tutti gli stati finali di $\mathcal{A}$ con lo stato iniziale di $\mathcal{B}$. Otteniamo così un'automa che legge il linguaggio $V \cdot L$, quindi $V \cdot L$ è ω-regolare.
Dimostro *(3)*. Per quanto riguarda l'unione, prendiamo $\mathcal{A}, \mathcal{B}$ automi tali che $\mathcal{A}$ accetta L_1 e $\mathcal{B}$ accetta L_2. Costruiamo $\mathcal{C}$ unendo gli stati iniziali di $\mathcal{A}$ e $\mathcal{B}$. Abbiamo quindi che $\mathcal{C}$ accetta tutte le parole di $L_1 \cup L_2$. Sappiamo inoltre che i linguaggi ω-regolari sono chiusi per complementazione, quindi possiamo riscrivere $L_1 \cap L_2 = \neg(\neg L_1 \cup \neg L_2)$ ed ottenere la chiusura per intersezione.

Esercizio 2.13

Fornire un esempio di parola non definitivamente periodica
Un esempio è:

$$ababbab^3ab^4ab^5...$$

Esercizio 2.16

Dimostrare che una congruenza è una relazione di equivalenza invariante destra
Invariante a destra significa che $\forall x, y, z \in A$, se $x \sim y$ allora $xz \sim yz$. Questo è sempre vero perchè, concatenando la stessa parola ad x, y finiremo in un'unica classe di equivalenza.

Esercizio 2.19

Dato un automa di Büchi $\mathcal{A} = (\mathcal{Q}, A, \Delta, q_0, F)$, dimostrare che, per ogni $s, s' \in \mathcal{Q}$, $W^F_{ss'}$ è regolare
Un linguaggio è regolare se esiste un'automa in grado di accettarlo. Per poterlo accettare deve avere almeno uno stato finale. Quindi se eliminiamo dall'automa $\mathcal{A}$, tutti gli stati e le relazioni non interessate dai possibili cammini tra s e s' otteniamo un automa in grado di leggere solo $W^F_{ss'}$, e questo fa di lui un linguaggio regolare.

Esercizio 2.23

Dimostrare che la relazione $\approx_A$ è una congruenza di indice finito
Perchè $\approx_A$ sia una congruenza deve valere che $\forall u, u', v, v' \in A^*$ se $u \approx_A v$ e $u' \approx_A v'$ allora $uu' \approx_A vv'$. Questo è vero perchè avendo $u \approx_A v$ e $u' \approx_A v'$, allora $\exists t$ tale che:

- $s \rightarrow_u t \Leftrightarrow s \rightarrow_v t$
- $s \rightarrow^F_u t \Leftrightarrow s \rightarrow^F_v t$
- $t \rightarrow_{u'} s' \Leftrightarrow t \rightarrow_{v'} s'$
- $t \rightarrow^F_{u'} s' \Leftrightarrow t \rightarrow^F_{v'} s'$

Quindi abbiamo che $\forall s, s' \in Q$:

- $s \rightarrow_{uu'} s' \Leftrightarrow s \rightarrow_{vv'} s'$
- $s \rightarrow^F_{uu'} s' \Leftrightarrow s \rightarrow^F_{vv'} s'$

e quindi $uu' \approx_A vv'$. Il resto della dimostrazione è già stata svolta negli appunti.

Esercizio 2.25

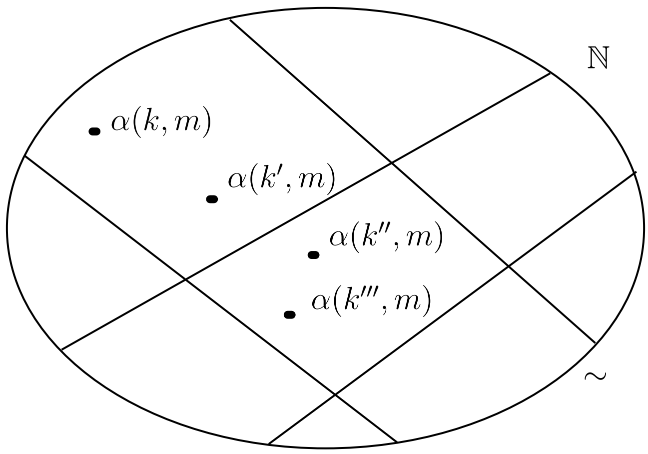
Si dimostri che la relazione $\cong_\alpha$ è una relazione di equivalenza sui naturali di indice finito
Riprendo la definizione di $\cong_\alpha$. Sia $\sim$ una congruenza su A^* di indice finito. Sia $\alpha \in A^\omega$ e siano k, k' posizioni. Diciamo che $k \cong^m_\alpha k'$ (k, k' si riuniscono in $m > k, k'$) se $\alpha(k, m) \sim \alpha(k', m)$. Diciamo che $k \cong_\alpha k'$ se esiste m per cui $k \cong^m_\alpha k'$.
Dimostro che è una relazione di equivalenza:

- **Riflessività**: $\forall k \in \mathbb{N}$ è sempre vero che $k \cong_\alpha k$ perchè $k \cong_\alpha k \Leftrightarrow \exists m\ t.c.\ \alpha(k,m) \sim \alpha(k,m)$ e questo è vero $\forall k$ perchè $\sim$ è una relazione di equivalenza

- **Simmetria**: $\forall k, k' \in \mathbb{N}$ è sempre vero che $k \cong_\alpha k' \Rightarrow k' \cong_\alpha k$ perchè $k \cong_\alpha k' \Leftrightarrow \exists m\ t.c.\ \alpha(k,m) \sim \alpha(k',m)$. Dato che $\sim$ è una relazione di equivalenza allora vale che $\exists m\ t.c.\ \alpha(k',m) \sim \alpha(k,m)$, il che significa che $k' \cong_\alpha k$.

- **Transitività**: $\forall i, j, k \in \mathbb{N}$ è sempre vero che $i \cong_\alpha j$ e $j \cong_\alpha k \Rightarrow i \cong_\alpha k$, perchè $i \cong_\alpha j \Leftrightarrow \exists m\ t.c.\ \alpha(i,m) \sim \alpha(j,m)$ e $j \cong_\alpha k \Leftrightarrow \exists n\ t.c.\ \alpha(j,n) \sim \alpha(k,n)$. Senza perdere di generalità pongo $n > m$, quindi abbiamo che $\exists m\ t.c.\ \alpha(i,m) \sim \alpha(j,m)$ e $\alpha(j,m) \sim \alpha(k,m)$. Dato che $\sim$ è una relazione di equivalenza allora vale che $\exists m$ tale che $\alpha(i,m) \sim \alpha(j,m)$ e $\alpha(j,m) \sim \alpha(k,m) \Rightarrow \alpha(i,m) \sim \alpha(k,m)$, il che significa che $i \cong_\alpha k$.

Dimostro che $\cong_\alpha$ ha indice finito. Dato che $\sim$ ha indice finito, per un m fisso, ci troviamo in una situazione del genere:



Dove il numero di classi di equivalenza è limitato dal numero di classi di equivalenza di $\sim$, e sappiamo che $\sim$ ha un numero finito di classi di equivalenza, quindi anche $\cong_\alpha$ avrà un numero finito di classi di equivalenza.

Esercizio 2.44

Dimostrare la chiusura della classe dei linguaggi riconosciuti dagli automi di Büchi deterministici rispetto alle operazioni di unione e intersezione

Siano $\mathcal{A} = (\mathcal{Q}_\mathcal{A}, A, \Delta_A, q_{0A}, F_A)$ e $\mathcal{B} = (\mathcal{Q}_\mathcal{B}, A, \Delta_B, q_{0B}, F_B)$

Unione: Se assumiamo che $\mathcal{Q}_\mathcal{A} \cap \mathcal{Q}_\mathcal{B} = \emptyset$ allora possiamo costruire l'automa unione $\mathcal{C}$ come segue:

- $\mathcal{Q}_\mathcal{C} = \mathcal{Q}_\mathcal{A} \cup \mathcal{Q}_\mathcal{B} \cup \{q_{0C}\}$

- A rimane invariato

- $\Delta_C = \Delta_A \cup \Delta_B$

- q_{0C} come nuovo stato iniziale, con le stesse relazioni di q_{0A} e q_{0B}, finale nel caso che almeno uno tra q_{0A} e q_{0B} sia uno stato finale

- $F_C = F_A \cup F_B$

Intersezione: Costruiamo l'automa intersezione $\mathcal{C}$, partendo dal prodotto cartesiano degli stati:

- $\mathcal{Q}_\mathcal{C} = \mathcal{Q}_\mathcal{A} \times \mathcal{Q}_\mathcal{B} \times \{1, 2\}$
- A rimane invariato
- $\Delta_C = \Delta_1 \cup \Delta_2$ dove
  - $\Delta_1 = \{((q_A, q_B, 1), a, (q'_A, q'_B, i)) \mid (q_A, a, q'_A) \in \Delta_A\ e\ (q_B, a, q'_B) \in \Delta_B\ e\ se\ q_A \in F_A\ allora\ i = 2\ altrimenti\ i = 1\}$
  - $\Delta_2 = \{((q_A, q_B, 2), a, (q'_A, q'_B, i)) \mid (q_A, a, q'_A) \in \Delta_A\ e\ (q_B, a, q'_B) \in \Delta_B\ e\ se\ q_B \in F_B\ allora\ i = 1\ altrimenti\ i = 2\}$
- $q_{0C} = (q_{0A}, q_{0B}, 1)$
- $F_C = \{(q_a, q_b, 2) \mid q_B \in F_B\}$

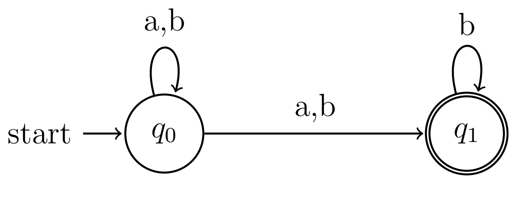
Per costruzione, $r_C = (q^0_A, q^0_B, i^0), (q^1_A, q^1_B, i^1), \ldots$ è un'esecuzione su $\mathcal{C}$ per la parola w se:

- $r_A = q^0_A, q^1_A, \ldots$ è un'esecuzione su $\mathcal{A}$ per w
- $r_B = q^0_B, q^1_B, \ldots$ è un'esecuzione su $\mathcal{B}$ per w

r_A e r_B sono accettate se r_C è la concatenazione di una serie infinita di segmenti finiti di stati 1 (stati con terza componente 1) e stati 2 (stati con terza componente 2) alternativamente. Questa sequenza esiste se r_C è accettato da $\mathcal{A}$

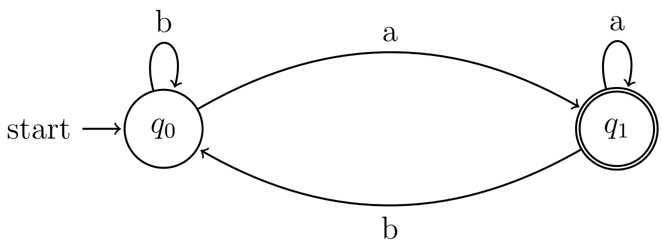
Esercizio 2.46

Sia $A = \{a, b\}$ e $L = \{\alpha \in A^\omega . \exists^{<\omega}\ n\ \alpha(n) = a\}$. Si costruisca un automa di Büchi non deterministico che riconosca il linguaggio L



Esercizio 2.48

Sia $A = \{a, b\}$ e $L = \overrightarrow{\{b^* a^*\}}$. Si costruisca un automa di Büchi deterministico che riconosca il linguaggio L



Esercizio 2.50

Dimostrare che la classe degli ω-linguaggi ω-regolari coincide con la classe degli ω-linguaggi riconosciuti dagli automi di Muller non deterministici

Dato che un ω-linguaggio per essere ω-regolare deve essere accettato da un automa di Büchi , mi basta dimostrare l'equivalenza tra gli automi di Büchi non deterministici e quelli di Muller non deterministici.

Sia $\mathcal{A} = (\mathcal{Q}, A, \Delta, q_0, F)$ un automa di Büchi , possiamo costruire un automa di Muller: $\mathcal{M} = (\mathcal{Q}, A, \Delta, q_0, \mathcal{F})$ dove $\mathcal{F} = \{X | X \in 2^Q \wedge X \cap F \neq \emptyset\}$. Si può osservare che una ω-parola viene accettata da $\mathcal{A}$ se e solo se passa infinite volte per uno stato finale. Quindi viene accettata anche da $\mathcal{M}$ perchè, presa una sua computazione σ, abbiamo che $In(\sigma) \cap F \neq \emptyset$ e $In(\sigma) \in 2^Q \Rightarrow In(\sigma) \in \mathcal{F}$ quindi la stessa ω-parola viene accettata da $\mathcal{F}$. Lo stesso ragionamento lo possiamo fare al contrario. Se una ω-parola viene accettata da $\mathcal{M}$ allora $In(\sigma) \cap F \neq \emptyset$ quindi esiste una computazione che passa infinite volte per uno stato finale, quindi la stessa ω-parola viene accettata da $\mathcal{A}$.

Esercizio 2.57

Dimostrare che l'insieme $W_V \subseteq A^*$ dei V-testimoni, con V classe di congruenza $\approx_{\mathcal{A}}$ è regolare

Riprendo la definizione di V-Testimone: $\alpha(k_0, m)$ è un V-testimone se, per qualche k, con $k_0 < k < m$, m è la più piccola posizione tale che $\alpha(k_0, k) \in$ V e $k_0 \cong_\alpha^m k$. Chiamiamo W_V insieme dei V-testimoni della classe V.

Possiamo riscrivere questa definizione nel seguente modo:

$$W_V : \left\{ \alpha(k_0, m) \mid \begin{array}{l} \exists k_0, k, m \ t.c. \ k_0 < k < m \\ \alpha(k_0, k) \in V \\ k_0 \cong_\alpha^m k \\ \nexists m' < m \ t.c. \ k_0 \cong_\alpha^{m'} k \end{array} \right\}$$

Una parola x è un V-testimone se esistono $v \in V, u \in U (U \approx_{\mathcal{A}}$-classe) tali che $x = v \cdot u$. Inoltre sappiamo che m è minimo, quindi u a sua volta dovrà essere minimo.

Dato $U \approx_{\mathcal{A}}$-classe definiamo:

$$U_{\min} = \{u \in U \text{ tali che } \exists v \in V \ v \cdot u \in U \wedge v \cdot u \cdot \Sigma+ \notin U\}$$

In conclusione abbiamo che:

$$W_V = \bigcup_{U \approx_{\mathcal{A}}\text{-classe}} (U \cap (V \cdot U_{\min}))$$

2 Exercices of chapter 0 of Temporal Verification of Reactive Systems

Problem 0.1

$$\begin{array}{l} \quad\quad \textbf{out } \mathrm{x}\text{: } \textbf{integer where } \mathrm{x} = 0 \\ l_0 : \begin{bmatrix} [l_1 : \textbf{while } x \geq 0 \textbf{ do } l_2 : x := x + 1] \\ \quad \textbf{or} \\ [l_3 : \textbf{await } x > 0] \end{bmatrix} \\ l_4 : \end{array}$$

Program S8 (strange behavior).

a) *Identify the locations of this program as equivalence classes of labels. List the post-location of each of the statements.*
There are three classes:

$$l_0 = \{l_0, l_1, l_3\}$$
$$l_2 = \{l_2\}$$
$$l_4 = \{l_4\}$$

While the post-locations are:

$$post(l_0) = post(l_1) = post(l_3) = [l_4]$$
$$post(l_2) = [l_0]$$

b) *Show that this program has a terminating computation.*
The program can terminate because the post-location of the body of the while is the selection statement. So the await condition can be satisfied and this terminate the program.

c) *Define transitions and transition relations for this version of a WHILE statement. Show that the version of program SB in which the while statement has been replaced by this WHILE statement has no terminating computation.*
Recall of the new WHILE statement:

$$l_1 : [WHILE\ c\ DO\ [l_2 : S; \hat{l_2}]]; l_3 : \text{ where } \hat{l_2} \nsim l_1$$

We can define its transitions $\tau_{l_1}, \tau_{\hat{l_2}}$ and transition relations $\rho_{l_1}, \rho_{\hat{l_2}}$ as follow:

$\rho_{l_1} : \rho^T_{l_1} \vee \rho^F_{l_1}$ where

$$\rho^T_{l_1} : move(l_1, l_2) \wedge c \wedge pres(Y)$$
$$\rho^F_{l_1} : move(l_1, l_3) \wedge \neg c \wedge pres(Y)$$

$\rho_{\hat{l_2}} : \rho^T_{\hat{l_2}} \vee \rho^F_{\hat{l_2}}$ where

$$\rho^T_{\hat{l_2}} : move(\hat{l_2}, l_2) \wedge c \wedge pres(Y)$$
$$\rho^F_{\hat{l_2}} : move(\hat{l_2}, l_3) \wedge \neg c \wedge pres(Y)$$

As we can see, now the program cannot terminate anymore, because once in the WHILE loop, the transition $\rho^F_{\hat{l_2}}$ will be never satisfied.

Problem 0.2

$$\begin{array}{c} \textbf{out } \mathrm{y}\text{: } \textbf{integer where } \mathrm{y} = 0 \\ \textbf{local } \mathrm{x}\text{: } \textbf{boolean where } \mathrm{x} = \mathrm{T} \\ \\ P_1 :: \begin{bmatrix} l_0 : \textbf{while } x \textbf{ do} \\ \quad l_1 : y := y + 1 \\ l_2 : \end{bmatrix} \quad \| \quad P_2 :: \begin{bmatrix} m_0 : x := F \\ m_1 : \end{bmatrix} \end{array}$$

Program ANY-NAT (computing any natural number).

a) *Consider program ANY-NAT. Argue (informally) that this program always terminates. Also show that, for every natural number $n \geq 0$, there exists a computation of ANY-NAT such that $y = n$ on termination.*

The program terminates whenever m_0 is executed. The transition τ_{m_0} is always enabled and because of *justice* requirement we know that, soon or later, it will be taken. Once executed $\rho_{l_0}^F$ is satisfied and control goes to $\pi = \{l_2, m_1\}$. Furthermore, the justice requirement allow only a finite (but unbounded) number of transition τ_{l_0}. So, $\forall n \in \mathbb{N}$ there exists a run where τ_{l_0} is taken n times before τ_{m_0}, leading to have $y = n$ on termination.

b) *Construct a program with a single process that exhibits a similar behavior; that is, all of its computations terminate and, for each natural number n, there exists a computation producing n.*

$$\begin{array}{l} \quad\quad\quad\quad \textbf{out } \mathrm{y}\text{: } \textbf{integer where } \mathrm{y} = 0 \\ \quad\quad\quad\quad \textbf{local } \mathrm{x}\text{: } \textbf{boolean where } \mathrm{x} = \mathrm{T} \\ \\ l_0 : \begin{bmatrix} \textbf{while } x \textbf{ do } l_1 : \begin{bmatrix} [l_2 : y := y + 1] \\ \quad \textbf{or} \\ [l_3 : x := F] \end{bmatrix} \end{bmatrix} \\ l_4 : \end{array}$$

This Program exibits the same behavior of the previus one:

- All of its computation terminates, because τ_{l_3} is always enabled and thank to the justice requirement we know that soon or later it will be taken and change the value of x ending the cycle.
- As the previous program, the number of iteration before the execution of τ_{l_3} is finite but unbounded, so y could have any value in $\mathcal{N}$ at the end of the computation.

c) *Prove that no single-process program with only just transitions can have the same behavior as ANY-NAT.* This kind of program need:

- an incremental statement
- a termination statement

Problem 0.3

a) *Consider the two statements*

$$S_1 :: [x := 1; l_1 : x := 2]$$

$$S_2 :: [x := 1; l_2 : x := x + 1]$$

and the context

$$P[S] :: l_0 : [\boldsymbol{out}\, x : \boldsymbol{integer\ where}\, x = 0; S]\hat{l_0} :$$

Show that $P[S_1]$ and $P[S_2]$ are termination equivalent.
$P[S_1]$ and $P[S_2]$ have the initial state with the same interpretation of the variable x (equal to 0). They both have a single possible run, that is:

$$\begin{aligned} P[S_1] &: \langle \pi : [l_0], x : 0\rangle \xrightarrow{l_0} \langle \pi : [l_1], x : 1\rangle \xrightarrow{l_1} \langle \pi : [\hat{l_0}], x : 2\rangle \\ P[S_2] &: \langle \pi : [l_0], x : 0\rangle \xrightarrow{l_0} \langle \pi : [l_2], x : 1\rangle \xrightarrow{l_2} \langle \pi : [\hat{l_0}], x : 2\rangle \end{aligned}$$

This lead to have ending states sharing the interpretation of variable x equal to 2. Thus $P[S_1]$ and $P[S_2]$ are termination equivalent.

b) *Consider the preceding two statements and the context*

$$Q[S] :: [\boldsymbol{out}\, x : \boldsymbol{integer\ where}\, x = 0; [[l_0 : S; \hat{l_0} :] \parallel [m_0 : x := 0; \hat{m_0} :]]].$$

Show that $Q[S_1]$ and $Q[S_2]$ are not termination equivalent. We may conclude that S_1 and S_2 are not congruent.
$Q[S_1]$ and $Q[S_2]$ are not termination equivalent, because the are some computation that lead to different ending states, like:

$$\begin{aligned} Q[S_1] :&\langle \pi : \{l_0, m_0\}, x : 0\rangle \xrightarrow{l_0} \langle \pi : \{l_1, m_0\}, x : 1\rangle \xrightarrow{m_0} \\ &\langle \pi : \{l_1, \hat{m_0}\}, x : 0\rangle \xrightarrow{l_1} \langle \pi : \{\hat{l_0}, \hat{m_0}\}, x : 2\rangle \\ Q[S_2] :&\langle \pi : \{l_0, m_0\}, x : 0\rangle \xrightarrow{l_0} \langle \pi : \{l_2, m_0\}, x : 1\rangle \xrightarrow{m_0} \\ &\langle \pi : \{l_2, \hat{m_0}\}, x : 0\rangle \xrightarrow{l_2} \langle \pi : \{\hat{l_0}, \hat{m_0}\}, x : 1\rangle \end{aligned}$$

c) *Show the following congruence:*

1. $P :: [S_1 \mathbf{or} S_2] \approx Q :: [S_2 \mathbf{or} S_1]$: If P selects S_1 (resp. S_2), also Q can select S_1 (resp. S_2) and vice versa.

2. $[S_1 \parallel S_2] \approx [S_2 \parallel S_1]$: As the previous case, since there is no precondition to the parallel composition of S_1 and S_2, each transition on P can be choose on Q.

3. $S \approx [S; \mathbf{skip}]$: We know that **skip** always terminates, so S;**skip** terminates if and only if S does. Moreover, **skip** does not alter the data (its data transformation is the identity).

4. **await** $c \approx$ **while** $\neg c$ **do skip** While c is $FALSE$ the programs loop forever. When c is $TRUE$, both programs terminates without altering the values of output variables.

d) *Are the two statements:*

$$\textbf{\textit{await}}\ x \ \textit{and}\ \textbf{\textit{skip}}\ m : \textbf{\textit{await}}\ x$$

congruent? Prove your answer.
The statements are not congruent since there exist a context $P[S]$ like:

$$P[S] :: \begin{bmatrix} \text{local } x : \text{ boolean where } x = F \\ [S \text{ or await } \neg x] \end{bmatrix}$$

where $P[\textbf{await}\ x]$ always terminates because **await** x is not enabled and **await** $\neg x$ is selected, terminating the computation. While $P[\textbf{skip}\ m : \textbf{await}\ x]$ is always enabled, due to the **skip**, and if it is chosen, it leads to a non terminating computation.

e) *Let y be a boolean variable. Which of the three following statements are congruent?*

1. $y := T$
2. $\textbf{\textit{if}}\, y\ \textbf{\textit{then}}\, y := T\ \textbf{\textit{else}}\, y := T$
3. $[[\textbf{\textit{await}}\, y;\ y := T]\ \textbf{\textit{or}}\ [\textbf{\textit{await}}\, \neg y;\ y := T]]$

While *1* and *2* are equivalently enabled, the selection *3* can be disregarded by a scheduler ensuring justice requirement, since the two await commands may be not continuously enabled. Consider the following context:

$$P[S] :: \begin{bmatrix} \text{local } x : \text{ boolean where } x = T \\ \text{local } y : \text{ boolean where } y = F \\ [S; x := F] \parallel [\text{while } x \text{ do } y := \neg y] \end{bmatrix}$$

While *1* and *2* always terminates in computations that ensure justice, with *3* there exists a fair computation that stays forever in loop while x do $y := \neg y$, as the **await** commands are enabled in alternation.

Problem 0.4

$$\textbf{local } y_1, y_2\textbf{: integer where } y_1 = 0,\ y_2 = 0$$

$$P_1 :: \begin{bmatrix} l_0 : \textbf{loop forever do} \\ \quad \begin{bmatrix} l_1 : \textbf{noncritical} \\ l_2 : \textbf{await } y_2 = 0 \\ l_3 : y_1 := 1 \\ l_4 : \textbf{critical} \\ l_5 : y_1 := 0 \end{bmatrix} \end{bmatrix} \quad \parallel \quad P_2 :: \begin{bmatrix} m_0 : \textbf{loop forever do} \\ \quad \begin{bmatrix} m_1 : \textbf{noncritical} \\ m_2 : \textbf{await } y_1 = 0 \\ m_3 : y_2 := 1 \\ m_4 : \textbf{critical} \\ m_5 : y_2 := 0 \end{bmatrix} \end{bmatrix}$$

Program TRY-MUX1(proposed solution to mutual exclusion).

a) *Is program TRY-MUXI a good solution to the mutual-exclusion problem?* No, because it does not satisfy neither the exclusion requirement, neither the accessibility requirement. First of all there exists a computation where a state $\pi = \{l_4, m_4\}$ is accessible as shown by the following computation:

$$\begin{aligned}
\langle \pi : \{l_0, m_0\}, y_1 : 0, y_2 : 0\rangle &\xrightarrow{l_0} \langle \pi : \{l_1, m_0\}, y_1 : 0, y_2 : 0\rangle \\
&\xrightarrow{l_1} \langle \pi : \{l_2, m_0\}, y_1 : 0, y_2 : 0\rangle \\
&\xrightarrow{l_2} \langle \pi : \{l_3, m_0\}, y_1 : 0, y_2 : 0\rangle \\
&\xrightarrow{m_0} \langle \pi : \{l_3, m_1\}, y_1 : 0, y_2 : 0\rangle \\
&\xrightarrow{m_1} \langle \pi : \{l_3, m_2\}, y_1 : 0, y_2 : 0\rangle \\
&\xrightarrow{m_2} \langle \pi : \{l_3, m_3\}, y_1 : 0, y_2 : 0\rangle \\
&\xrightarrow{l_3} \langle \pi : \{l_4, m_3\}, y_1 : 1, y_2 : 0\rangle \\
&\xrightarrow{m_3} \langle \pi : \{l_4, m_4\}, y_1 : 1, y_2 : 1\rangle
\end{aligned}$$

Furthermore P_2 can indefinitely wain in m_2: in fact, P_1 is allowed to visit locations l_3 and l_5, disabling and enabling respectively the statement in m_2. Without a compassion requirement, it has not to be taken eventually.

b) *The same questions for TRY-MUX2, a version of program TRY-MUXl in which statements l_2 and l_3 are interchanged and so are statements m_2 and m_3.*
In this case, program TRY-MUX2 guarantees exclusion but not accessibility since there is a deadlock computation:

$$\begin{aligned}
\langle \pi : \{l_0, m_0\}, y_1 : 0, y_2 : 0\rangle &\xrightarrow{l_0} \langle \pi : \{l_1, m_0\}, y_1 : 0, y_2 : 0\rangle \\
&\xrightarrow{l_1} \langle \pi : \{l_2, m_0\}, y_1 : 0, y_2 : 0\rangle \\
&\xrightarrow{l_2} \langle \pi : \{l_3, m_0\}, y_1 : 1, y_2 : 0\rangle \\
&\xrightarrow{m_0} \langle \pi : \{l_3, m_1\}, y_1 : 1, y_2 : 0\rangle \\
&\xrightarrow{m_1} \langle \pi : \{l_3, m_2\}, y_1 : 1, y_2 : 0\rangle \\
&\xrightarrow{m_2} \langle \pi : \{l_3, m_3\}, y_1 : 1, y_2 : 1\rangle \not\rightarrow
\end{aligned}$$

local t: **integer where** $t = 1$

$$P_1 :: \left[\begin{array}{l} l_0 : \textbf{loop forever do} \\ \quad \left[\begin{array}{l} l_1 : \textbf{noncritical} \\ l_2 : \textbf{await } t = 1 \\ l_3 : \textbf{critical} \\ l_4 : t := 2 \end{array}\right] \end{array}\right] \quad \| \quad P_2 :: \left[\begin{array}{l} m_0 : \textbf{loop forever do} \\ \quad \left[\begin{array}{l} m_1 : \textbf{noncritical} \\ m_2 : \textbf{await } t = 2 \\ m_3 : \textbf{critical} \\ m_4 : t := 1 \end{array}\right] \end{array}\right]$$

Program TURN (turn taking).

c) *The same questions for program TURN.*
This program satisfies exclusion but neither accessibility nor common accessibility.

exclusion Consider the case in which P_1 enter in critical section (at_{l_3} we have $t = 1$) so P_2 must be in $m_0, m1, m2$ or waiting in front of m_2. The same consideration can be applied when P_2 enters in its critical section.

accessibility P_1 and P_2 enters in critical their section in a forced alternation: since it is not required to a non-critical section to terminate, if process P_1 (resp., P_2) is stuck in its non-critical section, P_2 (resp., P_1) waits inde

finitely for its turn. If both non-critical sections are ensured to terminate, then the protocol would satisfy both requirements.

Problem 0.5

$$\textbf{local } \alpha_1, \alpha_2, \beta_1, \beta_2\text{: } \textbf{channel of boolean}$$

$$P_1 :: \left[\begin{array}{l} \textbf{local } x_1 : \textbf{boolean} \\ l_0 : \textbf{loop forever do} \\ \quad \left[\begin{array}{l} l_1 : \textbf{noncritical} \\ l_2 : \alpha_1 \Leftarrow T \\ l_3 : \beta_1 \Rightarrow x_1 \\ l_4 : \textbf{critical} \\ l_5 : \alpha_1 \Leftarrow F \end{array}\right] \end{array}\right]$$

$$\|$$

$$A :: \left[\begin{array}{l} \textbf{local } y : \textbf{boolean} \\ k_0 : \textbf{loop forever do} \\ \quad k_1 : \left[\begin{array}{l} [k_1^a : \alpha_1 \Rightarrow y; k_2\beta_1 \Leftarrow T; k_3 : \alpha_1 \Rightarrow y] \\ \quad \textbf{or} \\ [k_1^b : \alpha_2 \Rightarrow y; k_4\beta_2 \Leftarrow T; k_5 : \alpha_2 \Rightarrow y] \end{array}\right] \end{array}\right]$$

$$\|$$

$$P_2 :: \left[\begin{array}{l} \textbf{local } x_2 : \textbf{boolean} \\ m_0 : \textbf{loop forever do} \\ \quad \left[\begin{array}{l} m_1 : \textbf{noncritical} \\ m_2 : \alpha_2 \Leftarrow T \\ m_3 : \beta_2 \Rightarrow x_2 \\ m_4 : \textbf{critical} \\ m_5 : \alpha_2 \Leftarrow F \end{array}\right] \end{array}\right]$$

Program MUX-SYNCH (mutual exclusion by synchronous communication).

a) *Argue (informally) that program MUX-SYNCH is a good solution to the mutual exclusion problem. That is, show that each computation of the program satisfies the requirements of exclusion and accessibility.*

Program MUX-SYNCH satisfies exclusion. P_1 (resp. P_2) enters in critical section only if at_{k_3} (resp. at_{k_5}). Being in a selection statement, ($at_{k_3} \wedge at_{k_3}$) is inconsistent. k_1^b (or k_1^a) can be chosen only after the execution of k_3 (resp. k_5), which is enabled only after P_1 (resp. P_2) has terminated its critical section. Program MUX-SYNCH satisfies accessibility. If at_{k_1} , and both the lecture from channels α_1 and α_2 can synchronize, then k_1^a (or k_1^b) is selected; since the critical section is ensured to terminate, eventually α_1 (or α_2) receives F. Control goes back to k_1. The left-over request cannot be discarded infinitely often because k_1^b (or k_1^a) is always enabled and must be eventually selected by k_1.

b) *Consider a non-standard transition system for program MUX-SYNCH, in which transitions associated with communication statements are taken to be just but not compassionate. We refer to this interpretation of the program as MUX-SYNCH-J. Is this program (transition system) a good solution to the mutual exclusion problem? Provide an informal argument in support of a positive answer or, alternately, show a computation that violates one of the two requirements.*
If only justice is ensured for channel communication, the accessibility property would still hold since there is no way the the reading in location k_1^a (or k_1^b) can be disabled, once enabled. Since the request to enter is continuously renewed, even with a justice requirement this will be eventually accepted.

c) *Consider program MUX-ASYNCH which is obtained from program MUX-SYNCH by redeclaring channels $\alpha_1, \alpha_2, \beta_1$ and β_2 as asynchronous channels. Is program MUX-ASYNCH a good solution to the mutual-exclusion problem? Provide an informal argument in support of a positive answer or, alternately, show a computation that violates one of the two requirements.*
With asynchronous channels, the properties would still hold. Exclusion would be guaranteed by the selection in k_1; besides, if the channel is not-empty, then the lecture is always enabled and the corresponding branch of selection is eventually taken.

local $\alpha_1, \alpha_2, \beta_1, \beta_2$: **boolean where** $\alpha_1 = \alpha_2 = \beta_1 = \beta_2 = F$

$$P_1 :: \left[\begin{array}{l} l_0 : \textbf{loop forever do} \\ \quad \left[\begin{array}{l} l_1 : \textbf{noncritical} \\ l_2 : \alpha_1 := T \\ l_3 : \textbf{await } \beta_1 \\ l_4 : \textbf{critical} \\ l_5 : \alpha_1 := F \\ l_6 : \textbf{await } \neg\beta_1 \end{array}\right] \end{array}\right]$$

$$\|$$

$$A :: \left[\begin{array}{l} k_0 : \textbf{loop forever do} \\ \quad k_1 : \left[\begin{array}{l} [k_1^a : \textbf{await } \alpha_1; k_2 : \beta_1 := T; k_3 : \textbf{await } \neg\alpha_1; k_4 : \beta_1 := F] \\ \quad \textbf{or} \\ [k_1^b : \textbf{await } \alpha_2; k_5 : \beta_2 := T; k_6 : \textbf{await } \neg\alpha_2; k_7 : \beta_2 := F] \end{array}\right] \end{array}\right]$$

$$\|$$

$$P_2 :: \left[\begin{array}{l} m_0 : \textbf{loop forever do} \\ \quad \left[\begin{array}{l} m_1 : \textbf{noncritical} \\ m_2 : \alpha_2 := T \\ m_3 : \textbf{await } \beta_2 \\ m_4 : \textbf{critical} \\ m_5 : \alpha_2 := F \\ m_6 : \textbf{await } \neg\beta_2 \end{array}\right] \end{array}\right]$$

Program MUX-SHARED (mutual exclusion by shared variables).

d) *Is program MUX-SHARED a good solution to the mutual-exclusion problem? Present an informal argument or show a computation that violates one of the two requirements.*
Program MUX-SHARED satisfies exclusion due to the selection statement in location k_1. Accessibility is also satisfied because the **await** α_1 (resp. **await** α_2) are always enabled if a process attempts to enter in its critical section, and must be eventually taken.

3 Additional exercices

Esercizio1

Dato un linguaggio $L \subseteq A^*$, dimostrare se che L è un linguaggio star-free, allora L è definibile nel frammento al prim'ordine di $S1S_A$, con la relazione di ordinamento $<$ e i predicati unari Q_a, con $a \in A$.
Un linguaggio star-free è costruito su:

1. un alfabeto A
2. l'insieme vuoto $\emptyset$
3. operatori booleani *AND OR NOT*
4. operazione di concatenazione $\cdot$

Mentre un linguaggio del frammento al prim'ordine di $S1S_A$ è costruito su:

1. un alfabeto A
2. una costante 0
3. operatori booleani *AND OR NOT*
4. quantificatore $\forall$ e $\exists$
5. un insieme di variabili $\{x, y, \ldots\}$
6. un insieme di predicati $\{<\} \cup \{Q_a \forall a \in A\}$

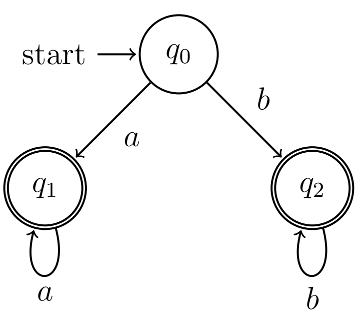
Esercizio2

Dimostrare che l'insieme dei linguaggi riconosciuti da automi di Büchi su alberi infiniti con insieme degli stati finali singoletto è strettamente contenuto nell'insieme dei linguaggi riconosciuti da automi di Büchi su alberi infiniti.
L'insieme degli stati finali di un automa di Büchi su alberi infiniti non può essere sempre ristretto ad un singoletto. In particolare, sia $\Sigma = \{a, b\}$ un alfabeto e siano:

1. $T_a = \{t \in T_A^\omega | \forall w \in dom(t) : t(w) = a\}$ albero con sole a
2. $T_b = \{t \in T_A^\omega | \forall w \in dom(t) : t(w) = b\}$ albero con sole b

Preso $T = T_a \cup T_b$, questo è accettato dal seguente automa di Büchi :

$$\mathcal{A} = (\{q_0, q_1, q_2\}, \Sigma, \Delta, q_0, \{q_1, q_2\})$$



Dimostriamo per assurdo che non può esistere un automa di Büchi $\mathcal{A}' = (Q, \Sigma, \Delta', q_0, \{q_f\})$ in grado di accettare T con uno stato finale singoletto.

Consideriamo un cammino $\sigma_a = a^\omega$ in un albero $t_a \in T_a$: esiste quindi una run π_a su A' tale che $\pi_{a|\sigma}$ ha infinite occorrenze di q_f. Perciò esiste $w_a = a^i$ con $i > 1$ tale che $\pi_a(w_a) = q_f$.
Similmente, consideriamo un cammino $\sigma_b = b^\omega$ in un albero $t_b \in T_b$: esiste quindi una run π_b su A' tale che $\pi_{b|\sigma}$ ha infinite occorrenze di q_f. Perciò esiste $w_b = b^j$ con $j > 1$ tale che $\pi_b(w_b) = q_f$.
Consideriamo ora l'albero ottenuto sostituendo il sotto-albero di t_a radicato in w_a con il sotto-albero di t_b radicato in w_b. Abbiamo così ottenuto un albero infinito i cui cammini sono condivisi o con t_a o un cammino che in partenza ha a^{i-1} e poi continua con cammini accettati di T_2. Quest'ultimo legge parole del tipo $a^i - 1, b^\omega$ con $i - 1 > 0$. Segue che $\mathcal{A}'$ è forzato ad accettare anche alberi $t \notin T_a \cap T_b$.

Esercizio3

Sia $A = \{a, b\}$ e $T_1 = \{t \in T_A^\omega$:tutti i cammini di t contengono un numero finito di occorrenze di $a\}$. T_1 contiene l'insieme di tutti gli alberi t_i, con $i \geqslant 0$, tali che t_i ha un'occorrenza di a nelle posizioni $\epsilon, 1^{m_1}0, \ldots, 1^{m_1}01^{m_2}0 \ldots 1^{m_i}0$, con $m_1, m_2, \ldots, m_i > 0$. Immaginiamo che esista un automa di Büchi $\mathcal{A} = (\mathcal{Q}, A, \Delta, q_0, F)$ con $n+1$ stati, con $n \geqslant 1$, incluso lo stato iniziale q_0 che occorre solo in posizione ϵ tale che $L(\mathcal{A}) = T_1$ e sia r un run di successo di $\mathcal{A}$ su t_n. Mostrare che deve esistere un cammino in t_n contenente 3 nodi u,v e w, con $u < v < w$, tali che $r(u) = r(w) = s \in Fet_n(v) = a$.
Consideriamo una run accettata π su $\mathcal{A}$. Abbiamo che $\forall \sigma, In(\pi_{|\sigma}) \cap F \neq \emptyset$. Questo ci porta a considerare due sequenze: $k_1, ...k_{n+1}$ e $f_1, ..., f_{n+1}$ tali che:

1. $\pi(1^{k_1}) = f_1 \in F$ (deriva dal fatto che 1^ω è un cammino accettato)
2. $\pi(1^{k_1}01^{k_2}) = f_2 \in F$ (deriva dal fatto che $1^{k_1}01^\omega$ è un cammino accettato)
3. ...
4. $\pi(1^{k_1}01^{k_2}...1^{k_n}01^{k_{n+1}}) = f_{n+1} \in F$ (deriva dal fatto che $1^{k_1}01^{k_2}...1^{k_n}01^\omega$ è un cammino accettato)

Dato che, il numero di stati dell'automa è $n + 1$ e q_0 occorre solo ad ϵ, deve esistere una coppia di stati finali ripetuta. Siano quindi $i < j \leq n + 1$ tali che $f_i = f_j$. Abbiamo trovato una tripla:

1. $u = 1_1^k01_2^k...1_i^k$
2. $v = 1_1^k01_2^k...1_i^k0$
3. $v = 1_1^k01_2^k...1_i^k01_{i+1}^k...1_j^k$

tale che $\pi(u) = \pi(w) = f_i = f_j \in F$ e $t_n(v) = a$

Esercizio4

Siano $C = \{c_1, \ldots, c_m\}$ e $\bar{c}= (c_1, \ldots, c_m)$. Sia dato $T \subseteq T_A^\omega$ tale che $T = T_0 \cdot \bar{c}(T_1, \ldots, T_m)^{\omega_{\bar{c}}}$ con $T_0, T_1, \ldots, T_m \subseteq T_{A \cup C}$ insiemi riconoscibili. Mostrare che T è riconoscibile da un automa di Büchi .
Siano $C = \{c_1, \ldots, c_m\}$ e $\bar{c}= (c_1, \ldots, c_m)$ e $\forall i\ 0 \leq i \leq m$, sia $T_i \subseteq T_{A \cup C}$ un insieme di alberi riconosciuti dall'automa di Büchi $\mathcal{A}_i = (\mathcal{Q}_i, A \cup C, \Delta_i, \{q_{0_i}\}, F)$. L'insieme $T \subseteq T_A^\omega$ tale che $T = T_0 \cdot \bar{c}(T_1, \ldots, T_m)^{\omega_{\bar{c}}}$ è riconosciuto dall'automa di Büchi $\mathcal{Q}, A, \Delta, \mathcal{Q}_0, F)$ dove:

- $\mathcal{Q} = \bigcup_{0 \leq i \leq m} \mathcal{Q}_i$
- $\mathcal{Q}_0 = \{(q_{0_0})\} \cup \{q_{0_l} | 1 \leq l \leq m, \exists c \in C \cdot (q_{0_0}, c, q_x, q_y) \in \Delta_0\}$
- $F = \bigcup_{0 \leq i \leq m} F_i$
- $\Delta = \bigcup_{0 \leq i \leq m} ((\Delta_i \setminus \Delta_i^-) \cup \Delta_i^+)$ dove $\forall i \; 0 \leq i \leq m$:
  - $\Delta_i^- = \{(q_j, c, q_y, q_z) \in \Delta_i | c \in C\} \cup \{(q_k, c, q_j, q_x) | a \in A, \exists c \in C \cdot (q_j, c, q_y, q_z) \in \Delta_i\}$
  - $\Delta_i^+ = \{(q_k, c, q_{0_l}, q_x) | 1 \leq l \leq m, a \in A, \exists c \in C \cdot \{(q_k, a, q_j, q_x), (q_j, c, q_y, q_z)\} \subseteq \Delta^-\}$

Quindi, Δ è ottenuta rimuovendo da ogni Δ_i le transizioni che leggono un simbolo in C e sostituendo le transizioni immediatamente precedenti (che leggono un simbolo di A) con nuove transizioni, leggenti lo stesso simbolo ma indirizzate verso uno stato iniziale q_{0_l} preso da uno degli $\mathcal{A}_l$ con $l > 1$. Infine, se lo stato iniziale q_{0_0} può leggere un simbolo in C, ogni altro stato iniziale q_{0_l} deve essere aggiunto all'insieme degli stati iniziali di $\mathcal{A}$

Esercizio5

Dimostrare la (correttezza e completezza della) caratterizzazione di uno degli operatori di CTL (diverso da AF) quale minimo punto fisso di un'opportuna trasformazione di predicato.

Dimostro che l'operatore EFf_1 è un minimo punto fisso del predicato $\tau(Z) = f_1 \vee EFZ$.

Correttezza) Dimostro che EFf_1 è un punto fisso di $\tau(Z)$. Uno stato $s \models EFf_1 \Leftrightarrow \exists$ un cammino $s...s_i$ tale che $s_i \in f_1$. Quindi:

1. $s \in f_1$, oppure
2. $\exists s_1$ tale che $(s, s_1) \in R$ e $s_1 \in EFf_1$

Segue che $s \models EFf_1$ se e solo se $s \models f_1 \vee EXEFf_1 = \tau(EFf_1)$.

Completezza) Dimostro che $EFf_1 = \cup_i \tau^i(\mathit{FALSE})$. In primo luogo, dimostro la monotonocità della trasformazione di predicato τ. $\forall P_1 \subseteq P_2$,

$$\begin{aligned} s \in \tau(P_1) &\Rightarrow s \models f_1 \text{or} \exists s' \text{t.c.}((s, s') \in R \wedge s' \in P_1) \\ &\Rightarrow s \models f_1 \text{or} \exists s' \text{t.c.}((s, s') \in R \wedge s' \in P_2) \\ &\Rightarrow s \in \tau(P_2) \end{aligned}$$

Dato che il numero di stati è finito, la monotonocità della trasformazione implica che esiste un limite alla catena crescente $\tau^0(\mathit{FALSE}) \subseteq \tau^1(\mathit{FALSE}) \subseteq \ldots \subseteq \tau^i(\mathit{FALSE})$. Poniamo questo limite a $\tau^l(\mathit{FALSE}) = \cup_i \tau^i(\mathit{FALSE})$, dimostro che $\tau^l(\mathit{FALSE}) = EFf_1$.

$\subseteq$ Per induzione su i, mostro che $\forall i, \tau^i(\mathit{FALSE}) \subseteq EFf_1$.
Per $i = 0$ abbiamo che $\tau^0(\mathit{FALSE}) = \mathit{FALSE} \subseteq EFf_1$.
Assumiamo che $\tau^i(\mathit{FALSE}) \subseteq EFf_1$. Per la monotonocità di τ abbiamo che $\tau^{i+1}(\mathit{FALSE}) \subseteq \tau(EFf_1)$. Dato che EFf_1 è un punto fisso di τ, abbiamo che $\tau^{i+1}(\mathit{FALSE}) \subseteq EFf_1$. Segue che $\tau^l(\mathit{FALSE}) \subseteq EFf_1$.

$\supseteq$ Sia $s \models EFf_1$. Questo significa che $\exists s_i \ldots s_0$ cammino tale che $s_0 \models f_1$ con $s = s_i$.
Per induzione, mostro che $s_i \in \tau^{i+1}(\mathit{FALSE})$.
Per $i = 0$ abbiamo che $s_0 \models f_1 = f_1 \vee \mathit{FALSE} = \tau^1(\mathit{FALSE})$.
Per $i > 0$, sia $s_{i-1} \in \tau^i(\mathit{FALSE})$. Abbiamo che $s_i \models EX\tau^i(\mathit{FALSE})$ perciò $s_i \models f_1 \vee EX\tau^i(\mathit{FALSE}) = \tau^{i+1}(\mathit{FALSE})$.
Segue che $\forall s \in EFf_1 \exists i$ tale che $s \in \tau^i(\mathit{FALSE})$, cioè $EFf_1 \subseteq \tau^l(\mathit{FALSE})$.

Esercizio6

Dimostrare la (correttezza e completezza della) caratterizzazione di uno degli operatori di CTL (diverso da EG) quale massimo punto fisso di un'opportuna trasformazione di predicato.

Dimostro che l'operatore AGf_1 è il massimo punto fisso del predicato $\tau(Z) = f_1 \wedge AGZ$.

Correttezza) Dimostro che AGf_1 è un punto fisso di $\tau(Z)$. Uno stato $s \models AGf_1 \Leftrightarrow s_i \models f_1$ e $\forall s...s_i$ cammini, $\forall i$, vale che $s_i \in f_1$. Quindi:

1. $s \in f_1$, oppure
2. $\forall s_1$ tale che $(s, s_1) \in R$, allora vale $s_1 \in AGf_1$

Segue che $s \models AGf_1$ se e solo se $s \models f_1 \vee AXAGf_1 = \tau(AGf_1)$.

Completezza) Dimostro che $AGf_1 = \cap_i \tau^i(TRUE)$. In primo luogo, dimostro la monotonocità della trasformazione di predicato τ. $\forall P_1 \subseteq P_2$,

$$\begin{aligned} s \in \tau(P_1) &\Rightarrow s \models f_1 \text{and} \forall s' \text{t.c.}((s, s') \in R \wedge s' \in P_1) \\ &\Rightarrow s \models f_1 \text{and} \forall s' \text{t.c.}((s, s') \in R \wedge s' \in P_2) \\ &\Rightarrow s \in \tau(P_2) \end{aligned}$$

Dato che il numero di stati è finito, la monotonocità della trasformazione implica che esiste un limite alla catena decrescente $\tau^0(TRUE) \subseteq \tau^1(TRUE) \subseteq \ldots \subseteq \tau^i(TRUE)$. Poniamo questo limite a $\tau^l(TRUE) = \cap_i \tau^i(TRUE)$, dimostro che $\tau^l(TRUE) = AGf_1$.

- $\supseteq$ Per induzione su i, mostro che $\forall i, AGf_1 \subseteq \tau^i(TRUE)$.
  Per $i = 0$ abbiamo che $AGf_1 \subseteq \tau^0(TRUE) = TRUE$.
  Assumiamo che $AGf_1 \subseteq \tau^i(TRUE)$. Per la monotonocità di τ abbiamo che $\tau(AGf_1) \subseteq \tau^{i+1}(TRUE)$. Dato che AGf_1 è un punto fisso di τ, abbiamo che $AGf_1 \subseteq \tau^{i+1}(TRUE)$.
  Segue che $AGf_1 \subseteq \tau^l(FALSE)$.
- $\subseteq$ Sia $s \in \tau^l(TRUE)$. Voglio mostrare che $\forall(s_i, s_{i+1}) \in R$ su tutti i cammini $s_0, \ldots, s_i$, con $s_0 = s$, allora $s_i \models f_1$ (per avere che $s_i \models AGf_1$). Per induzione, mostro che $\forall(s_i, s_{i+1}) \in R$ su tutti i cammini $s_0, \ldots, s_i$, con $s_0 = s$, allora $s_i \in \tau^l(TRUE)$
  Per $i = 0$: triviale (per ipotesi)
  Assumiamo che $s_i \models \tau^l(TRUE)$, abbiamo che $s_i \models f_1 \wedge AX\tau^{l+1}(TRUE)$, quindi $s_{i+1} \in \tau^{l+1}(TRUE)$. Dato che $\tau^{l+1}(TRUE) = \tau^l(TRUE)$, abbiamo che $s_{i+1} \in \tau^l(TRUE)$.
  Segue che $\tau^l(TRUE) \subseteq AGf_1$.