UNIVERSITÀ
DEGLI STUDI
DI BRESCIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Sviluppo Programma Per L'Elaborazione Di Automi A Stati Finiti

Relatori:

Rovetta Alberto

Tamburini Alberto

Anno Accademico 2020/2021

INTRODUZIONE

Si è scelto di utilizzare il linguaggio python, nonostante si fosse inizialmente svolto il progetto in java. Una bozza del lavoro è presente qui. Quest'ultimo è stato abbandonato a causa dell'illegibilità che stava assumendo il codice, divenendo eccessivamente verboso e limitante. Dunque, dopo attenta riflessione si è optato per il linguaggio python, in particolare per l'enorme disponibilità di strutture dati presenti nativamente senza alcuna complicazione (si pensi ai dizionari e le tuple, dove queste ultime in java non sono presenti se non importando un modulo esterno) e per la presenza di strutture di controllo semplici, leggibili ed estremamente efficienti; si pensi alle list comprehension.
La leggibilità rimane la ragione primaria, in quanto apprezziamo un codice quanto più elegante quando possibile, obiettivo che java non ci ha consentito in alcun modo di ottenere data la natura del progetto.
Segnaliamo che, ove presente la ricorsione, non è stato compiuto alcuno sforzo per ottenere una ricorsione in coda (come era stato fatto in java); questo perché in python la ricorsione in coda non esiste come qui documentato dal benevolo dittatore a vita di python.
Tutto il codice utilizzato è stato testato sotto python3.9, python3.8, python3.7, python3.6. Tutte queste versioni funzionano in ubutu18.04 e sono state sviluppate sotto arch linux (chiaramente aggiornato) sul quale funzionano. Non si ha alcuna garanzia su sistemi operativi quali mac-os o windows. Ad ogni modo, su windows verosimilmente non dovrebbe andare in quanto non conforme per quanto riguarda i separatori nel filesystem (usato '/' anzichè '\'), i quali tuttavia sono presenti solo all'interno della suite di test.
Con versioni meno recenti di python non può funzionare a causa dell'utilizzo di alcune primitive non presenti precedentemente nella sintassi del linguaggio.
Va anche chiarito che i test sono incentrati sui primi compiti. Per gli ultimi, quali le regex, sarebbe stato necessario computare l'equivalenza fra varie regex prodotte.

ALGORITMI UTILIZZATI

ALGORITMO PARTE UNO

La prima implementazione del programma consente di creare uno spazio comportamentale ricevendo in ingresso un file ".json" in cui sono presenti gli automi, con relativi stati e transizioni. (file esempio json)
Nello pseudocodice il file json è già stato decodificato rendendolo un insieme di oggetti.

BUILD-CFA-SPACE(net)
▷ **s[#key]** ← value (indica un dizionario, in cui "key" indica la chiave con cui effettuare la ricerca)

```
init ← init[net]
present[#init] ← init
▷ S è una coda
S ← ∅
Enqueue(S, init)
do repeat
        state ←DEQUEUE(S)
        ▷ allowed-transition descrive le transizioni che possono uscire da state
        for each transition ∈ allowed-transition[state]
                do next ← successor[transition]
                ▷ mette next in un dizionario nexts con chiave transition
                INSERT-SUCCESSOR(state, transition, next)
                if present[#next] = NIL
                        ENQUEUE(S, next)
                        present[#next] ← next
until QUEUE-EMPTY(S)
pruned ← PRUNE(present)
return pruned
```

PRUNE(states)

```
▷ è indifferente che si prendano le chiavi o i valori
▷ mantain, remove, visited sono dizionari
mantain ← ∅
remove ← ∅
for each state ∈ states
        do if mantain[#state] = NIL AND remove[#state] = NIL
                than
                visited ← ∅
                PRUNE-RECURSION(state, visited, mantain, remove)

▷ element è considerato un elemento con attributi key e value
for each element ∈ remove

        do state ← key[element]
        for each next ∈ states
                do rimuovi tutte le transazioni con successor = next

return mantain
```

```
PRUNE-RECURSION(state, visited, mantain, remove)
    ▷ is-final[state] è considerato un attributo booleano di state
    if is-final[state]
        than
        mantain[#state] ← TRUE
        return FALSE

    for each element ∈ nexts[state]
        do trans ← key[element]
        next ← value[element]
        if remove[#next] = NIL
            than
            ▷ viene creato un oggetto arc con tre attributi come segue
            state[arc] ← state
            transition[arc] ← trans
            successor[arc] ← next
            if visited[#arc] = NIL
                than visited[#arc] ← TRUE
                if mantain[#next] ≠ NIL OR NOT
                        PRUNE-RECURSION(next, visited,
                                        mantain, remove)
                    than if mantain[#next] = NIL
                        than mantain[#state] ← TRUE
                        if remove[#state] ≠ NIL
                            than DELETE(remove, state)
                    return FALSE

    if mantain[#state] = NIL
        remove[#state] ← TRUE
    return TRUE
```

ALGORITMI PARTE DUE

La seconda implementazione del programma consente di creare uno spazio comportamentale relativo ad una osservazione ricevendo in ingresso un file “.json” in cui sono presenti gli automi, con relativi stati e transizioni. (file esempio json)
Nello pseudocodice il file json è già stato decodificato rendendolo un insieme di oggetti.
Inoltre, sia qui che in ALGORITMO PARTE TRE is-final è TRUE quando lo stato è finale E non ha alcuna osservazione successiva.

```
▷ Precondizione è che obs sia un vettore, al limite vuoto. Non deve essere NIL.
BUILD-CFA-SPACE-OBSERVATION(net, obs)
        init ← init[net]
        obs_index[init] ← 0
        DFS-VISIT(init, obs, ∅, ∅)
        ▷ mantain è un dizionario
        mantain ← ∅
        ▷ remove è una lista
        remove ← ∅
        PRUNE-SPACE-OBS(init, mantain, remove, ∅)
        for each arc ∈ remove
                do state ← state[arc]
                trans ← transition[arc]
                next ← successor[arc]
                next_list ← nexts[state]
                if mantain[#next] = NIL AND next_list[#trans] ≠ NIL
                        than DELETE(state, trns)
        return DICT-TO-LIST(mantain)

DFS-VISIT(state, obs, grey, result)
        grey[#state] ← TRUE
        if EMPTY(obs)
                than obs_val ← NIL
        else
                obs_val ← obs[0]

        for each transition ∈ allowed-transition[state]
                trans_obs ← observation[transition]
                than if trans_obs = NIL OR trans_obs = obs_val
                        than next ← successor[transition]
                        if grey[#next] = NIL
                                than APPEND(result, next)
                                if trans_obs ≠ NIL
                                        than obs_index[next] ← obs_index[state] + 1
                                if trans_obs = obs_val
                                        than new_obs ← SUBVECTOR(obs, 1, len[obs])
                                        DFS-VISIT(next, new_obs, grey, result)
                                else
                                        DFS-VISIT(next, obs, grey, result)
                                REMOVE(grey, state)
```

PRUNE-SPACE-OBS(state, mantain, remove, grey)

```
reach_final ← FALSE
if is-final[state]
        than mantain[#state] ← TRUE
        reach_final ← TRUE
for each element ∈ nexts[state]
        do trans ← key[element]
        next ← value[element]
        state[arc] ← state
        transition[arc] ← trans
        successor[arc] ← next
        if grey[#arc] = NIL
                than grey[#arc] ← TRUE
                if PRUNE-SPACE-OBS(next, mantain, remove, grey)
                        than if NOT reach_final
                                than mantain[#state] ← TRUE
                                reach_final ← TRUE
                else
                        APPEND(remove, arc)
                REMOVE(grey, arc)
return reach_final
```

ALGORITMO PARTE TRE

La terza implementazione del programma consente di creare una regex ricevendo in ingresso una presunta osservazione relativa allo spazio degli stati in cui sono presenti gli automi, con relativi stati e transizioni.

BUILD-REGEX-M1(net, obs)

```
net ← BUILD-CFA-SPACE-OBSERVATION(net, obs)
▷ controllo osservazione valida
if net ≠ NIL AND len[net] < 1
        than error “purtroppo non è nota l’osservazione relativa alla rete”
▷ rilevanze
for each state ∈ net
        do for each element ∈ nexts[state]
                do trans ← key[element]
                if relevance[trans] = NIL OR relevance[trans] = “”
                        than relevance[trans] ← ε
▷ stati finali
for each state ∈ net
        do if is-final[state]
                than acceptation[state] ← TRUE
▷ All’interno di EspressioneRegolare per ottenere l’espressione di rilevanza r si utilizza
l’attributo relevance della transizione
return EspressioneRegolare(net)
```

ALGORITMO PARTE QUATTRO

La quarta implementazione del programma consente di creare il diagnosticatore ricevendo in ingresso uno spazio comportamentale.

Costruzione chiusura

SILENT-CLOSURE(space_state, init_state)

```
valid ← FALSE
for each prev_trans ∈ prev_trans[init_state]
        do if observation[prev_trans] ≠ NIL
                than valid ← TRUE
                break
if valid
        than T← ∅
        APPEND(T, init_state)
        ▷ Le seguenti strutture dati sono dei dizionari
        work_space[#T[0]] ← NIL
        to_decorate ← ∅
        final_states ← ∅
        exit_states ← ∅
        grey_list[#T[0]] ← T[0]
        for each state ∈ T
                ▷ external_nexts, inizialmente vuoto, conterrà le transizioni con
        osservazione, ovvero quelle che escono dalla chiusura
                ext ← external_nexts[state]
                do for each element ∈ nexts[state]
                        do trans ← key[element]
                        next ← value[element]
                        if observation[trans] ≠ NIL
                                than REMOVE(nexts[state], trans)
                                ext[#trans] ← next
                        if grey_list[#next] = NIL
                                than if observation[trans] = NIL
                                        than work_space[#next] ← NIL
                                        APPEND(T, next)
                        grey_list[#next] ← next
                if len(external_nexts[state]) > 0 OR is-final[state]
                        than to_decorate[#state] ← NIL
                if is-final[state]  ≠ NIL
                        than final_states[#state] ← -100
                if len(external_nexts[state]) > 0
                        than exit_states[#state] ← -100

        ▷ Incarna EspressioniRegolari, dove gli elementi di to_decorate sono stati di
accettazione. Il value di ogni elemento dei dizionari contiene la relativa regex.
        DECORATE(work_space, to_decorate, final_states, exit_states)
        final_states[closure] ← final_states
        exit_states[closure] ← exit_states
        return closure
else
        return NIL
```

Costruzione spazio delle chiusure

BUILD-CLOSURE-SPACE(space_state)

```
closures ← ∅
for each init_state ∈ space_state
        do closure ← SILENT-CLOSURE(space_state, init_state)
        if closure ≠ NIL
                than APPEND(closures, closure)
                init_closure[closure] ← init_state

for each closure ∈ closures
        do out[closure] ← BUILD-NEXT(exit_states[closure], closures)
return closures
```

BUILD-NEXT(exit_states, closures)

```
out ← ∅
for each element_state ∈ exit_states
        do state ← key[element_state]
        for each element_trans ∈ external_nexts[state]
                do trans ← key[element_trans]
                out[#observable[trans]] ←  ∅
for each element_state ∈ exit_states
        do state ← key[element_state]
        regex ← value[element_state]
        for each element_trans ∈ external_nexts[state]
                do trans ← key[element_trans]
                successor ← value[element_trans]
                for each closure ∈ closures
                        if successor = init_closure[closure]
                                than new_regex ← CONCAT(regex, relevance[trans])
                                vect ← [closure, new_regex]
                                APPEND(out[#observable[trans]], vect)
                                break
return out
```

Costruzione di un diagnosticatore

DIAGNOSTICATOR(space_state)

```
closures ← BUILD-CLOSURE-SPACE(space_state)
for each closure ∈ closures
        do temp_regex ← NIL
        for each element ∈ final_states[closure]
                do to_add ← value[element]
                if temp_regex ≠ NIL
                        than temp_regex ← CONCAT(temp_regex, "|", to_add)
                else
                        temp_regex ← to_add
        regex[closure] ← temp_regex
return closures
```

CONFRONTO E SCELTE NELLE STRUTTURE DATI

Durante la progettazione di tale elaborato sono state utilizzate diverse strutture dati: liste(ovvero vettori), dizionari ed insiemi.
Le liste sono state utilizzate quando vi era la consapevolezza di dover accedere a tutti gli elementi che sarebbero stati inseriti all'interno della stessa, in altre parole si era certi di dover utilizzare una struttura di controllo "for each", o comunque di analogo quale un iteratore. Alcuni semplici esempi possono essere visti all'interno delle classi che fanno da base per entrambi i metodi, ovvero ComportamentalFANSpace e ComportamentalFANSObservation; in entrambi, gli stati sono rappresentati tramite una lista. Questa lista viene interamente analizzata dal diagnosticatore o comunque quando si svolge la diagnosi.

La seconda struttura utilizzata sono stati i dizionari in particolare quando vi era la necessità di collezionare oggetti e attributi su cui un accesso diretto, anziché sequenziale, poteva portare a un netto miglioramento delle prestazioni (in generale per effettuare la ricerca di un oggetto). Alcuni semplici esempi possono essere visti nelle medesime classi precedentemente citate, all'interno dei loro metodi "prune": in entrambi i casi è presente un dizionario che tiene traccia degli stati che durante la ricerca si sono mostrati finali; dunque, se uno stato può raggiungere uno stato all'interno di questo dizionario deve essere mantenuto (e inserito nel dizionario stesso). Qui, una lista avrebbe richiesto ogni volta una ricerca di costo sempre crescente fino, al limite, a n; tramite un dizionario l'accesso avviene in O(1) mediamente. Chiaramente ciò è vero posta l'ipotesi che la funzione di hash non abbia collisioni frequenti. Nel codice si è usato l'hash di default di python, il quale non dovrebbe generare troppe collisioni. Il caso ammortizzato rimane O(n). Va sottolineato che vi sono anche svantaggi:
il tempo di calcolo dell'hash rende O(1) un'operazione relativamente lunga se comparato con l'O(1) dell'iteratore;
i dizionari occupano più spazio in memoria.
Altri esempi possono essere individuati, ad esempio, all'interno della classe Closure dove l'elenco degli stati da decorare/finali/di uscita, con relative regex, sono tutti memorizzati tramite dictionary, sempre per la stessa ragione.
Un uso diverso dei dizionari lo si può vedere nei vari metodi "dict_per_json", i quali hanno come uscita un dizionario da una classe; in questo caso l'uso è dettato dal fatto che in python i dizionari vengono "naturalmente" tradotti in json, e abbiamo sfruttato questo automatismo per fornire le uscite persistenti.

Altri esempi si possono osservare nelle operazioni di rimozione:
nel caso di liste non si effettua mai la rimozione in loco, ma si crea una lista parallela non contenente gli elementi da rimuovere; questo perché il costo dell'operazione non sarebbe O(1), in quanto bisogna scalare tutti gli elementi successori;
nel caso di dictionary viene effettuata la rimozione in loco.
Chiaramente l'operazione di rimozione nelle liste rimane di complessità O(n); tuttavia in alcuni casi si è deciso di mantenere la lista in preferenza al dizionario in quanto la trasformazione da lista a dizionario e viceversa sarebbe costata nel complesso 2n ed era noto che la rimozione avrebbe coinvolto un solo elemento, potendo quindi troncare l'iteratore prima di arrivare a n (ammesso che non fosse l'ultimo elemento); inoltre così si son potute usare le list comprehension, ben più efficienti di un tradizionale for loop.
Una possibile alternativa poteva essere l'utilizzo di strutture dati del tipo dict2list e list2dict per mappare le due strutture in caso di necessità. Si è deciso di non farlo per non rendere il codice troppo caotico e risparmiare memoria. Tuttavia, conoscendo il sistema di destinazione questo aspetto potrebbe essere mutato: se infatti si ipotizzasse una macchina con RAM molto estesa, si potrebbe rendere il passaggio più auspicabile, risolvendo il problema della leggibilità tramite le facilitazioni del linguaggio stesso.

Gli insiemi sono stati utilizzati solo all'interno di "OutTransition", per poter avere accesso diretto sui link, in modo che al crescere del numero di link il tempo per ottenerne uno particolare rimanga costante. Ciò è risultato particolarmente comodo in quanto questo accesso avviene svariate volte in ogni porzione di codice. Avere qui un accesso linearmente crescente col numero di link avrebbe portato a costi irragionevoli anche nei primi compiti, se in ingresso fossero state fornite reti composte da molti FA.
Gli esempi forniti non sono esaustivi, tuttavia si rende noto che ogniqualvolta sia stato necessario un compito simile si è usata la stessa struttura dati per la stessa ragione.
Si noti che non vi è alcun accenno all'estensione delle dimensione degli spazi utilizzati. Infatti tutte le strutture dati in questione non hanno nativamente una dimensione massima statica. Dunque nessun accorgimento è stato preso, se non da un punto di vista dell'efficienza, ovvero:
quando possibile, si è preferito costruire la struttura dati e contestualmente effettuarne il riempimento, o tramite una primitiva del linguaggio o tramite una list/dictionary comprehension. Questo approccio è più efficiente per l'implementazione stessa del linguaggio. Quando invece non era computabile nell'immediato, o avrebbe inficiato eccessivamente la leggibilità del codice, si è deciso di creare la struttura e poi di effettuarne il riempimento successivamente.

LIMITAZIONI NELLE PROVE

Per quanto riguarda le limitazioni, non ne vengono riscontrate durante lo svolgimento dei benchmark. Tuttavia non è possibile valutare la correttezza di quelli forniti nella rete fornita denominata “benchmark”; comunque la computazione viene svolta giungendo senza errori a terminazione. L’impossibilità è dettata dal fatto che non è a noi noto il risultato che dovrebbe essere presente in uscita e computarlo a mano ci è parso troppo oneroso. Tuttavia un controllo minimale è stato comunque svolto: data una osservazione che doveva contenere, per esempio, solo i simboli ‘f’ e ‘***ε***’, si è verificato che l’uscita contiene solo questi simboli. Oltre non sappiamo valutare.

Per quanto non sia una limitazione dal punto di vista delle funzionalità, ci pare necessario sottolineare una limitazione derivante dalla non semplificazione delle regex. In svariate prove si sono visti risultati del tipo:
“(***ε***|f)| (f|***ε***)” oppure “(***ε***|***ε***)” oppure “(fa)***ε***” oppure “(fa)***ε*** | (fa) | ***ε***(fa)”. In tal senso sono stati presi accorgimenti solo minimali, ad esempio in modo da evitare nella maggior parte dei casi delle ripetizioni di ***ε***. Tuttavia, soprattutto per il diagnosticatore, questi controlli non sono del tutto esaustivi. Una spiegazione di questo è dato dal fatto che le regex delle varie closure sono fra loro indipendenti.

MANUALE UTILIZZO

L’utilizzo del programma è stato pensato in modo da essere assolutamente familiare agli utilizzatori di sistemi Unix-like, con particolare riferimento alla CLI.
Dunque, alcune informazioni che seguono potrebbero apparire superflue, ma vengono fornite per completezza.
Anzitutto, il main è contenuto all’interno del file __main__.py nella cartella FSM_algorithm. Eseguire questo file senza alcuna opzione porta in automatico a suggerimenti sull’utilizzo tramite questo output:

```
“usage: __main__.py [-h] -t {1,2} [--json JSON] [--bin BIN] -o
OUT_FILE [-O OBS_LIST] [-d] [-T MAX_TIME]
__main__.py: error: the following arguments are required: -t/--type,
-o/--out-file”.
```

Si evince che l’opzione ‘t’ e l’opzione ‘o’ siano obbligatorie: questo perché è necessario conoscere quale dei due metodi utilizzare e dove si vuole salvare l’output.

Si deduce anche che è presente un helper più completo, invocabile tramite '-h', di seguito il risultato:

```
"usage: __main__.py [-h] -t {1,2} [--json JSON] [--bin BIN] -o
OUT_FILE [-O OBS_LIST] [-d] [-T MAX_TIME]
A program to execute different computation on regex described with
finite state machines.
optional arguments:
  -h, --help          show this help message and exit
Command list:
  -t {1,2}, --type {1,2}
                      The task to accomplish. 1 - Compute the
Comportamental FA Network Space 2 - Compute the
                      CFANS relative to an observation
  --json JSON         File containing the ComportamentalFANetwork
  --bin BIN           File containing the binary structure
  -o OUT_FILE, --out-file OUT_FILE File to output results
  -O OBS_LIST, --obs-list OBS_LIST List of observations.
          Use -Oo1 -Oo2 if the observation is o1 followed by o2.
  -d, --diagnosis     State that the diagnosis must be computed
  -T MAX_TIME, --max-time MAX_TIME
          Maximum execution time in seconds".
```

Dunque, '-t 1' ci permette di eseguire tutti i compiti col primo metodo descritto (e dunque eventualmente utilizzare il diagnosticatore); '-t 2' utilizza l'altro metodo.
Le opzioni "--json" e "--bin" servono per distinguere fra un ingresso in formato json, che per forza deve essere la rete di FA comportamentali, e un binario; il binario deve essere un binario generato nelle computazioni precedenti, ed è il risultato della serializzazione dell'output (e.g. se si vuole computare lo spazio comportamentale si otterrà questo in formato binario e poi lo si potrà riutilizzare per svolgere le computazioni a partire da esso, quali il calcolo del diagnosticatore). Riassumendo --bin riutilizza i risultati delle computazioni precedenti, mentre --json svolge tutto sin dall'inizio. Gli oggetti sono stati serializzati con la libreria "pickle", dunque si è deciso di usare come estensione ".pkl".
L'opzione "-o" indica il path dove salvare l'output. Nel path descritto verrà salvato sia l'output "leggibile" con estensione json sia il binario in caso di riutilizzo. Si presti attenzione: il path è relativo al file json. Dunque un esempio potrebbe essere /tmp/prova.json. Si consiglia caldamente di inserire l'estensione. All'interno di /tmp/ verrà salvato anche prova.pkl.
L'opzione "-O", costituisce la lista delle osservazioni. Ha un uso decisamente meno intuitivo delle precedenti: ogni singola osservazione deve essere preceduta da "-O". Dunque se non si hanno osservazioni non si mette nulla, se ne sono presenti tre, in ordine [o1, o2, o2], la sintassi è la seguente: -Oo1 -Oo2 -Oo2. La ragione è spiegata nella documentazione della

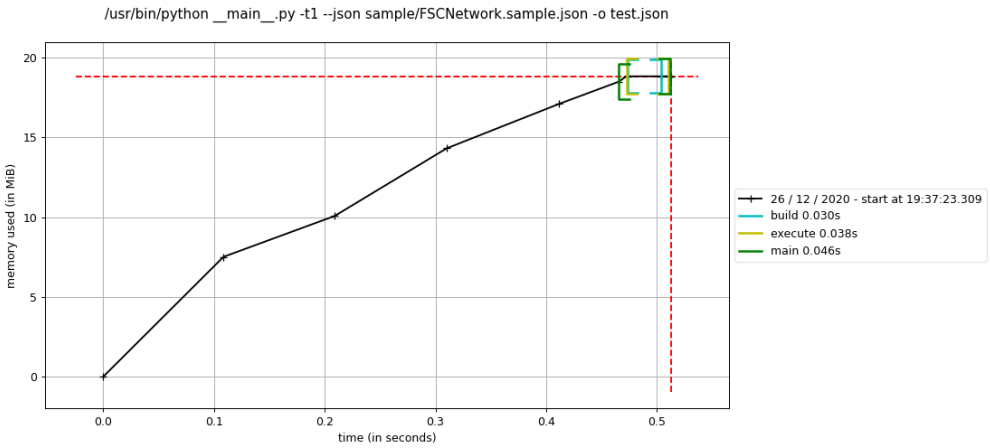
libreria “argparse”. Se si facesse -O o1 o2 o2 potrebbero esserci problemi con eventuali attributi posizionali (che comunque non sono presenti).
L’opzione “-T”, consente di dichiarare il tempo massimo di esecuzione in secondi, oltre il quale il programma in esecuzione verrà interrotto. Un output parziale verrà comunque salvato. Non ci sono garanzie sulla qualità di quest’ultimo.
L’opzione “-d” permette per i due metodi di eseguire la diagnosi/diagnosticatore dipendendo dal metodo dichiarato con “-t”. Va chiarito che il diagnosticatore può essere computato senza che venga dichiarata alcuna osservazione. Invece per la diagnosi col secondo metodo l’osservazione è obbligatoria.

Se si desidera un’interruzione prematura dei processi vi sono due metodi non mutuamente esclusivi: il primo è dichiarare il tempo massimo come spiegato in precedenza, il secondo è l’interruzione “manuale” inviando il segnale SIGINT al processo da CLI (tipicamente digitando “ctrl + c”). Entrambi salveranno in maniera persistente i risultati ottenuti. Interrompere chiudendo l’interprete da linea di comando non salverà nulla.

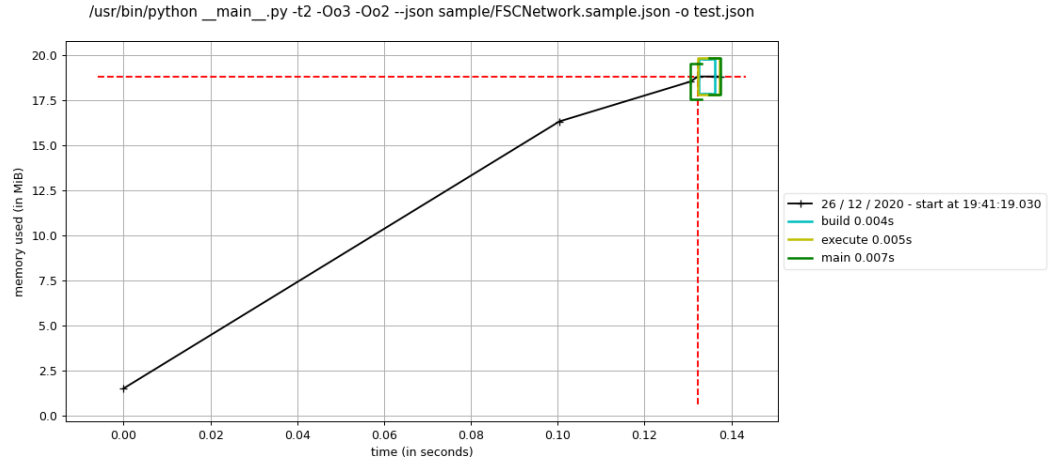
CONFRONTI SPAZIALI

Per i confronti spaziali è stata utilizzata la libreria python memory profiler. Questa libreria è decisamente di comune utilizzo per il linguaggio python. Per gli scopi proposti si è dimostrata sufficientemente precisa, a patto di sapere in che punti l’utilizzo di memoria può crescere. Nel nostro caso si tratta quasi sempre di metodi dal nome “build” i quali si occupano di costruire/riempire le strutture dati quali le liste, i dizionari o gli oggetti.

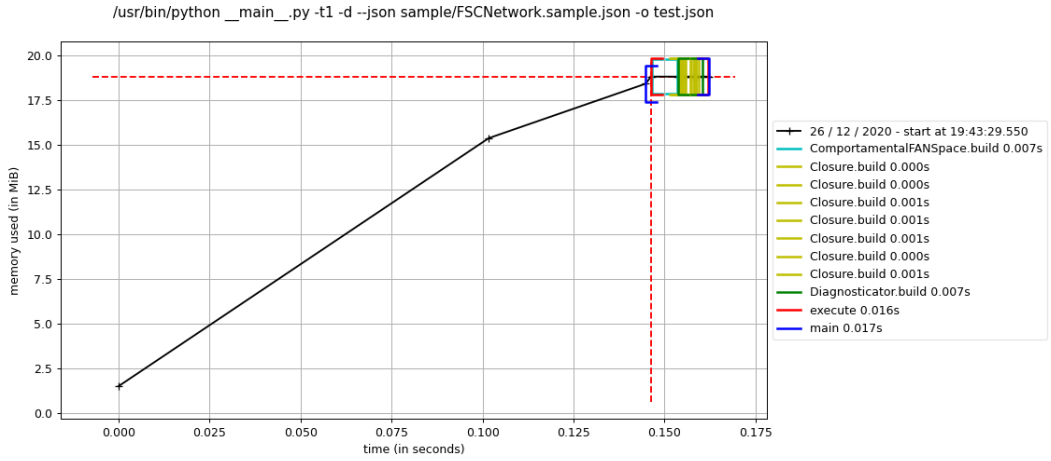
Di seguito verranno allegate delle immagini che rappresentano l’occupazione spaziale e temporale del programma rispetto ai vari task rispetto alla prima rete fornita come benchmark.



Per quanto riguarda il primo task, ovvero il calcolo dello spazio comportamentale degli automi viene evidenziato un utilizzo di 0.37 MB nel metodo build della classe CFANSpace in aggiunta allo spazio utilizzato dal linguaggio per istanziare l'environment.



Per quanto riguarda il secondo task, ovvero il calcolo dello spazio comportamentale degli automi rispetto ad una osservazione, non viene evidenziato un utilizzo maggiore della memoria in build CFANObservation in aggiunta dello spazio utilizzato dal linguaggio per istanziare l'environment.

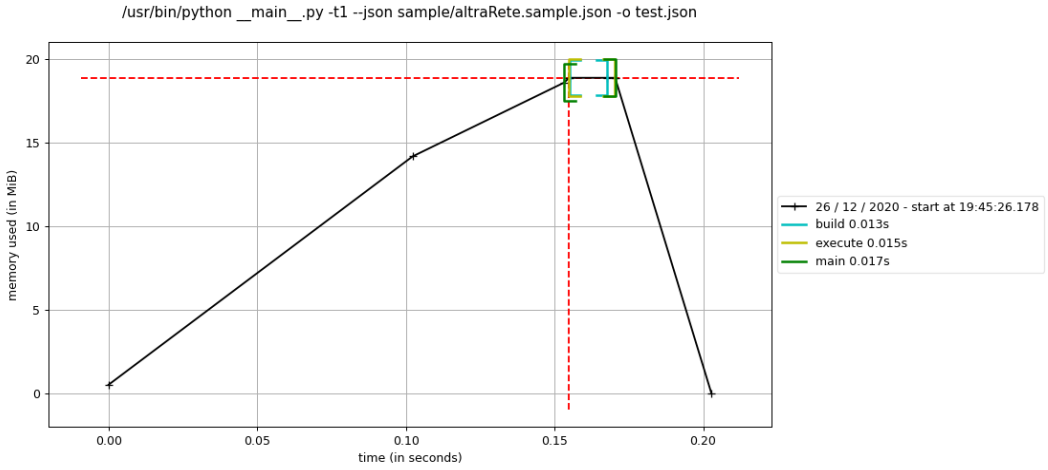


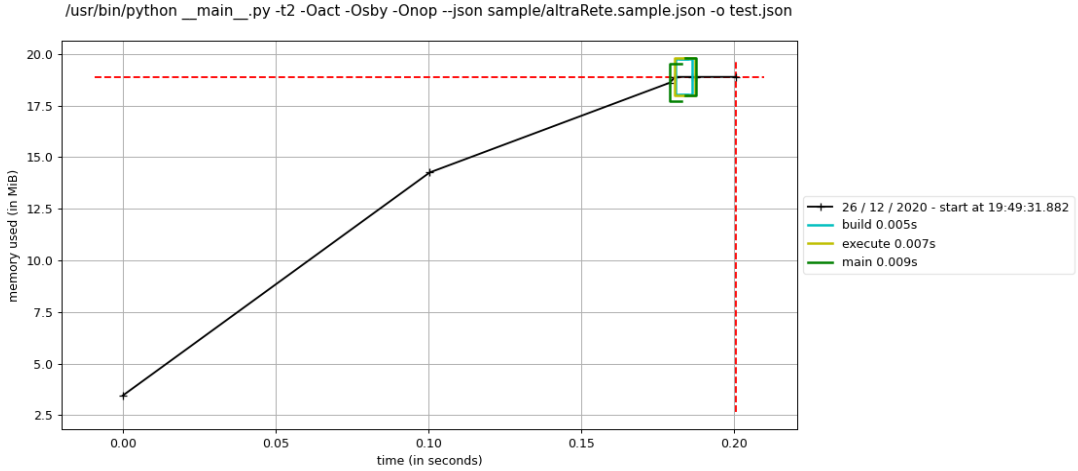
Per quanto riguarda l’ultimo test è stato eseguito il task 4, ovvero il calcolo delle chiusure; viene riscontrato un utilizzo simile a quello del primo task all’interno del build in CFANSpace e non viene evidenziato un utilizzo della memoria in build Closure, in aggiunta dello spazio utilizzato dal linguaggio per istanziare l'environment.

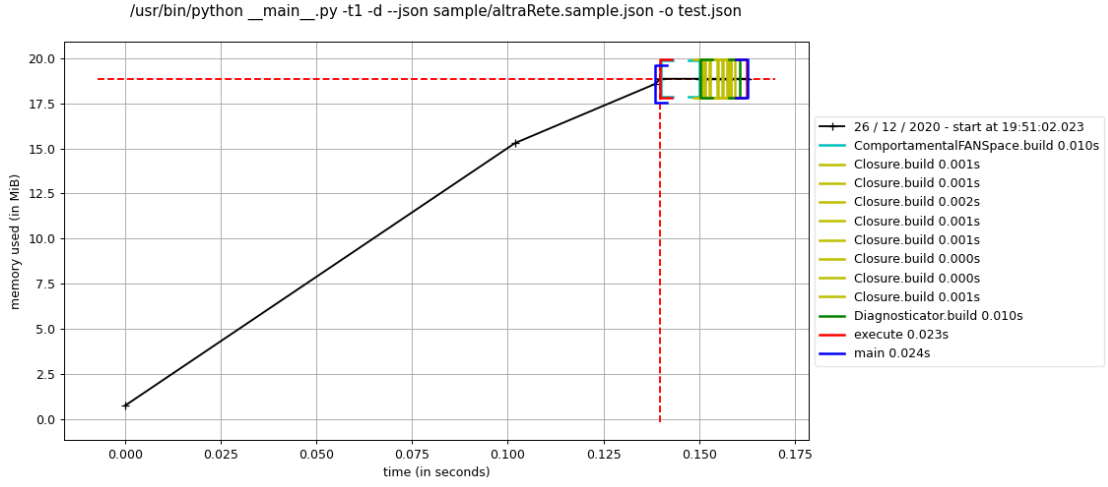
Vengono ora allegate le immagini relative agli ulteriori tre benchmark. Per visionare i valori numerici relativi all’effettivo uso della memoria relativi a tali immagini cliccare qui. Si anticipa qui quanto verrà approfondito nella sezione di commento relativa ai risultati: quanto osservato su questa rete e sulle successive non mostra un andamento generale ben chiaro dalle sole evidenze sperimentali che possa assumere valore di generalità. Un tentativo in tal direzione verrà comunque effettuato nella sezione di commento, integrando gli esperimenti con altre considerazioni.

ALTRA RETE

Di seguito vengono riportate le immagini relative all'utilizzo della memoria rispetto al benchmark "Altra Rete" rispetto ai task 1,2 e 4.

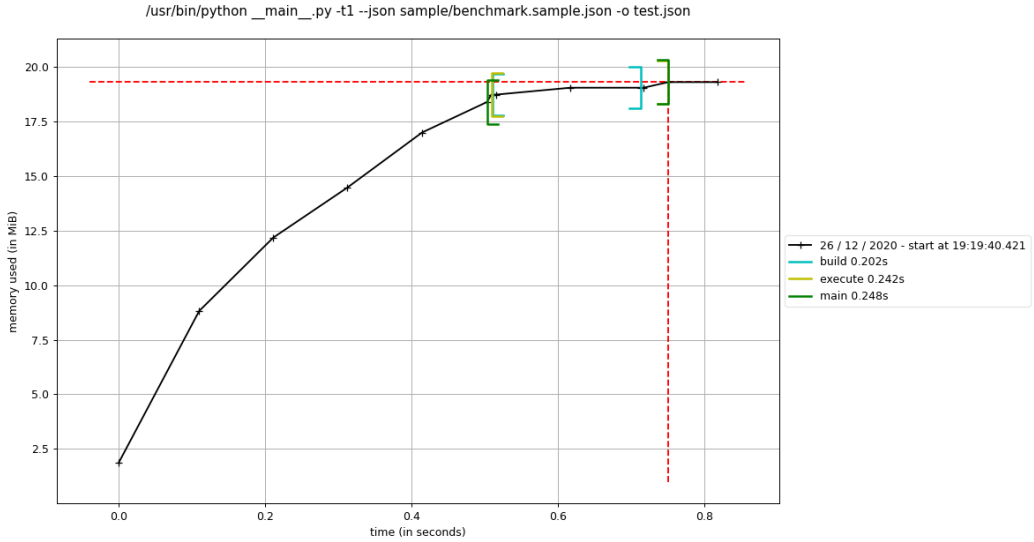


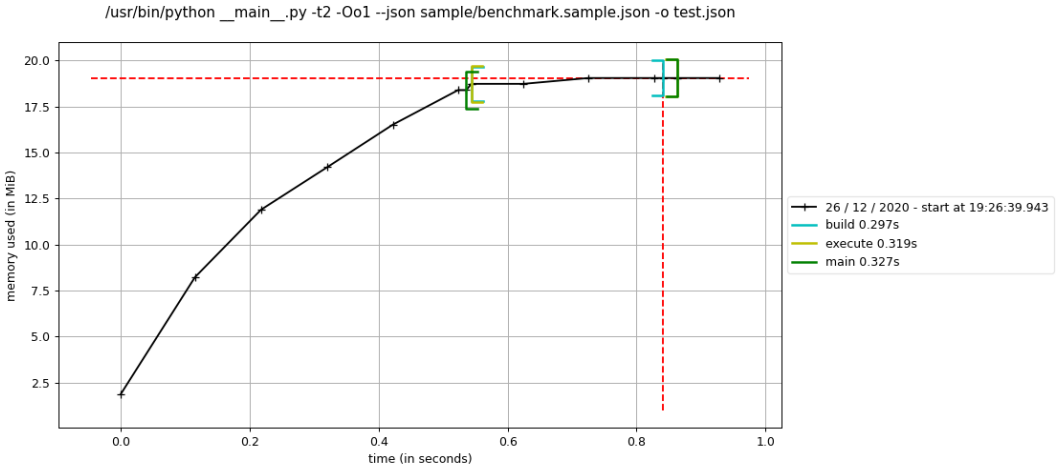


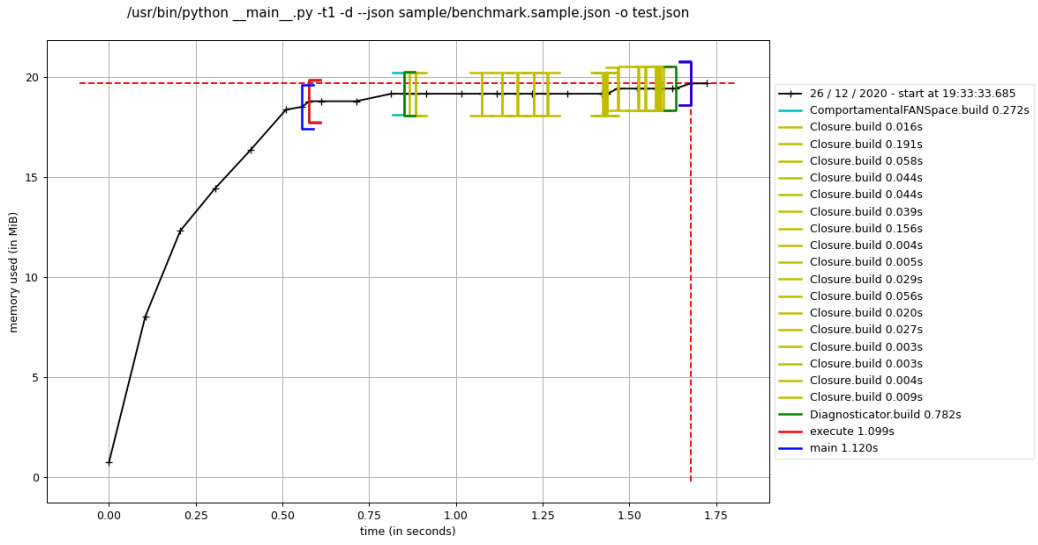


BENCHMARK

Di seguito vengono riportate le immagini relative all'utilizzo della memoria rispetto al benchmark "Benchmark" rispetto ai task 1,2 e 4.

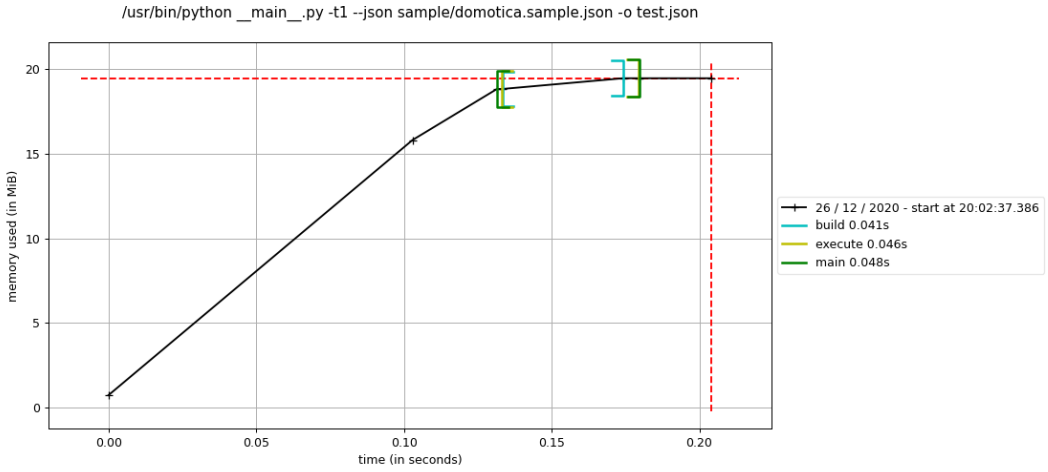


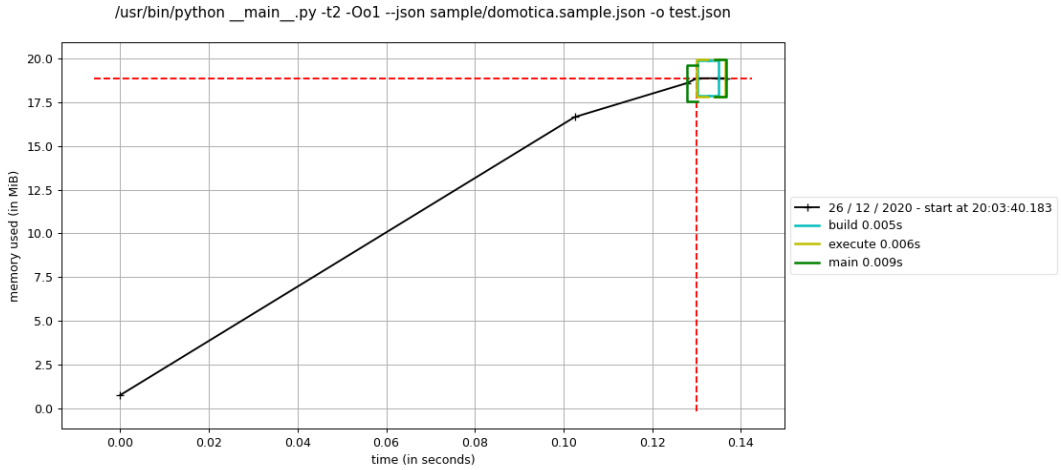


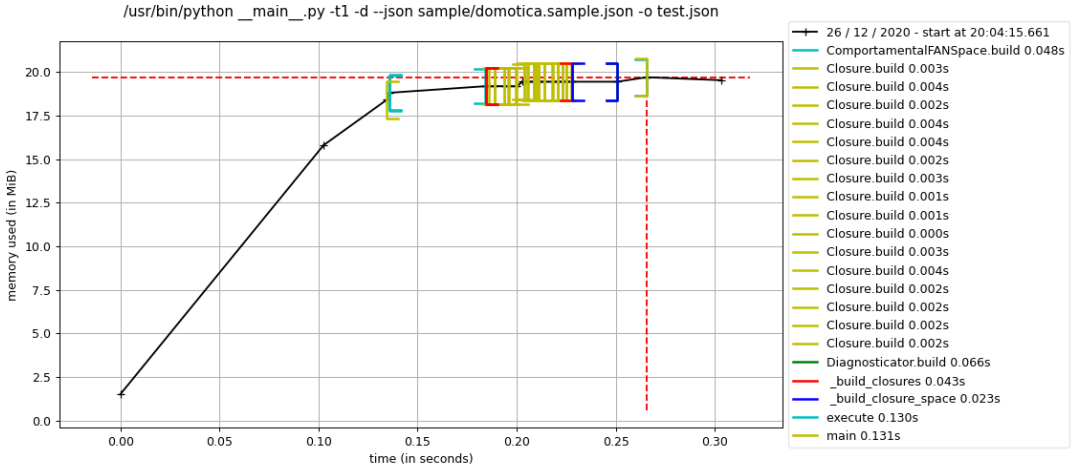


DOMOTICA

Di seguito vengono riportate le immagini relative all'utilizzo della memoria rispetto al benchmark "Domotica" rispetto ai task 1,2 e 4.







COMMENTI:

Come anticipato non è facile trarre una conclusione globalmente valida. Seguono alcune considerazioni che riteniamo valide al fine di meglio comprendere quando un metodo potrebbe essere preferito all'altro:

1. La dimensione della rete (e dello spazio delle chiusure) con il primo metodo è sempre uguale a prescindere dalle osservazioni;

2. La dimensione della rete con il secondo metodo varia al variare dell'osservazione cui è sottoposta;
    a. Ciò comporta in particolare che per osservazioni sufficientemente brevi, dipendendo dalla rete in questione, questo metodo costruirà reti arbitrariamente piccole;
    b. Ciò comporta in particolare che per osservazioni sufficientemente lunghe, dipendendo dalla presenza di cicli, questo metodo costruirà reti arbitrariamente grandi. Si consideri la prima rete presentata nella consegna: si può ripetere un numero arbitrario di volte la sequenza ['o3', 'o2'] per aumentare la dimensione dell'osservazione senza alcuna limitazione, asintoticamente fino a ∞;

3. Il primo metodo deve costruire anche lo spazio delle chiusure, il quale occupa certamente, a meno di casi limite, spazio in memoria addizionale;

4. Lo spazio occupato dal primo metodo durante la diagnosi lineare dipende dalla dimensione di χ. Questa può effettivamente crescere al crescere della dimensione dell'osservazione; ma si tratterebbe solo di un reference (e della regex associata che qui ignoriamo) a un oggetto già istanziato, dunque della dimensione necessaria per indirizzare la memoria; contrariamente, col secondo metodo, durante la creazione dello spazio relativo ad una osservazione, vanno creati degli oggetti nuovi ogni volta che si distinguono solo per l'indice dell'osservazione dai loro predecessori.

Dunque, ipotizzando che l'utente abbia conoscenza della rete che desidera analizzare:

- se lo spazio di memoria è ridotto e l'osservazione ragionevolmente breve data la dimensione e topologia della rete, sarebbe auspicabile utilizzare il secondo metodo in modo da tenere sotto controllo la dimensione della rete costruita, evitando di costruire stati "inutili" in quanto irraggiungibili date le osservazioni desiderate;

- se lo spazio di memoria è ridotto e l'osservazione pare lunga, in particolare se si intuisce che possa creare cicli particolarmente estesi rispetto alla dimensione originaria della rete, sarebbe auspicabile utilizzare il primo metodo, in quanto questa informazione non viene presa in considerazione fino alla diagnosi lineare, che come detto non produce un incremento paragonabile a quello della costruzione dello spazio comportamentale relativo a un'osservazione;
- Se lo spazio in memoria è sufficientemente grande si rimanda alla sezione riguardo il confronto temporale.

Concludendo, si può dire che il secondo metodo utilizza solo la memoria di cui necessita sempre; tuttavia, quest'ultima può essere molta di più di quella che può servire al primo metodo.
Il primo metodo può allocare memoria che non utilizzerà per molteplici osservazioni, e dunque allocata inutilmente; tuttavia, a prescindere dall'osservazione presa in considerazione, tale ammontare di memoria rimarrà pressoché costante.

Segue un caso patologico per cui il secondo metodo porta a un utilizzo di memoria che può divenire facilmente insostenibile:
sia dato lo stato 'As' (iniziale e finale) con una sola transizione in uscita che va in 'A1' (non finale) con osservazione 'O'; ora 'A1' va in 'A2' (non finale) senza osservazione, 'A2' in 'A3' (non finale) allo stesso modo e così fino a ad un certo 'An' (non finale), il quale rientra in 'As'. Ora, si può scegliere 'n' arbitrariamente lungo, ad esempio 1G. L'osservazione ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'] costituita da trentadue 'O' consumerebbe 32GB di RAM col secondo metodo, con l'ipotesi che ogni stato consumi un solo byte (B). Il primo metodo, con le stesse ipotesi si sarebbe limitato a 1GB.

Un esempio con un numero arbitrario di stati irraggiungibili data un'osservazione può mostrare le problematiche del primo metodo. Tuttavia, tutti gli stati irraggiungibili sarebbero presenti una e una sola volta, anche se in maniera superflua. Dunque è facile prevedere quanta memoria verrà occupata noti il numero di stati e lo spazio occupato da ogni stato in media.

CONFRONTI TEMPORALI

Per i confronti temporali è stato utilizzato il comando da terminale “time”.
Per la precisione time è un comando se eseguito con “bash”. Nel nostro caso è stato eseguito all’interno di “zsh”, e dunque è una keyword. A nostra conoscenza ciò non dovrebbe comportare alcuna differenza. È comunque reso noto al lettore per completezza delle informazioni.
Questo restituisce 3 valori:
user-time: ovvero il tempo in cui il processore è stato occupato in user-mode;
system-time: ovvero il tempo in cui il processore è stato occupato in kernel-mode;
total-time: il tempo passato dall’inizio dell’esecuzione fino alla fine di questa.
Il terzo valore verrà ignorato, in quanto non misura il tempo di utilizzo del processore ma il tempo trascorso dall’inizio alla fine del programma, ovvero include anche il tempo in cui altri processi sono stati eseguiti in caso di eventuale (in realtà certo) context-switch. Infatti, total-time varia molto fra diverse esecuzioni e ancor più dipendendo da cosa altro viene eseguito sulla stessa macchina (e.g. durante una chiamata remota tramite Discord/meet fra i membri costituenti del gruppo total-time varia molto).
Perciò il tempo di utilizzo della CPU verrà approssimato a user-time + system-time.
Anticipiamo che system-time è molto basso, quasi sempre 0-0.01s.
Nei risultati che seguono sarà calcolato sempre per primo il primo metodo, e per secondo il secondo.
Tutti i tempi sono tempi medi su 10 esecuzioni.
Tutti i test sono stati eseguiti sulla stessa macchina avente come processore:
Intel(R) Core(TM) i7-8565U CPU.

Risultati sulla **prima rete**:

- osservazione o3, o2:
    - 0.06s user 0.01s system;
    - 0.06s user 0.01s system;
- osservazione o3, o2, o3, o2:
    - 0.07s user 0.01s system;
    - 0.07s user 0.00s system;

Come era facilmente prevedibile, il test è troppo semplice per poter apprezzare differenze.

Risultati su **altra rete**:

- osservazione act, sby, nop:
    - 0.07s user 0.01s system;
    - 0.05s user 0.01s system;

Qui si può osservare che la seconda rete risulta più veloce, anche significativamente data la semplicità del compito. In questo caso particolare la media deriva da 20 prove.

Risultati su **benchmark**:

- osservazione o1:
    - 0.17s user 0.01s sysyem;
    - 0.06s user 0.01s system;

La differenza diviene netta: il primo metodo ci ha messo un tempo triplo rispetto alle reti più semplici, mentre il secondo è rimasto ancora pressoché uguale.

Risultati su **domotica**:

- osservazione o1:
    - 0.11s user 0.01s system;
    - 0.06s user 0.01s system;

Si conferma il risultato osservato su benchmark.

Nel complesso, dati questi esperimenti ci pare consono poter dire che il secondo metodo è più veloce. Non va sottovalutato quanto descritto nella sezione commenti riguardo l'utilizzo della memoria: infatti tutte le osservazioni sono molto brevi, e questo potrebbe aver favorito il secondo metodo. Tuttavia, all'incremento degli *stati* da visitare in caso di cicli per costruire la regex nel secondo metodo corrisponde un incremento delle *chiusure* durante "diagnosi lineare" nel primo. Per quanto una chiusura possa includere più stati, questo aspetto non ci sembra tanto gravoso come per la memoria. Tuttavia, non è nemmeno da sottovalutare: ipotizzando un caso patologico in cui una chiusura include milioni di stati facenti parte di un ciclo, visitare più volte milioni di stati e più volte una singola chiusura comporta certamente una differenza notevole a livello di tempi di utilizzo della CPU.

COMMENTI:

Seguono alcune valutazioni incentrate su un particolare utilizzo del primo metodo per renderlo comparabile, se non migliore, al secondo con un ampio grado di certezza.

Si prenda in considerazione una rete estremamente grande, con o senza cicli, sulla quale si desidera effettuare più osservazioni, o per la quale si prevede che si dovrà effettuare più osservazioni;
si ipotizzi di avere una macchina molto più potente della propria macchina locale (un server, un supercomputer, della potenza computazionale disponibile via cloud…), da qui in avanti chiamato server.
Ora, che questa potenza di calcolo venga utilizzata con o senza costi, può essere interessante evidenziare anzitutto un aspetto:
col secondo metodo, ogni calcolo verrà svolto ogni volta da zero al cambiare dell'osservazione, mentre col primo metodo si può calcolare una singola volta lo spazio delle chiusure, serializzarlo, e poi, per ogni osservazione riutilizzare lo spazio delle chiusure serializzato, computando dunque solo la diagnosi lineare; inoltre in questo modo si potrebbe anche dare al programma una matrice di osservazioni, nella quale ogni riga corrisponde a una esecuzione, in modo da dover caricare in memoria una sola volta l'oggetto serializzato riducendo al minimo le interazioni con lo storage e le systemcall (non previsto nell'implementazione corrente, ma facilmente implementabile); ancora oltre, si possono eseguire le diagnosi lineari in parallelo condividendo l'area di memoria in cui risiede lo spazio delle chiusure.
Se dunque una fase certamente onerosa, ovvero fino al calcolo dello spazio delle chiusure, venisse eseguita in remoto, la parte finale potrebbe essere eseguita da un qualunque thin-client in base all'osservazione desiderata data una chiusura già calcolata presente sul server, oppure in locale e comunicata al server; oppure sul server in parallelo come precedentemente ipotizzato.
Inoltre è da ipotizzarsi che per reti estremamente complesse un computer ordinario potrebbe non bastare con nessuno dei due metodi. Rifare il calcolo da zero, come fa il secondo metodo, può essere meno proficuo, nonostante la singola run sia più veloce (ipotizzando l'assenza di cicli patologici come già detto), rispetto al riutilizzo dei risultati già calcolati.
In questo modo, al crescere del numero di osservazioni il primo metodo si rivela sia più scalabile, sia più efficiente nel complesso. In un certo senso, con un certo abuso di terminologia, il primo metodo è migliore localmente (dove il neighborhood/vicinato è la singola osservazione, il problema completo l'insieme delle osservazioni).

Seguono gli stessi esempi precedenti per il primo metodo, con la chiusura già calcolata in anticipo:
Risultati su **Prima rete**:

- 0.05s user 0.02s system;
- 0.05s user 0.02s system;

Risultati su **Altra rete**:

- 0.06s user 0.01s system;

Risultati su **Benchmark**:

- 0.04s user 0.02s system;

Risultati su **Domotica**:

- 0.05s user 0.02s system;

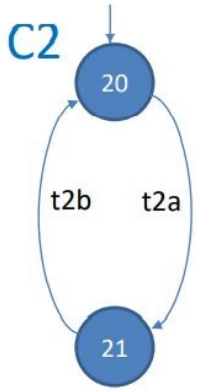
Mentre per “prima rete” e “altra rete” non si apprezza alcun miglioramento (se si considera anche il tempo di calcolo per la closure vi è anzi un peggioramento dovuto probabilmente alle operazioni di I/O), per “benchmark” e per “domotica” si osserva un netto miglioramento.

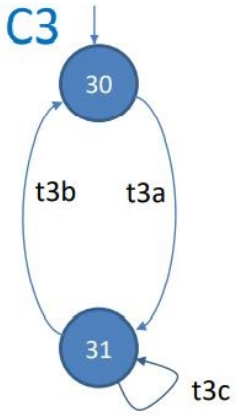
Se a queste considerazioni si aggiungono quelle presenti nei commenti riguardo la memoria, si può concludere che in generale il primo metodo sia a nostro avviso preferibile se non si ha modo di valutare criticamente la rete da eseguire, in quanto non richiede considerazioni sulla presenza di cicli, permette un elegante riutilizzo delle computazioni già effettuate e risulta più stabile in memoria.
La conclusione non è comunque assoluta: se si è certi di non dover eseguire altre osservazioni sulla medesima rete, o comunque poche, e non ci sono cicli creati dalle osservazioni il secondo metodo si dovrebbe comportare ragionevolmente meglio.

Concludendo, assumendo che il primo metodo non abbia complessità esponenziale (come anche il secondo), l’utilizzo del primo metodo con precomputazioni, ancora meglio se remote, può essere vantaggioso.

BENCHMARK

DUE FA COMPORTAMENTALI



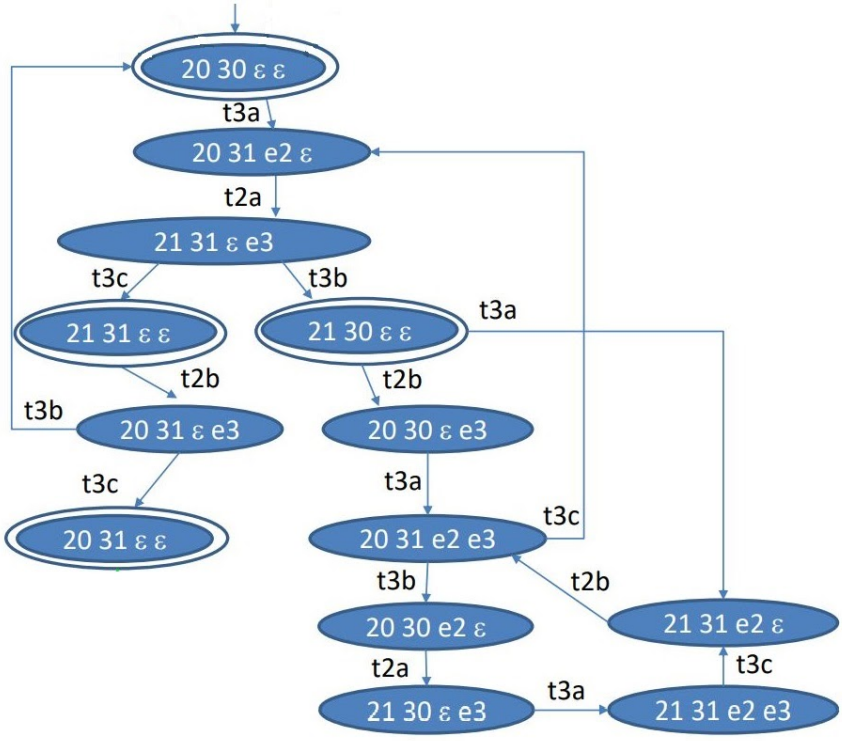


TRANSIZIONI C2
t2a: e2(L2) / {e3(L3)}
t2b: / {e3(L3)}

TRANSIZIONI C3
t3a: / {e2(L2)}
t2b: e3(L3)
t3c: e3(L3)

	OSSERVABILITÀ	RILEVANZA
t2a:	o2	***ε***
t2b:	***ε***	r
t3a:	o3	***ε***
t3b:	***ε***	***ε***
t3c:	***ε***	f

Calcolo relativo spazio comportamentale



Dati i due FA comportamentali precedenti al programma, il relativo spazio comportamentale viene computato in maniera identica a quello fornito di confronto.

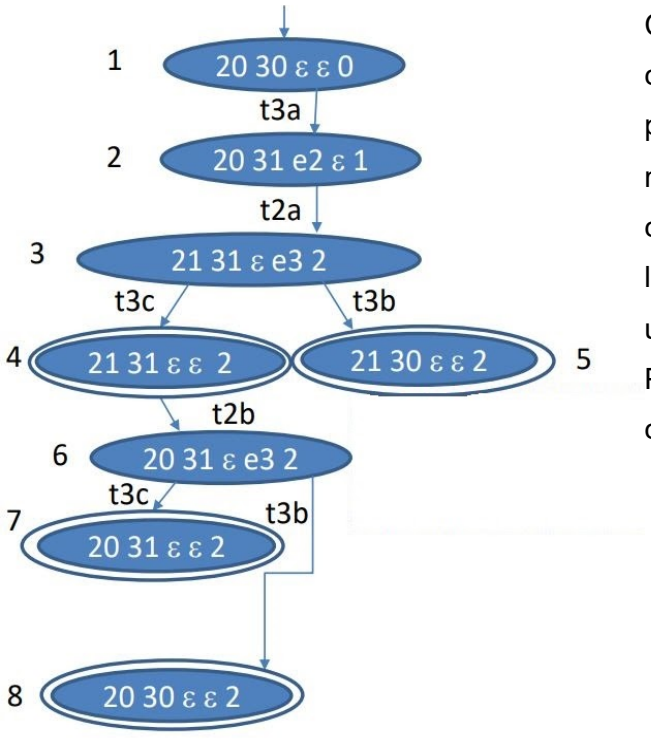
A sinistra viene rappresentato lo spazio comportamentale mentre di seguito verrà riportata una parte dell'output. Per poterlo visionare in maniera completa cliccare qua.

```
{
    "number comportamental FA": 2,
    "number states": 4,
    "number space states": 13,
    "number transitions": 16,
    "space_state": [
        {
            "name": 0,
            "link": {
                "L2": "\u03b5",
                "L3": "\u03b5"
            },
            "state": {
                "CFAN 0": "20",
                "CFAN 1": "30"
            },
            "next": {
                "t3a": {
                    "state": {
                        "CFAN 0": "20",
                        "CFAN 1": "31"
                    },
                    "link": {
                        "L2": "e2",
                        "L3": "\u03b5"
                    },
                    "observable": "o3",
                    "relevant": null
                }
            }
        },
```


L'output è rappresentato da un file "json" così costituito:

- una parte iniziale costituita dalle proprietà number comportamental FA (rappresenta il numero dei FA), il numero dei possibili stati dei FA, il numero effettivo degli stati dello spazio comportamentale ed il numero di transazioni totali. Si fa presente che tale output è quello finale, ovvero a seguito di un possibile pruning di eventuali stati e transizioni pendenti.
- l'effettivo space state al cui interno saranno presenti i relativi stati così descritti:
    - name (costituito da un numero univoco crescente);
    - link (lo stato dei link in quello stato)
    - gli stati degli Fa;
    - gli stati successivi e la transizione che li collega.

Costruzione dello spazio comportamentale relativo a un'osservazione



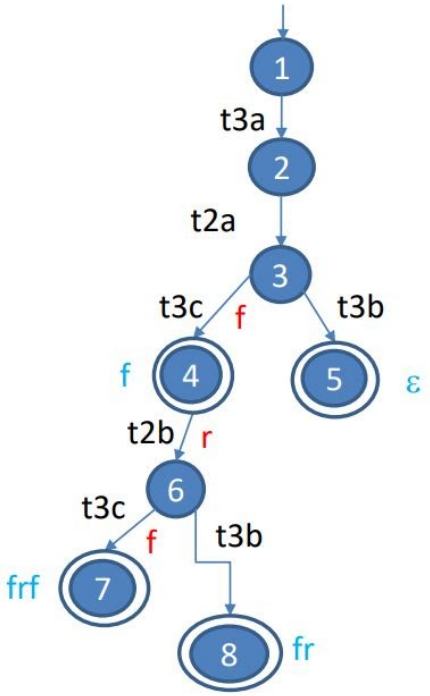
Come nel caso precedente il risultato fornito come confronto e l'output fornito del programma sono equiparabili. A sinistra la rappresentazione dello spazio comportamentale relativo all'osservazione lineare [O3, O2]. Di seguito viene riportato una porzione dell'output del programma. Per poterla visionare in maniera completa cliccare qua.

```
{
    "observation": [
        "o3",
        "o2"
    ],
    "number comportamental FA": 2,
    "number states": 4,
    "number space states": 8,
    "number transitions": 7,
    "space_state_linear_observation": [
        {
            "name": 3,
            "obs_index": 2,
            "link": {
                "L3": "\u03b5",
                "L2": "\u03b5"
            },
            "state": {
                "CFAN 0": "21",
                "CFAN 1": "30"
            },
            "next": {}
        },
        {
            "name": 2,
            "obs_index": 2,
            "link": {
                "L3": "e3",
                "L2": "\u03b5"
            },
            "state": {
                "CFAN 0": "21",
                "CFAN 1": "31"
```


L'output è rappresentato da un file "json" così costituito:

- una parte iniziale costituita dall'osservazione che è stata utilizzata per calcolare il relativo spazio comportamentale, dalle proprietà number comportamental FA (rappresenta il numero dei FA), il numero dei possibili stati dei FA, il numero effettivo degli stati dello spazio comportamentale ed il numero di transazioni totali. Si fa presente che tale output è quello finale, ovvero a seguito di un possibile pruning di eventuali stati e transizioni pendenti.
- l'effettivo space state al cui interno saranno presenti i relativi stati descritto come nel punto precedente con la sola aggiunta dell'attributo "obs_index", che rappresenta il numero di osservazioni dell'input visitate fino al relativo stato.

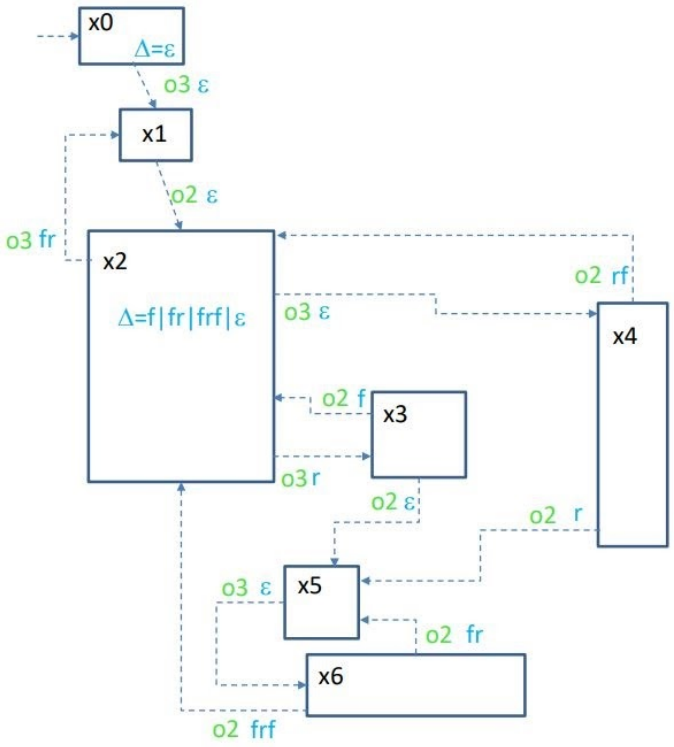
Diagnosi relativa a una osservazione lineare



Il terzo compito consiste nel computare una diagnosi rispetto ad una espressione regolare. L'output viene direttamente stampato a video una volta effettuato il calcolo dal programma.
Di seguito uno screen del risultato rispetto all'osservazione O = [O3, O2]

```
CFANS respect observation ['o3', 'o2'] complete
Start computing diagnosis
Diagnosis complete (ε|f(ε|r(ε|f)))
```

Diagnosticatore di una rete di FA comportamentali



Il successivo compito consiste nel produrre il diagnosticatore relativo ad una rete di FA comportamentali. A sinistra è presente la rappresentazione grafica del diagnosticatore mentre successivamente verrà riportata una porzione dell'output fornito dal programma. Per poterlo visionare in maniera completa cliccare qua.


```
{
    "number space states": 13,
    "number closures": 7,
    "number transactions": 12,
    "closure": [
        {
            "name": "x0",
            "in_state_id": 0,
            "regex": "\u03b5",
            "exit": {
                "o3": [
                    {
                        "successor": "x1",
                        "trns_regex": "\u03b5"
                    }
                ]
            }
        },
        {
            "name": "x1",
            "in_state_id": 1,
            "regex": "\u03b5",
            "exit": {
                "o2": [
                    {
                        "successor": "x3",
                        "trns_regex": "\u03b5"
                    }
                ]
            }
        },
        {
            "name": "x2",
            "in_state_id": 6,
            "regex": "\u03b5",
            "exit": {
```


L'output è rappresentato da un file "json" così costituito:

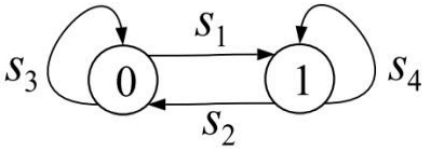
- una parte iniziale costituita dalle proprietà number comportamental FA (rappresenta il numero dei FA) come il numero effettivo degli stati dello spazio comportamentale, il numero di chiusure ed il numero di transazioni fra le chiusure.
- l'effettivo diagnosticatore al cui interno saranno presenti le chiusure così descritte:
    - name (costituito da un numero univoco crescente);
    - in_state_id che rappresenta l'id dello stato iniziale della chiusura
    - la regex associata alla chiusura
    - le transizioni uscenti con le seguenti informazioni:
        - osservazione
        - regex associata
        - stato successivo

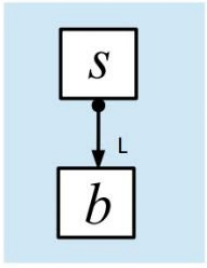
Calcolo diagnosi relativa ad una osservazione lineare

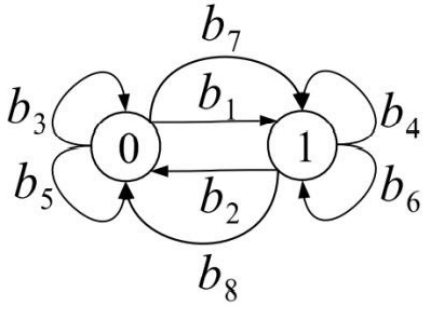
L' ultimo compito consiste nel produrre una diagnosi relativa ad una osservazione lineare. L'output viene direttamente stampato a video una volta effettuato il calcolo dal programma. Di seguito uno screen del risultato rispetto all'osservazione O = [O3, O2, O3, O2]

```
Start evaluate diagnosis respect observation ['o3', 'o2', 'o3', 'o2']
(rf|fr)(ε|f|fr|frf)
```

ALTRA RETE

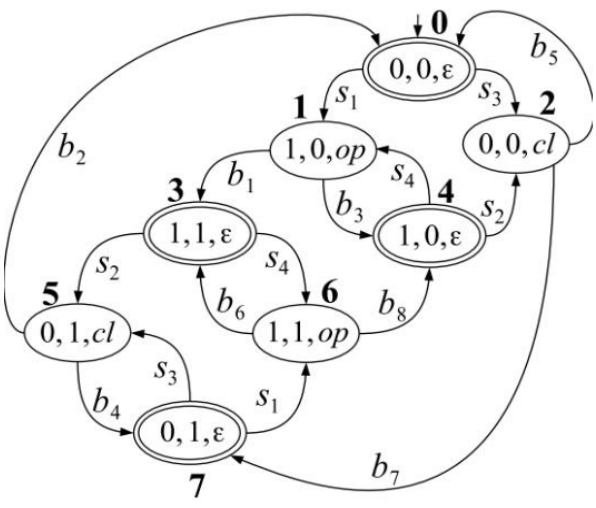






TRANSIZIONI	OSSERVAZIONI	RILEVANZA
s1:/{op(L)}	act	***ε***
s2:/{cl(L)}	sby	***ε***
s3:/{cl(L)}	***ε***	f1
s4:/{op(L)}	***ε***	f2
b1: op(L)	opn	***ε***
b2: cl(L)	cls	***ε***
b3: op(L)	***ε***	f3
b4: cl(L)	***ε***	f4
b5: cl(L)	nop	***ε***
b6: op(L)	nop	***ε***
b7: cl(L)	opn	f5
b8: op(L)	cls	f6

Calcolo relativo spazio comportamentale

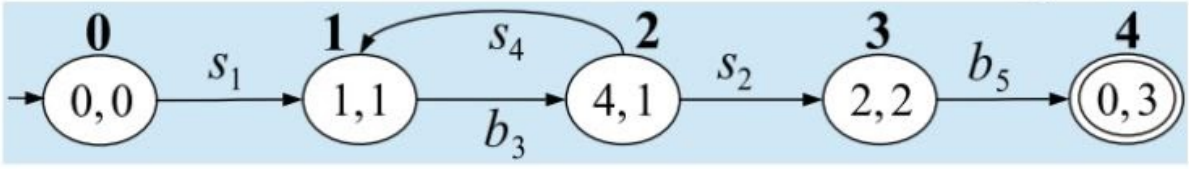


Di seguito viene presentato una porzione dell'output relativo a tale task. Per poterlo visionare in maniera completa cliccare qua.


```
{
    "number comportamental FA": 2,
    "number states": 4,
    "number space states": 8,
    "number transitions": 16,
    "space_state": [
        {
            "name": 0,
            "link": {
                "L": "\u03b5"
            },
            "state": {
                "CFAN 0": "0",
                "CFAN 1": "0"
            },
            "next": {
                "S1": {
                    "state": {
                        "CFAN 0": "1",
                        "CFAN 1": "0"
                    },
                    "link": {
                        "L": "op"
                    },
                    "observable": "act",
                    "relevant": null
                },
                "S3": {
                    "state": {
                        "CFAN 0": "0",
                        "CFAN 1": "1"
                    },
                    "link": {
                        "L": "cl"
                    },
                    "observable": null,
                    "relevant": "f1"
                }
            }
        },
```


Costruzione dello spazio comportamentale relativo a un'osservazione

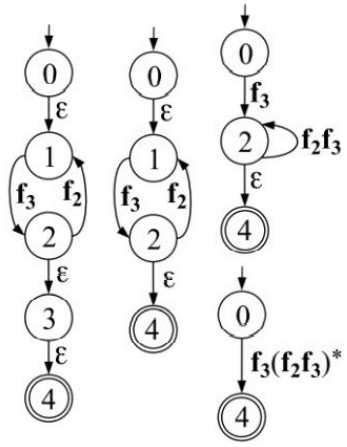
Di seguito viene riportata la rappresentazione grafica dello spazio comportamentale relativo all'osservazione lineare O = [act, sby, nop].



Di seguito viene presentato una porzione dell'output relativo a tale task. Per poterlo visionare in maniera completa cliccare qua.


```
{
    "observation": [
        "act",
        "sby",
        "nop"
    ],
    "number comportamental FA": 2,
    "number states": 4,
    "number space states": 5,
    "number transitions": 5,
    "space_state_linear_observation": [
        {
            "name": 4,
            "obs_index": 3,
            "link": {
                "L": "\u03b5"
            },
            "state": {
                "CFAN 0": "0",
                "CFAN 1": "0"
            },
            "next": {}
        },
        {
            "name": 3,
            "obs_index": 2,
            "link": {
                "L": "cl"
            },
            "state": {
                "CFAN 0": "0",
                "CFAN 1": "0"
            },
            "next": {
                "b5": {
                    "state": {
                        "CFAN 0": "0",
                        "CFAN 1": "0"
                    },
                    "link": {
                        "L": "\u03b5"
                    }
                }
            }
        },
```

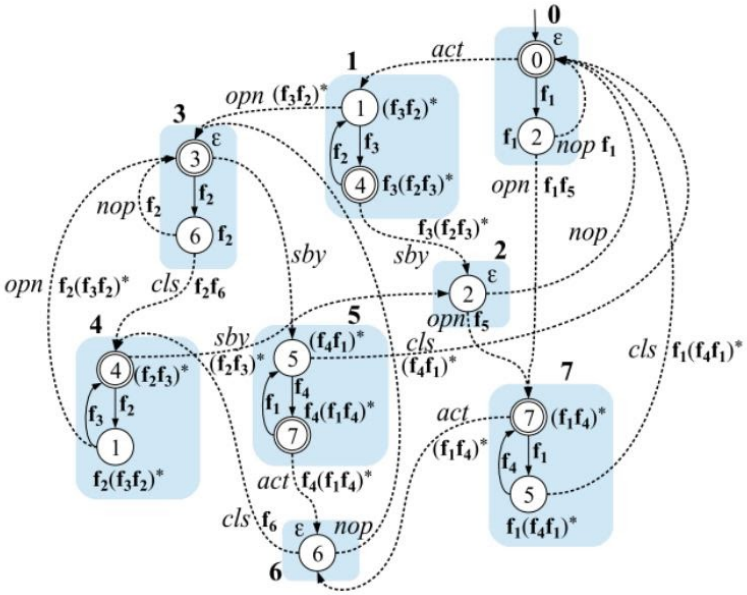
Diagnosi relativa a una osservazione lineare



Di seguito uno screen del risultato rispetto all'osservazione O = [act, sby, nop].

```
CFANS respect observation ['act', 'sby', 'nop'] complete
Start computing diagnosis
Diagnosis complete (f3f2)*f3
```

Diagnosticatore di una rete di FA comportamentali



Di seguito viene presentato una porzione dell'output relativo a tale task. Per poterlo visionare in maniera completa cliccare qua.

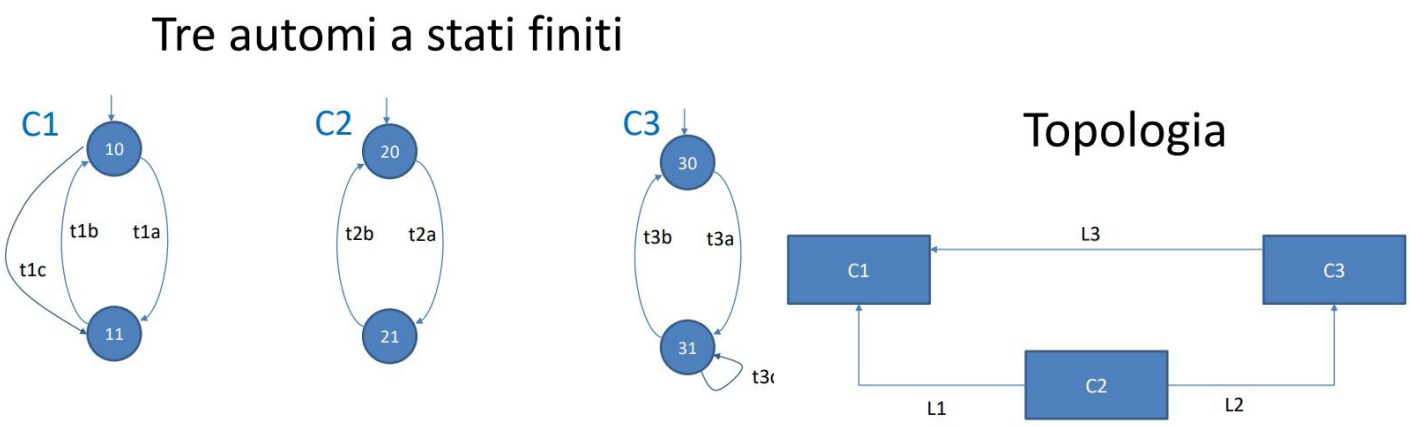

```
{
    "number space states": 8,
    "number closures": 8,
    "number transactions": 18,
    "closure": [
        {
            "name": "x0",
            "in_state_id": 0,
            "regex": "\u03b5",
            "exit": {
                "act": [
                    {
                        "successor": "x1",
                        "trns_regex": "\u03b5"
                    }
                ],
                "nop": [
                    {
                        "successor": "x0",
                        "trns_regex": "f1"
                    }
                ],
                "opn": [
                    {
                        "successor": "x4",
                        "trns_regex": "f1f5"
                    }
                ]
            }
        },
        {
            "name": "x1",
            "in_state_id": 1,
            "regex": "f3(f2f3)*",
            "exit": {
                "opn": [
                    {
                        "successor": "x3",
                        "trns_regex": "\u03b5|f3(f2f3)*f2"
                    }
                ],
                "sby": [
                    {
                        "successor": "x5",
                        "trns_regex": "f3(f2f3)*"
                    }
                ]
            }
        },
```

Calcolo diagnosi relativa ad una osservazione lineare

Di seguito uno screen del risultato rispetto all'osservazione O = [act, sby, nop]

```
Start evaluate diagnosis respect observation ['act', 'sby', 'nop']
(f3(f2f3)*)(ε)
```

BENCHMARK



TRANSIZIONI	OSSERVAZIONI	RILEVANZA
t1a: e1(L1)	ε	ε
t1b: e2(L3)	ε	ε
t1c:	ε	f1
t2a: /{e1(L1), e3(L2)}	o1	ε
t2b: /{e1(L1)}	o2	ε
t3a: /{e2(L3)}	ε	ε
t3b: e3(L2)	ε	ε
t3c: e3(L2)	ε	f3

Calcolo relativo spazio comportamentale

Di seguito viene presentato una porzione dell'output relativo a tale task. Per poterlo visionare in maniera completa cliccare qua.


```
{
    "number comportamental FA": 3,
    "number states": 6,
    "number space states": 41,
    "number transitions": 85,
    "space_state": [
        {
            "name": 0,
            "link": {
                "L2": "\u03b5",
                "L1": "\u03b5",
                "L3": "\u03b5"
            },
            "state": {
                "CFAN 0": "10",
                "CFAN 1": "20",
                "CFAN 2": "30"
            },
            "next": {
                "T1C": {
                    "state": {
                        "CFAN 0": "11",
                        "CFAN 1": "20",
                        "CFAN 2": "30"
                    },
                    "link": {
                        "L2": "\u03b5",
                        "L1": "\u03b5",
                        "L3": "\u03b5"
                    },
                    "observable": null,
                    "relevant": "f1"
                },
                "T2A": {
                    "state": {
                        "CFAN 0": "10",
                        "CFAN 1": "21",
                        "CFAN 2": "30"
                    },
                    "link": {
                        "L2": "e3",
                        "L1": "e1",
                        "L3": "\u03b5"
                    },
                    "observable": "o1",
                    "relevant": null
                },
```


Costruzione dello spazio comportamentale relativo a un'osservazione

Di seguito viene riportata la rappresentazione grafica dello spazio comportamentale relativo all'osservazione lineare O = [o1, o2]. Di seguito viene presentato una porzione dell'output relativo a tale task. Per poterlo visionare in maniera completa cliccare qua.


```
{
    "observation": [
        "o1",
        "o2"
    ],
    "number comportamental FA": 3,
    "number states": 6,
    "number space states": 39,
    "number transitions": 75,
    "space_state_linear_observation": [
        {
            "name": 10,
            "obs_index": 2,
            "link": {
                "L1": "\u03b5",
                "L3": "\u03b5",
                "L2": "\u03b5"
            },
            "state": {
                "CFAN 0": "11",
                "CFAN 1": "20",
                "CFAN 2": "31"
            },
            "next": {}
        },
        {
            "name": 9,
            "obs_index": 2,
            "link": {
                "L1": "e1",
                "L3": "\u03b5",
                "L2": "\u03b5"
            },
            "state": {
                "CFAN 0": "10",
                "CFAN 1": "20",
                "CFAN 2": "31"
            },
            "next": {
                "T1A": {
                    "state": {
                        "CFAN 0": "11",
                        "CFAN 1": "20",
                        "CFAN 2": "31"
                    },
                    "link": {
                        "L1": "\u03b5",
                        "L3": "\u03b5",
                        "L2": "\u03b5"
                    }
```

Diagnosi relativa a una osservazione lineare

Di seguito uno screen del risultato rispetto all’osservazione O = [o1, o2].

```
CFANS respect observation ['o1', 'o2'] complete
Start computing diagnosis
Diagnosis complete ((((f1(((((ε|ε)|(ε|ε))|(ε|(ε|ε)))|(ε|ε))|((ε|(ε|ε))|(ε|ε)))|f1((((((ε|ε)|(ε|ε))|(ε|(ε|ε)))|(ε|ε))|((((ε|ε)|
(ε|ε))|(ε|(ε|ε)))|(ε|ε)))|((ε|(ε|ε))|(ε|ε))))|(((f1((((ε|ε)|(ε|ε))|(ε|(ε|ε)))|(ε|ε))|((ε|(ε|ε))|(ε|ε)))|((((f3|((f3|f1)|((f3|
ε)|(ε|f1))))|((f1|ε)|(ε|f1)))|(f1|(ε|f1)))|(((f1|ε)|(ε|f1))|(f1|(ε|f1)))))|((((f3(ε|ε)|(((f3|((f3|f1)|((f3|ε)|(ε|f1))))|((f1|ε
)|(ε|f1)))|(f1|(ε|f1))))|(((f1|ε)|(ε|f1))|(f1|(ε|f1))))|((((((f1|f1)|f3)|f1(ε|ε))|((f3|f1)|((f3|ε)|(ε|f1))))|((f1|ε)|(ε|f1)))|
(f1|(ε|f1))))|((((f1|f1(ε|ε))|((f1|ε)|(ε|f1)))|(f1|(ε|f1)))|(f1(ε|ε)|(f1|(ε|f1))))))|((((f1|ε)|(ε|f1))|(f1|(ε|f1)))|((((f1|f1(
ε|ε))|((f1|ε)|(ε|f1)))|(f1|(ε|f1)))|(f1(ε|ε)|(f1|(ε|f1)))))))|(((((f3(ε|ε)|f1(((ε|ε)|(ε|ε))|(ε|(ε|ε)))|(ε|ε)))|f1((ε|(ε|ε))|(
ε|ε)))|((((f3(ε|ε)|(((f3|((f3|f1)|((f3|ε)|(ε|f1))))|((f1|ε)|(ε|f1)))|(f1|(ε|f1))))|(((f1|ε)|(ε|f1))|(f1|(ε|f1))))|((((((f1|f1)
|f3)|f1(ε|ε))|((f3|f1)|((f3|ε)|(ε|f1))))|((f1|ε)|(ε|f1)))|(f1|(ε|f1))))|((((f1|f1(ε|ε))|((f1|ε)|(ε|f1)))|(f1|(ε|f1)))|(f1(ε|ε)
|(f1|(ε|f1))))))|((((f1|ε)|(ε|f1))|(f1|(ε|f1)))|((((f1|f1(ε|ε))|((f1|ε)|(ε|f1)))|(f1|(ε|f1)))|(f1(ε|ε)|(f1|(ε|f1))))))|(f1((ε|
(ε|ε))|(ε|ε))|((((f1|ε)|(ε|f1))|(f1|(ε|f1)))|((((f1|f1(ε|ε))|((f1|ε)|(ε|f1)))|(f1|(ε|f1)))|(f1(ε|ε)|(f1|(ε|f1)))))))|(f1((((
(ε|ε)|(ε|ε))|(ε|(ε|ε)))|(ε|ε))|((((ε|ε)|(ε|ε))|(ε|(ε|ε)))|(ε|ε)))|((ε|(ε|ε))|(ε|ε)))|(((((f3(ε|ε)|f1(((ε|ε)|(ε|ε))|(ε|(ε|ε)))
|(ε|ε)))|f1((ε|(ε|ε))|(ε|ε)))|((((f3(ε|ε)|(((f3|((f3|f1)|((f3|ε)|(ε|f1))))|((f1|ε)|(ε|f1)))|(f1|(ε|f1))))|(((f1|ε)|(ε|f1))|(f1
|(ε|f1))))|((((((f1|f1)|f3)|f1(ε|ε))|((f3|f1)|((f3|ε)|(ε|f1))))|((f1|ε)|(ε|f1)))|(f1|(ε|f1))))|((((f1|f1(ε|ε))|((f1|ε)|(ε|f1))
)|(f1|(ε|f1)))|(f1(ε|ε)|(f1|(ε|f1))))))|((((f1|ε)|(ε|f1))|(f1|(ε|f1)))|((((f1|f1(ε|ε))|((f1|ε)|(ε|f1)))|(f1|(ε|f1)))|(f1(ε|ε)|
(f1|(ε|f1))))))|(f1((ε|(ε|ε))|(ε|ε))|((((f1|ε)|(ε|f1))|(f1|(ε|f1)))|((((f1|f1(ε|ε))|((f1|ε)|(ε|f1)))|(f1|(ε|f1)))|(f1(ε|ε)|(f1
|(ε|f1))))))))
```

Diagnosticatore di una rete di FA comportamentali

Di seguito viene presentato una porzione dell’output relativo a tale task. Per poterlo visionare in maniera completa cliccare qua.


```
{
    "number space states": 41,
    "number closures": 17,
    "number transactions": 64,
    "closure": [
        {
            "name": "x0",
            "in_state_id": 0,
            "regex": "\u03b5|f1|(f1|f1)|(f1|f1)f1",
            "exit": {
                "o1": [
                    {
                        "successor": "x1",
                        "trns_regex": "\u03b5"
                    },
                    {
                        "successor": "x2",
                        "trns_regex": "f1"
                    },
                    {
                        "successor": "x6",
                        "trns_regex": "\u03b5"
                    },
                    {
                        "successor": "x5",
                        "trns_regex": "(f1|f1)"
                    },
                    {
                        "successor": "x4",
                        "trns_regex": "(f1|f1)"
                    },
                    {
                        "successor": "x7",
                        "trns_regex": "(f1|f1)f1"
                    }
                ]
            }
        },
        {
            "name": "x1",
            "in_state_id": 2,
            "regex": "((\u03b5|\u03b5)|((\u03b5|\u03b5)|\u03b5))|((((\u03b5|\u03b5)f1|(f1|f1))|
                    ((\u03b5|\u03b5)|((\u03b5|\u03b5)|\u03b5))f1)|(((f1|f1)|f1)|(f1|f1)))|((((\u03b5

            "exit": {
                "o2": [
                    {
                        "successor": "x3",
                        "trns_regex": "\u03b5"
                    },
```

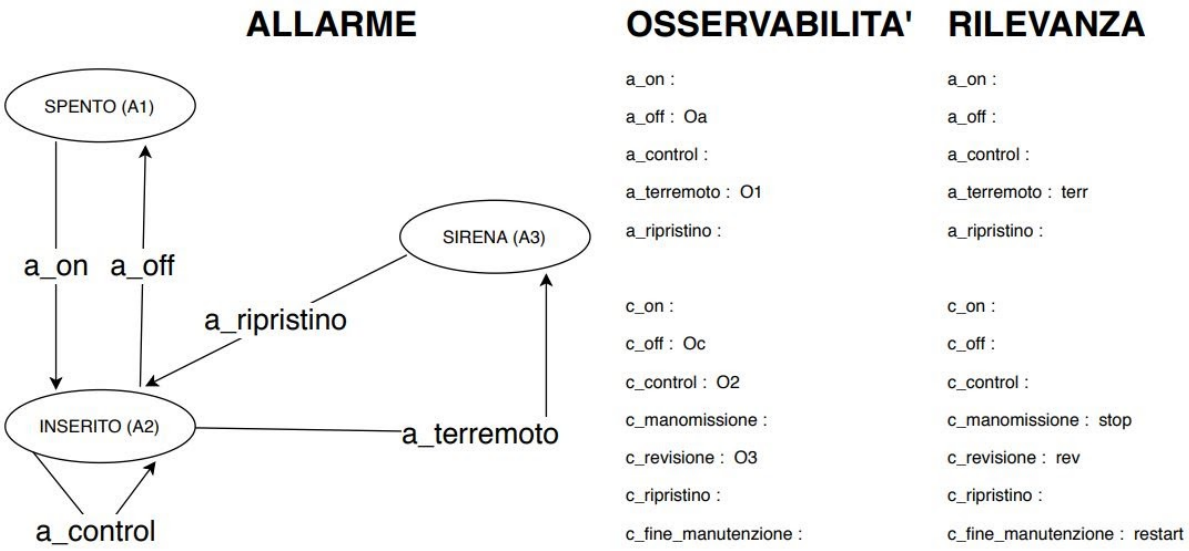

Calcolo diagnosi relativa ad una osservazione lineare

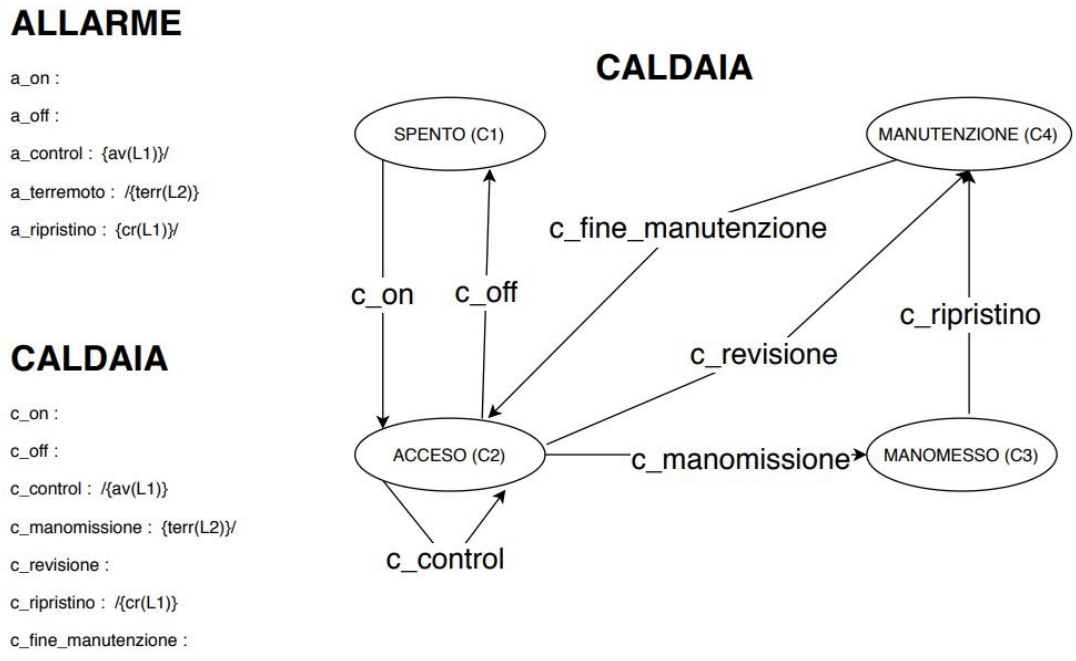
Di seguito uno screen del risultato rispetto all’osservazione O = [o1, o2]

```
Start evaluate diagnosis respect observation ['o1', 'o2']
(ε((ε|(ε|ε))|((f3|((f3|f3)|(f1|(ε|ε)f1)))|(ε|ε)f1)|((ε|(ε|ε))|(ε|ε))|((ε|(ε|ε))|(ε|ε))f1))|((ε|ε)|ε((ε|(ε|ε))|((f3|((f3|f3)|(f
1|(ε|ε)f1)))|(ε|ε)f1)|((ε|(ε|ε))|(ε|ε))|((ε|(ε|ε))|(ε|ε))f1))|((ε|ε)|ε((ε|ε)|((f3|f3)|((f1|f1)|f1))|((ε|ε)|ε)|((ε|ε)|ε)f1))|((
(ε|ε)|ε)|(ε|ε)(ε|(ε|ε)|((f1|f1)|(ε|ε)f1)))|(((ε|ε)f1|(f1|f1))|f1|(f1|f1)|(f1|f1)|(f1|f1)(ε))|(((ε|ε)|((ε|ε)|ε))|(ε|(ε|ε))(ε|(ε
|ε)|((f1|f1)|(ε|ε)f1)))|(((((ε|ε)f1|(f1|f1))|((ε|ε)|((ε|ε)|ε))f1)|(((f1|f1)|f1)|(f1|f1)))|f1(ε|(ε|ε))|(((f1|f1)|(ε|(ε|ε))f1)|(
(f1|f1)|f1))|(f1|f1)(ε|(ε|ε))|(f1|f1)(ε|ε)(ε))|(((((ε|ε)f3|f3)|(((f1|f1)|f1)|(f1|f1)))|((((ε|ε)f1|(f1|f1))|((ε|ε)|((ε|ε)|ε))f1
)|(((f1|f1)|f1)|(f1|f1))))|((ε|ε)|((ε|ε)|ε))f1|f1((ε|(ε|ε))|(ε|ε))|(((f3|f3)|((f1|f1)|f1))|(((f1|f1)|(ε|(ε|ε))f1)|((f1|f1)|f1)
))|(ε|(ε|ε))f1|(f1|f1)((ε|(ε|ε))|(ε|ε))|(f1|f1)((ε|ε)|ε)(ε))|((((ε|ε)f3|((ε|ε)f3|f3))|(((f1|f1)|f1)|(f1|f1)))|((((ε|ε)f1|(f1|
f1))|((ε|ε)|((ε|ε)|ε))f1)|(((f1|f1)|f1)|(f1|f1))))|((ε|ε)|((ε|ε)|ε))f1|f1((ε|(ε|ε))|(ε|ε))|(((f3|(f3|f3))|((f1|f1)|f1))|(((f1|
f1)|(ε|(ε|ε))f1)|((f1|f1)|f1)))|(ε|(ε|ε))f1|(f1|f1)((ε|(ε|ε))|(ε|ε))|(f1|f1)((ε|ε)|ε)(ε))|(((ε|ε)|((ε|ε)|ε))|(ε|(ε|ε))(ε|(f1|f
1)))
```

DOMOTICA

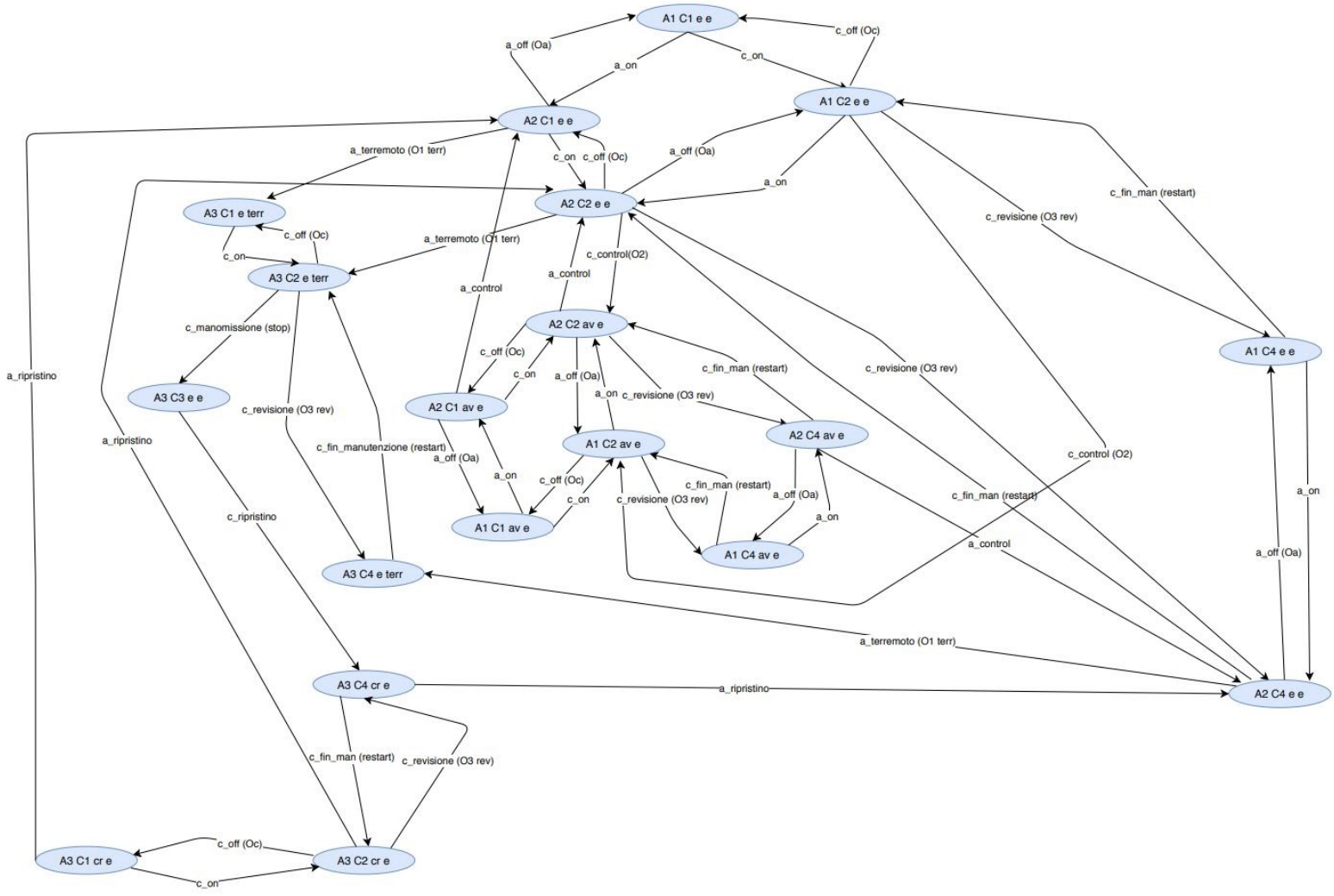
L'ultimo benchmark è stato fatto su una ipotetica rete reale costituita da una caldaia che comunica con il relativo allarme.





Calcolo relativo spazio comportamentale

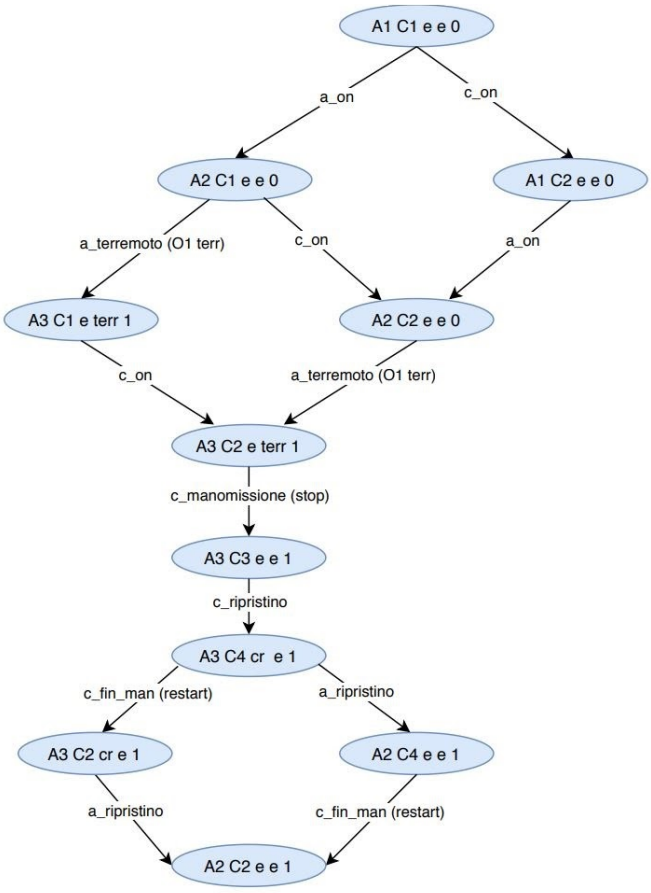
Di seguito viene rappresentato lo spazio comportamentale mentre di seguito verrà riportata una parte dell'output. Per poter visionare lo spazio comportamentale cliccare qua.



Per poterlo visionare in maniera completa l’output cliccare qua.


```
{
    "number comportamental FA": 2,
    "number states": 7,
    "number space states": 19,
    "number transitions": 49,
    "space_state": [
        {
            "name": 0,
            "link": {
                "L1": "\u03b5",
                "L2": "\u03b5"
            },
            "state": {
                "CFAN 0": "a1",
                "CFAN 1": "c1"
            },
            "next": {
                "a_on": {
                    "state": {
                        "CFAN 0": "a2",
                        "CFAN 1": "c1"
                    },
                    "link": {
                        "L1": "\u03b5",
                        "L2": "\u03b5"
                    },
                    "observable": null,
                    "relevant": null
                },
                "c_on": {
                    "state": {
                        "CFAN 0": "a1",
                        "CFAN 1": "c2"
                    },
                    "link": {
                        "L1": "\u03b5",
                        "L2": "\u03b5"
                    },
                    "observable": null,
                    "relevant": null
                }
            }
        },
        {
            "name": 1,
            "link": {
                "L1": "\u03b5",
                "L2": "\u03b5"
            },
```

Costruzione dello spazio comportamentale relativo a un’osservazione



A sinistra viene riportata la rappresentazione grafica dello spazio comportamentale relativo all’osservazione lineare O = [o1].

Di seguito viene presentato una porzione dell’output relativo a tale task. Per poterlo visionare in maniera completa cliccare qua.


```
{
    "observation": [
        "o1"
    ],
    "number comportamental FA": 2,
    "number states": 7,
    "number space states": 11,
    "number transitions": 13,
    "space_state_linear_observation": [
        {
            "name": 4,
            "obs_index": 1,
            "link": {
                "L1": "\u03b5",
                "L2": "\u03b5"
            },
            "state": {
                "CFAN 0": "a3",
                "CFAN 1": "c3"
            },
            "next": {
                "c_ripristino": {
                    "state": {
                        "CFAN 0": "a3",
                        "CFAN 1": "c4"
                    },
                    "link": {
                        "L1": "cr",
                        "L2": "\u03b5"
                    }
                }
            }
        },
        {
            "name": 6,
            "obs_index": 1,
            "link": {
                "L1": "\u03b5",
                "L2": "\u03b5"
            },
            "state": {
                "CFAN 0": "a2",
                "CFAN 1": "c4"
            },
            "next": {
                "c_fine_manutenzione": {
                    "state": {
                        "CFAN 0": "a2",
                        "CFAN 1": "c2"
                    },
```

Diagnosi relativa a una osservazione lineare

Di seguito uno screen del risultato rispetto all'osservazione O = [o1].

```
CFANS respect observation ['o1'] complete
Start computing diagnosis
Diagnosis complete (terr(stop|stop(ε|(restart|restart)))|(ε|ε)terr(stop|stop(ε|(restart|restart))))
```

Diagnosticatore di una rete di FA comportamentali

Di seguito viene presentato una porzione dell'output relativo a tale task. Per poterlo visionare in maniera completa cliccare qua.


```
{
    "number space states": 19,
    "number closures": 16,
    "number transactions": 159,
    "closure": [
        {
            "name": "x0",
            "in_state_id": 0,
            "regex": "\u03b5|\u03b5|\u03b5|(\u03b5|\u03b5)",
            "exit": {
                "oa": [
                    {
                        "successor": "x0",
                        "trns_regex": "\u03b5"
                    },
                    {
                        "successor": "x5",
                        "trns_regex": "(\u03b5|\u03b5)"
                    }
                ],
                "o1": [
                    {
                        "successor": "x1",
                        "trns_regex": "\u03b5terr"
                    },
                    {
                        "successor": "x6",
                        "trns_regex": "(\u03b5|\u03b5)terr"
                    }
                ],
                "oc": [
                    {
                        "successor": "x0",
                        "trns_regex": "\u03b5"
                    },
                    {
                        "successor": "x7",
                        "trns_regex": "(\u03b5|\u03b5)"
                    }
                ],
                "o2": [
                    {
                        "successor": "x2",
                        "trns_regex": "\u03b5"
                    },
                    {
                        "successor": "x8",
                        "trns_regex": "(\u03b5|\u03b5)"
                    }
                ],
                "o3": [
                    {
                        "successor": "x3",
                        "trns_regex": "\u03b5rev"
                    },
                    {
                        "successor": "x9",
                        "trns_regex": "(\u03b5|\u03b5)rev"
                    }
                ]
```


Calcolo diagnosi relativa ad una osservazione lineare

Di seguito uno screen del risultato rispetto all'osservazione O = [o1]

```
Start evaluate diagnosis respect observation ['o1']
(εterr(stop|stop|(stoprestart|stoprestart)))|((ε|ε)terr(stop|stop|(stoprestart|stoprestart)))
```