1 Calcolo complessità

1.1 cardinalità dello spazio comportamentale

prima di calcolare la complessità temporale per la funzione *step* dobbiamo fermarci sulla cardinalità dell'insieme dei nodi nella spazio comportmentale:
ogni nodo è identificato esattamente da un contesto e ogni contesto è caratterizzato dallo stato in cui ogni automa è presente e da il determinato evento contenuto in ogni link della rete, possiamo indicare con questa nomenclatura:

$$n_1, n_2, ..., n_a, l_1, l_2, ..., l_k$$

Dove per n_i intendiamo il numero di stati presenti nell'automa i, l_j il numero di eventi presenti nel link j, a il numero di automi nella rete e k il numero di link

Vista questa caratterizzazione ogni singola variazione di uno dei valori del contesto (stato o evento) implica un possibile nuovo contesto e quindi un nuovo nodo. Consideriamo allora tutte le possibili permutazioni:

$$\prod_{i=1}^{a} n_i \prod_{j=1}^{k} l_j$$

Questo valore rappresenta il numero massimo possibile di nodi che possono essere generati in ogni rete comportamentale, semplificando:

$$\prod_{i=1}^{a} n_i \prod_{j=1}^{k} l_j \leq n^a l^k$$

con $n = \max(n_1, n_2, ..., n_a)$ e $l = \max(l_1, l_2, ..., l_k)$.
Quindi se s è il numero esatto di nodi nella rete comportamentale possiamo indicare:

$$s = O(n^a l^k)$$

Questa crescita esponenziale del numero di nodi si può facilmente notare anche con i semplici esempi giocattolo lasciati per il testing per il progetto, sarà necessario tenere in considerazione questo valore s perchè risulterà critico nelle versioni successive.

1.2 Calcolo della complessità nella funzione step

Consideriamo ora lo pseudocodice di *step*, vista la natura ricorsiva prima troviamo il costo di ogni ricorsione: il cuore dell'algoritmo si trova nel ciclo while di riga 4 dove avviene un controllo di tutte le transizioni in uscita, quindi il ciclo si ripeterà esattamente t_{ji} volte dove t_{ji} rappresenta il numero delle transizioni uscenti dal j-esimo stato presente nell'i-esimo automa.

Controllando le singole funzioni definite le più costose risultano *isBufferFree* (riga 5) e *createNewContext* (riga 6); entrambe le funzioni ciclano sul numero di azioni prodotte in uscita dalla transizione presa in esame, definiremo l_{ki} come il numero di link attivati dalla k-esima transizione presente nell i-esimo automa.

Visto che questo controllo delle transizioni viene ripetuto per ogni stato "puntato" da ogni automa possiamo calcolare il costo complessivo di ogni ricorsione:

$$\sum_{i=1}^{a} t_{ij} l_{ik} \leq \sum_{i=1}^{a} (n_i - 1) l_{ki}$$

Considerando la rete completamente connessa (che rappresenta il caso peggiore) da ogni stato possono uscire al massimo $n_i - 1$ transizioni:

$$\leq (\sum_{i=1}^{a}(n_i - 1))(a-1)a$$

Visto che in ogni automa al massimo possiamo avere $a-1$ link:

$$\leq a(n-1)(a-1) = O(a^3 n)$$

1.3 calcolo della complessità nella ricorsione di step

step segue una strategia in profondità e non ha una condizione specifica di stop, si ferma solo quando ha calcolato tutto lo spazio comportamentale e quindi trovato tutti i nodi raggiungibili, una ricorrenza che modellizza lo pseudocodice può essere:

$$\begin{cases} T(1) = O(a^3 n) \\ T(s) = T(s-1) + O(a^3 n) \end{cases}$$

Visto che ogni ricorrenza è indipendente dalle altre il costo totale diventa:

$$\sum_{i=1}^{s} T(i) = O(a^3 n^{a+1} l^k)$$

La complessità quindi risulta molto sensibile alla crescita del numero degli automi e del numero di link questo a causa della crescia esponenziale del possibile numero di nodi nello spazio comportamentale