

Relazione progetto

Algoritmi e strutture dati

Pietro Venturini - 715166

Giacomo Bontempi - 715289

Università degli Studi di Brescia, 2021

Sommario

Sommario.....	1
Introduzione	3
Linguaggio utilizzato	3
Esecuzione del programma compilato	3
Esecuzione del programma utilizzando un IDE	4
Documentazione	4
Principali strutture dati utilizzate	4
Test-driven development.....	5
Implementazione: strutture dati	6
Automa a stati finiti.....	6
Automa a stati finiti usato come modello comportamentale	9
Rete di FA comportamentali	10
Spazio comportamentale.....	11
Chiusura silenziosa	12
Chiusura silenziosa decorata.....	13
Spazio delle chiusure decorato	15
Diagnosticatore.....	16
Implementazione: algoritmi.....	17
Calcolo dello spazio comportamentale.....	17
Spazio comportamentale relativo ad un'osservazione lineare.....	18
Calcolo della diagnosi relativa ad un'osservazione lineare	20
Calcolo della chiusura silenziosa relativa ad uno stato	21
Calcolo dello spazio delle chiusure decorato	22
Calcolo del diagnosticatore.....	23
Calcolo della diagnosi lineare.....	24
Salvataggio su file	26
Benchmarks	29
Utilizzo dell'applicazione.....	31

Menu iniziale	31
Menu del progetto corrente	32
Menu di creazione di un automa a stati finiti usato come modello comportamentale	34
Menu di creazione della rete di FA comportamentali.....	36
Menu per operare sulla rete di FA comportamentali	38
<i>Sperimentazione.....</i>	<i>44</i>
Esempio della consegna.....	44
Secondo esempio della consegna.....	45
Esempio benchmark.....	46
Esempio creato dal gruppo	46

Introduzione

Con questa relazione vogliamo innanzitutto descrivere le scelte di progettazione effettuate e le considerazioni che le hanno influenzate. Inoltre, descriveremo i principali algoritmi implementati. In conclusione, proporremo un paio di esempi di cui discuteremo i risultati ottenuti.

Linguaggio utilizzato

Per questo progetto si è scelto di utilizzare esclusivamente Java, in quanto, dal momento che le richieste pervenute implicavano un certo livello di strutturazione e organizzazione delle classi coinvolte, abbiamo preferito utilizzare un linguaggio con cui abbiamo maggiore confidenza rispetto alla creazione di progetti ben strutturati, con una quantità notevole di requisiti e funzionalità. Richieste come il fatto che *un automa comportamentale dovesse essere utilizzato all'interno di una struttura più complessa (una rete)*, o il fatto che *i nodi di uno spazio comportamentale dovessero rappresentare particolari configurazioni di una rete*, hanno indotto lunghe sessioni di riflessione su come effettuare fin da subito scelte implementative che potessero accogliere requisiti di tale natura. L'evoluzione del progetto sarebbe stata possibile solo grazie all'impiego di regole di buona progettazione legate all'ingegneria del software. Avendo già lavorato in quest'ottica con Java, nel il corso di Ingegneria del software, abbiamo deciso di utilizzare il medesimo linguaggio anche in questo progetto.

Esecuzione del programma compilato

È richiesta la versione **Java 15** affinché il programma possa essere correttamente eseguito.

Per eseguire il programma è sufficiente utilizzare un Terminale e, dopo essersi assicurati di trovarsi nella *root directory* del progetto (ossia nella directory Progetto/), eseguire il comando

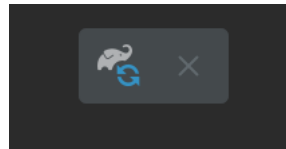
```
java -jar Automata.jar
```

Si avvierà un'applicazione a linea di comando, il cui funzionamento è spiegato nella sezione [Utilizzo dell'applicazione](#).

Nota: in alcuni casi, l'applicazione può essere avviata direttamente facendo doppio click sul file Automata.jar, tuttavia non siamo sicuri se il funzionamento di questa procedura dipenda dal sistema operativo in uso, piuttosto che da particolari configurazioni dello stesso. Si consiglia dunque la procedura da linea di comando descritta sopra.

Esecuzione del programma utilizzando un IDE

Affinché l'esecuzione del programma, compilando il codice al momento, abbia successo, è necessario installare tutte le dipendenze necessarie. Esse sono elencate all'interno del file `build.gradle` e possono essere installate comodamente utilizzando *gradle* (l'IDE dovrebbe suggerire automaticamente come includere tutte le dipendenze sfruttando tale strumento). Ad esempio, utilizzando IntelliJ e visualizzando il contenuto del file `build.gradle`, compare la seguente icona:



che, se premuta, sincronizza il progetto scaricando tutte le dipendenze necessarie.

Documentazione

La documentazione (javadoc) dell'intero progetto può essere trovata nella *root* directory del progetto, all'interno della directory `./javadoc`. La documentazione può essere navigata aprendo con un browser il file `index.html` presente in tale directory.

Principali strutture dati utilizzate

Sebbene nelle sezioni relative a ciascun componente discuteremo le scelte di implementazione più nel dettaglio, vogliamo comunque riportare alcune considerazioni generali legate alle principali strutture dati utilizzate nel progetto.

La maggior parte degli elementi coinvolti (FA comportamentali, spazi comportamentali, chiusure silenziose, diagnosticatori) sono degli automi a stati finiti o varianti di questo. Altri elementi, come le reti di FA comportamentali, hanno comunque una natura simile, in quanto sono composti da nodi interconnessi da archi, e in cui è possibile parlare di *stato della rete*. Abbiamo convenuto che una naturale rappresentazione di tali elementi fosse tramite dei **grafi**.

A tal proposito è seguita una ricerca di quali fossero le principali librerie Java già esistenti per la rappresentazione e gestione di grafi, individuando JUNG, JGraphT e Guava Graph. Il fatto di voler utilizzare una libreria ad hoc, al posto di implementare i grafi mediante matrici o liste di adiacenza, è dovuto sia alla volontà di abbracciare i principi di buona progettazione, favorendo il riuso e l'interoperabilità con altri sistemi di natura analoga a questo (che presumibilmente faranno anch'essi uso di librerie consolidate in materia), sia per poter utilizzare strutture dati ottimizzate

per la gestione di grafi, ed evitare quindi, per esempio, di rischiare di lavorare con matrici di adiacenza di grandi dimensioni ma magari fortemente sparse.

La libreria scelta è [Guava](#), una libreria di Google che fornisce un elevato numero di strutture dati ottimizzate e funzionalità: collezioni che estendono l'ecosistema di collezioni di Java (collezioni immutabili, multisets, multimaps, tabelle, mappe bidirezionali...), cache, primitive, funzioni di hashing, grafi e molto altro. In particolare, il package [Graphs](#) di tale libreria permette la modellizzazione di grafi semplici (Graph), grafi pesati (ValueGraph) e reti (Network).

Per il nostro progetto abbiamo adottato la terza tipologia, ossia Network, che permette di avere archi paralleli tra coppie di nodi e fornisce metodi aggiuntivi rispetto alle altre due, come `outEdges(node)`, `incidentNodes(edge)`, e `edgesConnecting(nodeU, nodeV)`.

A basso livello un oggetto di tipo Network è implementato principalmente mediante due hashmap:

- `nodeConnections`: a ciascun nodo associa un oggetto contenente a sua volta le mappe
 - `inEdgeMap`: associa a ciascun arco in ingresso il nodo sorgente
 - `outEdgeMap`: associa a ciascun arco in uscita il nodo destinazione
- `edgeToReferenceNode`: a ciascun arco associa il nodo sorgente

Altri dettagli implementativi possono essere reperiti ispezionando il codice sorgente della libreria, diciamo solo che oltre alle mappe sopra introdotte, vi sono anche attributi per descrivere la rete (come `isDirected`, `allowParallelEdges` e altri) strutture dati per memorizzare l'ordine di inserimento dei nodi (utile in quei contesti in cui ha senso tenerne traccia) e cache per velocizzare l'accesso a nodi e archi recentemente aggiunti o acceduti.

Test-driven development

Per tutta la fase di scrittura del codice abbiamo adottato il modello di sviluppo *test-driven development*, scrivendo i test automatici prima di implementare i vari concetti. Questo ci ha permesso di mantenere l'intero progetto sotto test per tutto lo sviluppo, assicurandoci in ogni momento che le modifiche più recenti non avessero compromesso il codice scritto fino a quel momento.

Implementazione: strutture dati

In questa sezione discuteremo come abbiamo implementato i principali elementi descritti nella consegna del progetto.

Automa a stati finiti

Il modo in cui abbiamo deciso di rappresentare un automa (ed elementi simili, come gli automi comportamentali) è cambiato all'incirca verso la metà del progetto, in quanto abbiamo ritenuto la scelta inizialmente fatta come non idonea ad accogliere le richieste che sarebbero pervenute nella seconda metà del progetto. Descriveremo ora la proposta iniziale, per poi dimenticarla e parlare, per tutti gli altri elementi, esclusivamente riferendoci alla versione finale del progetto.

Inizialmente, un automa a stati finiti (FA) era rappresentato mediante tre classi:

State(String name, EnumSet<StateType> type): rappresenta un singolo stato di un automa, caratterizzato da un nome, e da un insieme di *enum* di tipo StateType, che può essere INITIAL, ACCEPTANCE o FINAL.

Transition(String symbol): rappresenta una singola transizione, dotata di un simbolo.

FA(Network<State, Transition> network): rappresenta un automa a stati finiti, caratterizzato da un grafo i cui nodi sono oggetti di tipo State e i cui archi sono oggetti di tipo Transition.

Il principale problema riscontrato è dovuto al fatto che uno stato può essere considerato iniziale rispetto ad un automa, e non iniziale rispetto ad un altro automa. Ad esempio, lo stato “8” dello spazio comportamentale in Figura 1 non è iniziale, tuttavia, compare anche all'interno dello spazio delle chiusure silenziose in Figura 2, sia come stato iniziale (nella chiusura x3) che come stato non iniziale delle chiusure x4 e x6.

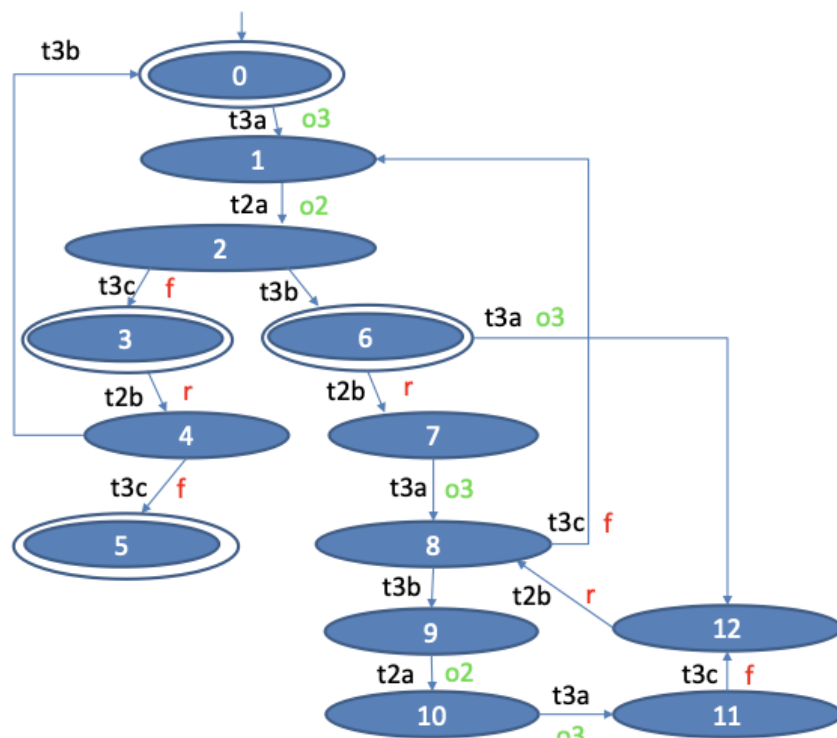


Figura 1: Esempio di spazio comportamentale proposto a pagina 38 della consegna del progetto.

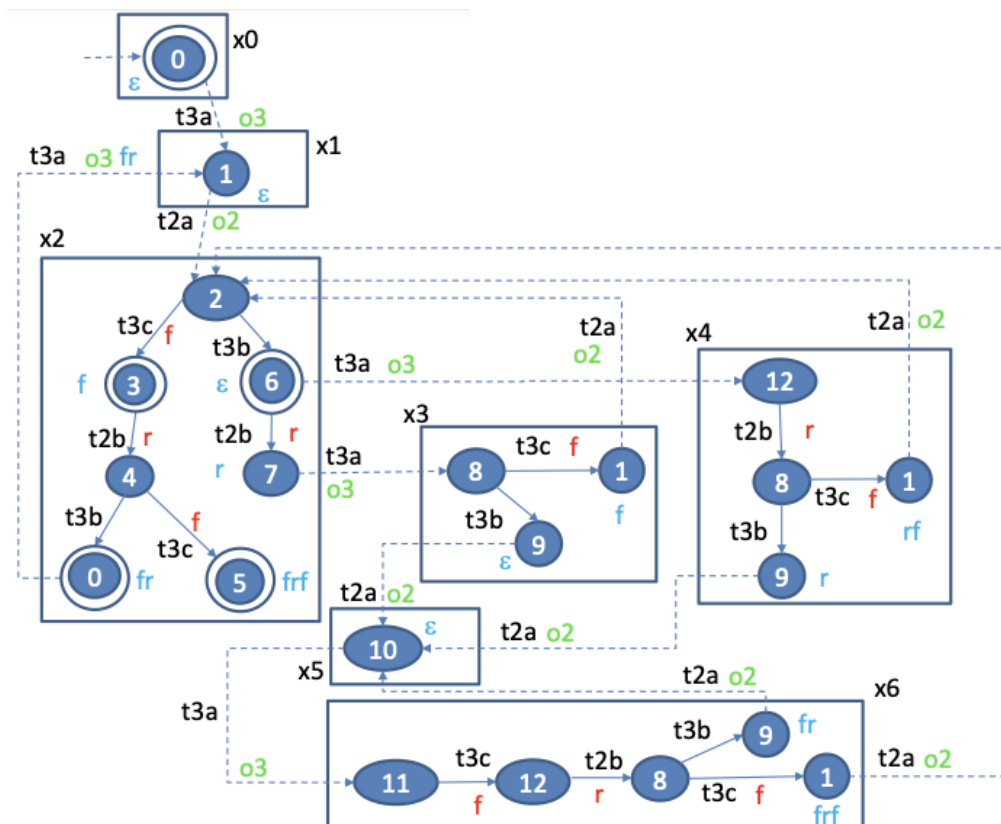


Figura 2: Spazio delle chiusure calcolato a partire dall'automa di pagina 38 della consegna, riportato in Figura 1.

Per questo motivo, la responsabilità di conoscere se uno stato sia iniziale, piuttosto che finale non deve essere data allo stato stesso, ma deve essere spostata al singolo automa. In questo modo, lo stesso stato (inteso come oggetto Java) può essere utilizzato all'interno di più automi differenti, che possono rappresentare concetti distinti, e per i quali tale stato può avere ruoli diversi. Le stesse considerazioni possono anche essere estese alle classi che rappresentano delle reti (come la rete di FA comportamentali di cui parleremo in seguito).

Tenendo conto di questo fatto, abbiamo riprogettato le classi che rappresentano automi a stati finiti (FA) in questo modo:

State(String name): rappresenta un singolo stato di un automa, caratterizzato da un nome.

Eventuali ulteriori attributi possono essere aggiunti nelle sottoclassi di questa.

Transition(String symbol): rappresenta una singola transizione, dotata di un simbolo. Eventuali ulteriori attributi (come etichette di osservabilità/rilevanza) possono essere aggiunti nelle sottoclassi di questa

FA<S extends State, T extends Transition>(Network<S, T> network, S initialState, Set<S> acceptanceStates, Set<S> finalStates): rappresenta un automa a stati finiti, caratterizzato da un grafo i cui nodi sono sottotipi di State e i cui archi sono sottotipi di Transition. Tiene inoltre traccia dello stato iniziale, dell'insieme di stati di accettazione e dell'insieme di stati finali. Eventuali ulteriori attributi possono essere specificati nelle sottoclassi.

La classe parametrica FA accetta come stati (nodi) qualsiasi sottoclasse di State e come transizioni (archi) qualsiasi sottoclasse di Transition.

L'istanziamento di un FA avviene utilizzando la classe FABuilder che implementa il pattern *builder*.

Automa a stati finiti usato come modello comportamentale

Questi elementi, che chiameremo BFA (Behavioral Finite Automata), sono stati implementati nella classe BFA in modo analogo a quanto fatto per i FA, tenendo però solo traccia dello stato iniziale e non di quelli di accettazione o finali. Tuttavia, grazie ad un attributo `currentState` viene tenuta traccia dello stato corrente. Un BFA è così rappresentato:

State(String name): rappresenta un singolo stato di un automa, caratterizzato da un nome.

EventTransition(String name, String inEvent, Set<String> outEvents, String observabilityLabel, String relevanceLabel): rappresenta una singola transizione, dotata di un nome, di un evento in ingresso, di un insieme di eventi in uscita, da un'etichetta di osservabilità e da un'etichetta di rilevanza. Il fatto che alcuni di questi attributi siano opzionali è gestito dai metodi mutatori.

BFA(Network<State, EventTransition> network, State initialState, State currentState): rappresenta un automa a stati finiti usato come modello comportamentale. È caratterizzato da un grafo i cui nodi oggetti di tipo `State` e i cui archi sono oggetti di tipo `EventTransition`. Tiene inoltre traccia dello stato iniziale e dello stato corrente.

L'istanziamento di un BFA avviene tramite la classe `BFABuilder` che implementa il pattern *builder*.

Rete di FA comportamentali

Una rete di BFA è rappresentata tramite i seguenti concetti:

Link(String name, String event): rappresenta un collegamento della rete. È caratterizzato da un nome e da un evento opzionale (l'opzionalità è gestita dai metodi mutatori).

BFANetwork(Network<BFA, Link> network): rappresenta la rete di BFA ed è caratterizzata da un attributo di tipo Network, i cui nodi sono oggetti di tipo BFA e i cui archi sono oggetti di tipo Link. Lo stato della rete viene calcolato ogni qual volta si invochino i metodi:

- `isFinal()`: verifica che tutti i link siano vuoti,
- `isInitial()`: verifica tutti i link siano vuoti e che tutti i nodi (BFA) siano nel loro stato iniziale.

La costruzione di una rete di BFA avviene utilizzando la classe `BFANetworkBuilder` che implementa il pattern *builder*.

Spazio comportamentale

Per rappresentare lo spazio comportamentale, si è utilizzata la [classe FA](#), dove però il tipo di stati e il tipo di transizioni è stato specificato da due nuove classi: `BSState` e `BSTransition`, dove *BS* è l'acronimo di *Behavioral Space*.

`BSState(String name, Map<BFA, State> bfas, Map<Link, String> links)`: rappresenta uno stato dello spazio comportamentale ossia uno *snapshot* della rete di BFA in un certo stato. Esso è costituito da un nome, da una mappa `bfas` che a ciascun nodo della rete di BFA fa corrispondere lo stato corrispondente, e infine da una mappa `links` che a ciascun collegamento della rete fa corrispondere l'evento in esso contenuto.

`BSTransition(String name, String observabilityLabel, String relevanceLabel)`: rappresenta una transizione di uno spazio comportamentale. Essa è costituita da un nome, da un'etichetta di osservabilità e da un'etichetta di rilevanza. La presenza di tali etichette può essere verificata tramite opportuni metodi della classe, consultabili nella documentazione.

La generazione dello spazio comportamentale a partire da una rete di BFA viene fatta attraverso il metodo `getBehavioralSpace` descritto nella [sezione sugli algoritmi](#).

Chiusura silenziosa

Per rappresentare una chiusura silenziosa, si è utilizzata la [classe FA](#), dove però il tipo di stati e il tipo di transizioni, sono specificati dalle classi BState e BTransition, in modo analogo a quanto fatto per lo [spazio comportamentale](#). Il calcolo della chiusura silenziosa (relativa ad uno stato di uno spazio comportamentale) avviene tramite il metodo silentClosure della classe BFANetworkSupervisor, descritto nella [sezione sugli algoritmi](#).

Naturalmente, lo stato iniziale dell'automa a stati finiti è lo stato cui la chiusura silenziosa fa riferimento, gli stati finali sono gli stati della chiusura che erano finali per lo spazio comportamentale di partenza, e gli stati di accettazione sono dati dall'unione degli stati finali e degli stati di uscita (calcolati dall'algoritmo in fase di creazione della chiusura silenziosa).

Eventuali etichette di osservabilità o di rilevanza sono ereditate dallo spazio comportamentale cui fa riferimento.

Chiusura silenziosa decorata

È rappresentata nello stesso modo di una normale chiusura silenziosa, eccetto per la classe utilizzata per rappresentare il tipo degli stati. Invece di estendere la classe `BSSState` per aggiungere la decorazione relativa allo stato, si è preferito utilizzare la composizione (seguendo il principio *prefer composition over inheritance* e il pattern *decorator*) creando una nuova classe `DBSSState`, che significa Decorated Behavioral Space State:

DBSSState(BSSState state, String decoration): rappresenta uno stato decorato dello spazio comportamentale (utilizzato all'interno delle chiusure silenziose decorate).

In questo modo evitiamo al problema che, nello spazio delle chiusure, ad un medesimo stato, possano corrispondere decorazioni differenti all'interno di chiusure differenti, come in Figura 3 dove, allo stato "1" corrispondono le decorazioni "e" nella chiusura x1, "f" nella chiusura x3, "rf" nella chiusura x4 e "frf" nella chiusura x6.

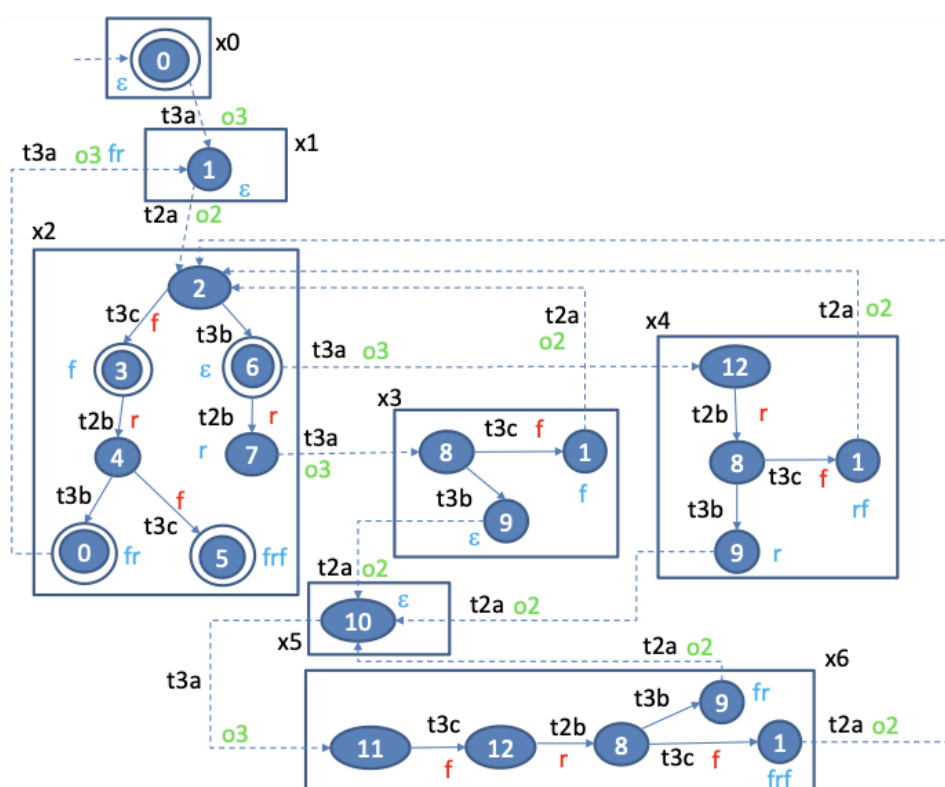


Figura 3: Spazio delle chiusure silenziose decorate relativo allo spazio comportamentale di Figura 1.

Sfruttando la classe `BFANetworkSupervisor`, è possibile calcolare una chiusura silenziosa decorata tramite il metodo `decoratedSilentClosure`, descritto successivamente nella [sezione sugli algoritmi](#). La diagnosi di una chiusura dotata di stati finali può essere calcolata con il metodo `diagnosis`.

Spazio delle chiusure decorato

Lo spazio delle chiusure decorato può essere calcolato utilizzando il metodo `decoratedSpaceOfClosures` della classe `BFANetworkSupervisor`. Non è stata creata una classe ad hoc per rappresentare questo elemento, bensì si è sfruttata la classe parametrica `FA` parametrizzata in questo modo:

- Gli stati dell'automa sono oggetti di tipo `FA<DBSState, BSTransition>`, ossia sono a loro volta dei `FA`. Questo al fine di non perdere l'informazione riguardante il contenuto di ciascuna chiusura, quando questa è vista come nodo di un automa di livello d'astrazione più alto.
- Le transizioni dell'automa sono oggetti di tipo `DSCTransition`, dove `DSC` è l'acronimo di *Decorated Silent Closure* ed è così formata:

`DSCTransition(String name, String symbol, String observabilityLabel)`: rappresenta una transizione di uno spazio delle chiusure. Essa è dotata di un nome (ereditato dallo spazio comportamentale da cui si è giunti a questo), da un simbolo (ottenuto concatenando alla decorazione dello stato sorgente, l'etichetta di rilevanza ereditata), e da un'etichetta di osservabilità (anch'essa ereditata).

Diagnosticatore

Dal momento che non era richiesto conoscere il contenuto degli stati del diagnosticatore (che sono chiusure silenziose decorate), è stata creata la classe `Diagnostician`, così formata:

FA<State, DSCTransition> fa: l'automa che rappresenta il diagnosticatore. Gli stati hanno un nome (ereditato dalla chiusura silenziosa corrispondente), e le transizioni sono transizioni decorate, ereditate dallo spazio delle chiusure.

Map<State, Map<DBSState, String>> diagnosis: una mappa che a ciascuno stato del diagnosticatore (ossia una chiusura) fa corrispondere un'altra mappa, che rappresenta la diagnosi di tale chiusura. Si è scelto di usare una mappa invece che direttamente una stringa (ad esempio "decorazione1 | decorazione2") al fine di tenere traccia di quale stato della chiusura corrisponda a ciascuna decorazione (anche perché il metodo che si occupa di calcolare la diagnosi di una singola chiusura, restituisce una mappa di questo tipo e non una stringa).

Implementazione: algoritmi

In questa sezione discuteremo i principali algoritmi implementati. Fin da subito diciamo che le classi principali che li contengono sono

- `AcceptedLanguage`: include l'implementazione dell'algoritmo Espressione Regolare, descritto alle pagine 11 e 12 della consegna del progetto.
- `AcceptedLanguages`: include l'implementazione dell'algoritmo Espressioni Regolari, descritto alle pagine 17-20 della consegna del progetto.
- `BFANetworkSupervisor`: include l'implementazione di tutti quei metodi che lavorano su una rete di BFA o su strutture dati da essa derivate (come il calcolo della diagnosi, la generazione dello spazio delle chiusure, e così via).

Calcolo dello spazio comportamentale

L'algoritmo richiesto nel primo compito, a pagina 39 della consegna del progetto, per la creazione dello spazio comportamentale di una rete di FA comportamentali, è mostrato di seguito:

Algorithm 1 Compute Behavioral Space

```
1: procedure BEHAVIORALSPACE(network)
2:   create an empty FA
3:   initialState  $\leftarrow$  the current state of network
4:   if all the links of network in initialState are empty then
5:     add initialState as initial and final to FA
6:   else
7:     add initialState as initial to FA
8:   end if
9:   toExplore  $\leftarrow$  {initialState}
10:  explored  $\leftarrow$  {}
11:  while toExplore is not empty do
12:    state  $\leftarrow$  a state inside toExplore
13:    rollback network to state
14:    for all BFA in network do
15:      for all transition enabled in the BFA do
16:        newState  $\leftarrow$  the state of network after the execution of
                           transition
17:        if newState  $\notin$  toExplore and newState  $\notin$  explored then
18:          add newState to FA and set it final if all the
                           links of network in newState are empty
19:          add newState to toExplore
20:        end if
```

```

21:         put a new transition in FA connecting state to newState with
           the same relevance and observability label of transition
22:         rollback network to state
23:     end for
24: end for
25: remove state from toExplore
26: add state to explored
27: end while
28: rollback network to initialState
29: for all state in FA do
30:     if reachableNodes(states) doesn't contain any final state then
31:         remove state from FA with all its transitions
32:     end if
33: end for
34: return FA
35: end procedure

```

Tale algoritmo è stato implementato nel metodo `getBehavioralSpace` della classe `BFANetworkSupervisor`.

Il modo in cui esso opera prende spunto dall'algoritmo BFS per visita in ampiezza un grafo. Infatti, il ciclo while di riga 11 richiama quello presente in BFS, con la differenza che nel nostro caso estraiamo un elemento alla volta da un insieme, invece che da una coda, in quanto l'ordine di estrazione degli elementi in quell'insieme non è importante.

La procedura "rollback network to state" di riga 13 fa in modo che lo stato corrente della rete di BFA diventi uguale a *state*, ossia ad uno stato che non è ancora stato esplorato del tutto (nel senso che sono possibili ancora dei passaggi di stato a partire da tale stato).

Spazio comportamentale relativo ad un'osservazione lineare

L'algoritmo richiesto nel secondo compito, a pagina 46 della consegna, prevede il calcolo dello spazio comportamentale di una rete di BFA, relativo ad un'osservazione lineare data. Lo pseudocodice che descrive tale algoritmo è mostrato di seguito e, come preconditione, richiede che l'osservazione lineare in input non sia vuota:

Algorithm 2 Compute Behavioral Space of a Linear Observation

```
1: procedure BEHAVIORALSPACELINOBS(network, linObs)
2:   create an empty FA
3:   initialState  $\leftarrow$  the current state of network
4:   initialState.obsIndex  $\leftarrow$  0
5:   add initialState as initial to FA
6:   toExplore  $\leftarrow$  {initialState}
7:   explored  $\leftarrow$  {}

8:   while toExplore is not empty do
9:     state  $\leftarrow$  a state inside toExplore
10:    rollback network to state
11:    for all BFA in network do
12:      for all transition enabled in the BFA do
13:        newState  $\leftarrow$  the state of network after the execution of
          transition
14:        if transition.obsLabel is not  $\varepsilon$  then
15:          newState.obsIndex  $\leftarrow$  state.obsIndex + 1
16:        else
17:          newState.obsIndex  $\leftarrow$  state.obsIndex
18:        end if
19:        if newState  $\notin$  toExplore and newState  $\notin$  explored then
20:          add newState to FA and set it final if all the
            links of network in newState are empty and
            newState.obsIndex is linObs.length
21:          add newState to toExplore
22:        end if
23:        put a new transition in FA connecting state to newState with
          the same relevance and observability label of transition
24:        rollback network to state
25:      end for
26:    end for
27:    remove state from toExplore
28:    add state to explored
29:  end while
30:  rollback network to initialState
31:  if FA doesn't contain any final state then
32:    return error
33:  end if
34:  for all state in FA do
35:    if reachableNodes(states) doesn't contain any final state then
36:      remove state from FA with all its transitions
37:    end if
38:  end for
39:  return FA
40: end procedure
```

Questo algoritmo è implementato nel metodo `getBehavioralSpaceForLinearObservation` della classe `BFANetworkSupervisor`.

Si noti che l'algoritmo richiede in input la rete di BFA comportamentali dal momento che la richiesta prevedeva che l'algoritmo non conoscesse a priori lo spazio comportamentale completo. Inoltre, a differenza dell'[Algoritmo 1](#), che alle righe 4-8 controllava se lo stato iniziale dovesse essere anche finale, nel caso dell'Algoritmo 2 questo controllo non è necessario, in quanto lo stato iniziale sarebbe anche finale se e solo se l'osservazione lineare in input fosse vuota. Tale caso non può verificarsi, in quanto, qualora l'osservazione lineare fosse vuota, verrebbe generata un'eccezione e segnalato all'utente.

Calcolo della diagnosi relativa ad un'osservazione lineare

Tale calcolo avviene a partire dallo spazio comportamentale relativo all'osservazione lineare stessa, il cui algoritmo è descritto nell'[Algoritmo 2](#). Tale calcolo è gratuitamente svolto dall'algoritmo `EspressioniRegolari` implementato all'interno della classe `AcceptedLanguages` e può essere calcolato invocando uno dei due metodi `reduceFatoMultipleRegex` o `reduceFatoMapOfRegex`. Ricordiamo che tale algoritmo richiede in input un automa a stati finiti di cui calcolerà le espressioni regolari accettate da ciascuno stato di accettazione; per questo motivo, è necessario un passaggio preliminare in cui gli stati di accettazione dello spazio comportamentale vengano posti uguali ai suoi stati finali. Lo pseudocodice è mostrato di seguito:

Algorithm 3 Compute Linear Diagnosis of Behavioral Space

```
1: procedure LINEARDIAGNOSISBS(behavioralSpace)
2:   set all final states in behavioralSpace to acceptance states
3:   diagnosis  $\leftarrow$  REGULAREXPRESSIONS(behavioralSpace)
4:   return diagnosis
5: end procedure
```

Calcolo della chiusura silenziosa relativa ad uno stato

Lo pseudocodice che implementa la richiesta del quarto compito, di pagina 73 della consegna del progetto, è mostrato di seguito:

Algorithm 4 Compute the Decorated Silent Closure

```
1: procedure DECORATEDSILENTCLOSURE(behavioralSpace, state)
2:   copy  $\leftarrow$  GETCOPY(behavioralSpace)
3:   observableTransitions  $\leftarrow$  GETOBSERVABLETRANSITIONS(behavioralSpace)
4:   REMOVETRANSITIONS(copy, observableTransitions)
5:   nodes  $\leftarrow$  REACHEBLENODES(copy, state)
6:   inducedSubgraph  $\leftarrow$  INDUCEDSUBGRAPH(copy, nodes)
7:   finalStates  $\leftarrow$  GETFINALSTATES(inducedSubgraph)
8:   exitStates  $\leftarrow$  GETEXITSTATES(inducedSubgraph, behavioralSpace)
9:   acceptanceStates  $\leftarrow$  finalStates  $\cup$  exitStates
10:  silentClosure  $\leftarrow$  a new FA having state as initial state,
      inducedGraph as graph, finalStates as final states and
      acceptanceStates as acceptance states
11:  decorations  $\leftarrow$  REGULAREXPRESSIONS(silentClosure)
12:  assign to each acceptance state in acceptanceStates its decoration
      in decorations
13:  return silentClosure
14: end procedure
```

Tale algoritmo è implementato dai metodi `silentClosure` e `decoratedSilentClosure` della classe `BFANetworkSupervisor`.

Si noti che tale algoritmo calcola anche le decorazioni relative a ciascuno stato di accettazione (ossia a ciascuno stato finale o di uscita), invocando `EspressioniRegolari`, implementato nella classe `AcceptedLanguages`.

La riga 3 dello pseudocodice richiede l'insieme delle transizioni dell'intero spazio comportamentale dotate di etichetta di osservabilità non nulla. La riga 4 rimuove tali transizioni dalla copia dello spazio che è stata precedentemente creata.

La funzione `reachableNodes` di riga 5 è implementata dall'omonima funzione privata della classe `BFANetworkSupervisor`, che altro non è che l'algoritmo Breadth First Search calcolato a partire dallo stato *state* di cui si vuole calcolare la chiusura silenziosa. Avendo precedentemente rimosso le transizioni non osservabili, invocando questa funzione calcoliamo l'insieme *nodes* di nodi dell'intero spazio comportamentale che siano raggiungibili tramite cammini non osservabili da *state*.

La funzione `inducedSubgraph` è fornita dalla libreria utilizzata e restituisce la porzione di grafo contenente tutti e soli i nodi specificati nell'insieme `nodes`, passato come argomento, più le transizioni che li connettono, scartando tutte le rimanenti.

Le righe 7-10 si occupano di costruire un nuovo FA a partire dal grafo appena ottenuto. Infine, le decorazioni relative agli stati finali o di uscita vengono calcolate invocando l'algoritmo `EspressioniRegolari`, implementato dal metodo `reduceFatoMapOfRegex` della classe `AcceptedLanguages`.

Calcolo dello spazio delle chiusure decorato

La seconda richiesta del compito di pagina 73 della consegna è svolta dal seguente algoritmo:

Algorithm 5 Compute the Decorated Closure Space

```

1: procedure CLOSURESPACE(behavioralSpace)
2:   entryStates  $\leftarrow$  GETENTRIES(behavioralSpace)
3:   decoratedSilentClosures  $\leftarrow$  {}
4:   for all state  $\in$  entryStates do
5:     decoratedClosure  $\leftarrow$  DECORATEDSILENTCLOSURE(behavioralSpace, state)
6:     add (state, decoratedClosure) to decoratedSilentClosures
7:   end for
8:   create an empty FA
9:   for all (s, dc)  $\in$  decoratedSilentClosures do
10:    if the initial state of dc is initial also for behavioralSpace then
11:      add dc as initial state to FA
12:    end if
13:    if dc contains any final state of behavioralSpace then
14:      add dc as acceptance state to FA
15:    end if
16:    exitStates  $\leftarrow$  GETEXITSTATES(dc, behavioralSpace)
17:    for all source  $\in$  exitStates do
18:      for all observable transition t outgoing from source do
19:        target  $\leftarrow$  the target of t in behavioralSpace
20:        dc'  $\leftarrow$  a decorated closure such that
21:          (target, dc')  $\in$  decoratedSilentClosures
22:        rel  $\leftarrow$  source.decoration concatenate to t.relLabel
23:        connect dc to dc' in FA with a transition having relevance
24:          label rel and observability label t.obsLabel
25:      end for
26:    end for
27:  return FA
28: end procedure

```

L'Algoritmo 5 è implementato dal metodo `decoratedSpaceOfClosures` della classe `BFANetworkSupervisor`.

Alla riga 2, la funzione `getEntries` si occupa di ricavare tutti gli stati dello spazio comportamentale che siano stati di cui possa essere calcolata la relativa chiusura silenziosa.

Tale funzione filtra gli stati che abbiano almeno una transizione osservabile in ingresso o che siano iniziali.

`decoratedSilentClosures` è una mappa che a ciascuno stato dello spazio comportamentale di cui è stata calcolata la relativa chiusura, fa corrispondere la relativa chiusura silenziosa decorata.

Alla riga 16, la funzione `getExitStates` restituisce gli stati della chiusura silenziosa considerata, che siano di uscita, ossia che abbiano almeno una transizione osservabile in uscita nello spazio comportamentale.

Alla riga 19, con `target` si intende lo stato destinazione su cui termina la transizione.

Calcolo del diagnosticatore

L'ultima richiesta del compito di pagina 73 della consegna è descritta dal seguente algoritmo:

Algorithm 6 Compute the Diagnostician

```
1: procedure DIAGNOSTICIAN(closureSpace)
2:   create an empty FA
3:   for all decoratedClosure  $\in$  closureSpace do
4:     create a new state s for the diagnostician
5:     if decoratedClosure is an acceptance state of closureSpace then
6:       assign to s the diagnosis GETDIAGNOSIS(decoratedClosure)
7:     end if
8:     add s to FA
9:     if decoratedClosure is the initial state of closureSpace then
10:      set s as the initial state of FA
11:    end if
12:    if decoratedClosure is an acceptance state of closureSpace then
13:      set s as an acceptance state of FA
14:    end if
15:  end for
16:  for all transition t in closureSpace do
17:    dc1  $\leftarrow$  the source decorated closure of t in closureSpace
18:    dc2  $\leftarrow$  the destination decorated closure of t in closureSpace
19:    create a new transition t' for the diagnostician
20:    put t' from the state s1 in FA created from dc1 to the state s2
      in FA created from dc2
21:    t'.relLabel  $\leftarrow$  t.relLabel
22:    t'.obsLabel  $\leftarrow$  t.obsLabel
23:  end for
24:  return FA
25: end procedure
```

Si ricorda che il diagnosticatore, seppur rappresentato da un'apposita struttura dati altro non è che un FA. L'Algoritmo 6 è implementato dal metodo `diagnostician` della classe `BFANetworkSupervisor`.

Alla riga 6, l'assegnamento di una diagnosi ad un particolare stato del diagnosticatore avviene tramite una mappa.

Le righe 3-15 si occupano di aggiungere al diagnosticatore tutti gli stati, mentre le righe 16-23 si occupano di aggiungere le transizioni tra questi. Per tenere traccia dello stato del diagnosticatore corrispondente a ciascuna chiusura si utilizza una mappa (necessaria poi alla riga 20 per recuperare lo stato del diagnosticatore a partire dalla chiusura).

Calcolo della diagnosi lineare

La richiesta di implementare un algoritmo che calcoli la diagnosi relativa ad un'osservazione lineare utilizzando il diagnosticatore (pagina 86 della consegna), è soddisfatta dal metodo `linearDiagnosis` della classe `BFANetworkSupervisor`.

Salvataggio su file

Al fine di soddisfare la richiesta di poter conservare in forma persistente gli oggetti costruiti e i risultati delle operazioni svolte, abbiamo adottato la serializzazione. La libreria utilizzata per la serializzazione è [Gson](#). Tuttavia, non abbiamo serializzato direttamente gli oggetti di cui disponevamo (i cui dettagli implementativi sono stati descritti nella [sezione sulle strutture dati](#)) per i seguenti motivi:

1. Molti attributi interni alle classi sono dettagli implementativi che non vanno salvati su file. Come è stato detto [inizialmente](#), gli oggetti della libreria Guava che rappresentano dei grafi contengono al loro interno molte strutture dati per migliorare l'efficienza (cache, iterator, ...). Memorizzare questi oggetti sarebbe inutile e ridondante. Inoltre, non avendo accesso diretto al codice di tale libreria, non potremmo determinare quali attributi serializzare e quali escludere tramite opportuni identificatori che andrebbero scritti in tali classi.
2. La serializzazione fallisce nel caso di associazioni riflessive: si pensi ad un nodo, di cui si memorizzano i nodi adiacenti, dei quali, a loro volta, vengono memorizzati i nodi adiacenti. Se tra una coppia di nodi vi fossero due archi in entrambe le direzioni, si entrerebbe in un loop infinito. Determinare la *“quanto profondamente serializzare”* è difficile, soprattutto nel nostro caso, in cui non abbiamo accesso diretto al codice delle librerie usate.
3. Gli automi e le reti possono essere istanziati solo tramite un builder, che prima di restituire l'oggetto creato, si occupa di verificarne la validità (ad esempio controlla che un FA non abbia più di uno stato iniziale). Deserializzando un FA, per esempio, verrebbe automaticamente invocato il costruttore, evitando quindi la fase di validazione.

Per i motivi elencati abbiamo creato delle classi “semplificate” (che possono essere trovate nel package `files`) rispetto a quelle utilizzate per i vari elementi che andremo a serializzare/deserializzare. Il nome di queste classi è dato dal nome della “vera” classe che rappresentano, seguito dalla stringa “Json”, in quanto il formato usato poi per salvare tale classe su file è appunto il formato Json. Ad esempio, alla classe BFA corrisponde la classe semplificata BFAJson.

Queste classi contengono esclusivamente il minimo indispensabile per poter memorizzare e ricostruire successivamente gli elementi corrispondenti.

Ad esempio, BFAJson è così formato:

```
String name;  
String[] states;
```

```
String initialState;           // nome dello stato iniziale
EventTransitionJson[] transitions;
```

Dove di ciascuno stato viene esclusivamente memorizzato il nome, e le transizioni sono memorizzate tramite un array di EventTransitionJson, oggetti cosiffatti:

```
String name;
String source;           // nome dello stato sorgente
String target;           // nome dello stato destinazione
String inEvent;
String[] outEvents;
String observabilityLabel;
String relevanceLabel;
```

L'utilizzo di questo livello intermedio di rappresentazione comporta due passaggi ulteriori: in fase di scrittura dobbiamo prima convertire gli oggetti nelle loro rappresentazioni semplificate, mentre in fase di lettura dobbiamo prima deserializzare gli oggetti semplificati, dopodiché, utilizzando i builder corrispondenti, andare a costruire gli oggetti veri e propri. Ad esempio, il BFA in Figura 4 viene codificato come mostrato in Figura 5.

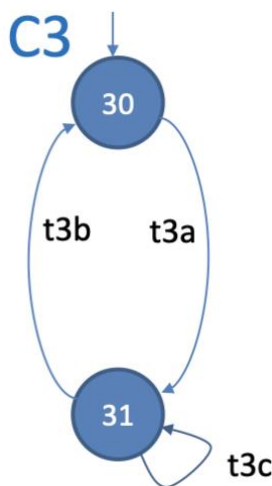


Figura 4: Esempio di automa a stati finiti usato come modello comportamentale.

```
{
  "name": "C3",
  "states": [
    "30",
    "31"
  ],
  "initialState": "30",
  "transitions": [
    {
      "name": "t3a",
      "source": "30",
      "target": "31",
      "outEvents": [
        "e2(L2)"
      ],
      "observabilityLabel": "",
      "relevanceLabel": ""
    },
    {
      "name": "t3b",
      "source": "31",
      "target": "30",
      "inEvent": "e3(L3)",
      "outEvents": [],
      "observabilityLabel": "",
      "relevanceLabel": ""
    },
    {
      "name": "t3c",
      "source": "31",
      "target": "31",
      "inEvent": "e3(L3)",
      "outEvents": [],
      "observabilityLabel": "",
      "relevanceLabel": ""
    }
  ]
}
```

Figura 5: Codifica del BFA di Figura 4 in formato json. È il risultato della serializzazione dell'oggetto BFAJson corrispondente.

Onde evitare di dover creare un'apposita “classe semplificata” per ciascuno degli elementi del progetto, abbiamo deciso di salvare su file solo i seguenti elementi:

- FA
- FA comportamentali
- Reti di FA comportamentali
- *Benchmarks*

Tutti gli elementi che non sono salvati su file, possono essere calcolati facilmente a partire dalla rete di FA comportamentali che invece è memorizzabile.

Ad ogni rete di FA comportamentali corrisponde un Progetto, il cui nome è scelto dall'utente, la cui funzione è quella di contenere tutti i file per la costruzione di una rete (i singoli FA comportamentali e la rete stessa), così come i risultati calcolati a partire da essa. Per ciascun progetto creato, viene creata una directory all'interno della directory files/ con lo stesso nome del progetto. I file sono dunque organizzati in questo modo:

```
files/
  nome_progetto_1/
    FAs/
      nome_fa_1.json
    BFAs/
      c2.json
      c3.json
    bfa_network.json
    benchmarks.json
```

La sottodirectory BFAs contiene I singoli FA comportamentali creati dall'utente (il cui file prende il nome dal nome dell'automa), che possono essere utilizzati nella creazione della rete di FA comportamentali, che verrà memorizzata in `bfa_network.json`.

Benchmarks

I *benchmarks*, ossia i risultati delle operazioni fatte per ciascun progetto, sono salvati all'interno del file `benchmarks.json`, presente in ciascun progetto, e possono essere consultati al fine di confrontare tra loro le operazioni svolte in passato (ad esempio, è possibile confrontare i benchmark relativi al

calcolo della diagnosi relativa ad un'osservazione lineare sia a partire dallo spazio comportamentale che usando il diagnosticatore).

Il file benchmark contiene un array di singoli benchmark, dove ogni singolo benchmark è costituito da:

- Data in cui il calcolo è stato effettuato
- Descrizione del calcolo effettuato (la complessità della descrizione varia a seconda della tipologia di calcolo)
- Durata in nanosecondi.

Un esempio del contenuto del file benchmarks.json è mostrato di seguito:

```
[
  {
    "date": {
      "date": {
        "year": 2021,
        "month": 2,
        "day": 2
      },
      "time": {
        "hour": 17,
        "minute": 38,
        "second": 13,
        "nano": 956720000
      }
    },
    "description": "Computation of the behavioral space ",
    "duration": 3832981
  },
  {
    "date": {
      "date": {
        "year": 2021,
        "month": 2,
        "day": 2
      },
      "time": {
        "hour": 17,
        "minute": 39,
        "second": 2,
        "nano": 87448000
      }
    },
    "description": "Computation of the diagnosis relating to the linear observation\n[03, 02, 03, 02].\nThe diagnosis is (fr|(r|r)f)fr|(fr|(r|r)f)f|(fr|(r|r)f)|(fr|(r|r)f)frf",
    "duration": 31021450
  }
]
```

I risultati possono essere visualizzati sia dal file, che interagendo con l'applicazione, come spiegato nella sezione successiva.

Utilizzo dell'applicazione

Per eseguire l'applicazione, fare riferimento alla sezione: [Esecuzione del programma compilato](#).

L'interazione tra utente e applicazione avviene attraverso una serie di menu proposti su shell, i quali offrono diverse opzioni in base alle esigenze del fruitore del sistema.

Menu iniziale

Il primo menu che viene mostrato all'utente subito dopo l'avvio dell'applicazione è il seguente:

```
Please make a selection:
1) Open an existing project
2) Create a new Project
0) Exit

Enter your choice: █
```

Esso permette all'utente di scegliere di aprire un progetto esistente creato in precedenza digitando 1 sulla tastiera oppure di creare un nuovo progetto da zero digitando 2. Ciascun progetto fa riferimento ad una specifica rete di FA comportamentali e alle strutture che a partire da questa possono essere costruite. Digitando 0 invece si termina l'esecuzione dell'applicazione.

Quando l'utente digita 1 per aprire un progetto, sullo schermo appare la lista dei progetti creati in passato; per esempio, potrebbe comparire il risultato che segue:

```
Please make a selection:
1) Open an existing project
2) Create a new Project
0) Exit

Enter your choice: 1
0) Network2
1) test

Enter your choice: █
```


Come si vede dalla figura, in questo caso l'utente aveva già generato un progetto di nome "Network2" e un progetto di nome "test": digitando 0 o 1 quindi potrà decidere quale dei due aprire nuovamente. Nel caso in cui l'utente non abbia ancora creato alcun progetto, l'applicazione mostra all'utente un messaggio che manifesta l'assenza di progetti disponibili e ripropone il menu iniziale.

Quando l'utente digita 2 per creare un nuovo progetto, il sistema richiede all'utente di inserire su tastiera il nome del nuovo progetto:

```
Please make a selection:
1) Open an existing project
2) Create a new Project
0) Exit

Enter your choice: 2

Insert the name of the project: esempio
```

Premendo invio verrà creato il nuovo progetto, il quale verrà subito aperto e porterà l'utente a uno dei menu successivi. Se esiste almeno un progetto creato in passato avente lo stesso nome appena inserito dal fruitore, verrà mostrato un messaggio che segnala l'impossibilità di creazione di progetti con lo stesso nome e verrà riproposto nuovamente il menu iniziale di selezione o creazione di un nuovo progetto.

Menu del progetto corrente

Una volta aperto un progetto, l'applicazione tenta in automatico di leggere tutti gli automi a stati finiti usati come modello comportamentale da file, se presenti. Inoltre, il sistema tenta di caricare una eventuale rete di FA comportamentali se in passato l'utente ne avesse generata una.

Dopodiché il sistema offre all'utente le seguenti scelte:

```
+-----+
|           Welcome to your project           |
+-----+

Please make a selection:
1) Create a BFA
2) Create a new BFA Network
3) Get results from the BFA Network
4) Show past Benchmarks
0) Go back

Enter your choice: █
```

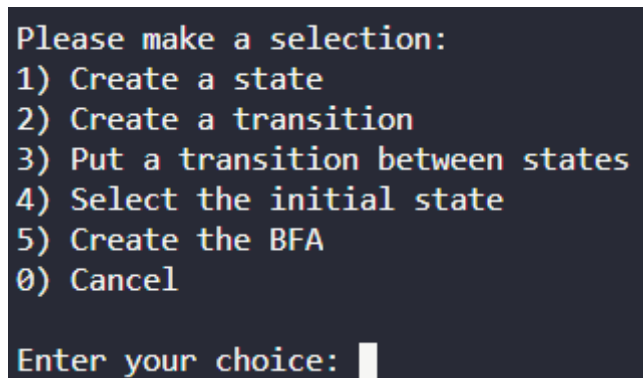
Questo menu propone all'utente la possibilità di:

- creare un nuovo automa a stati finiti usato come modello comportamentale (Behavioral Finite Automata) digitando 1 sulla tastiera.
- Creare una nuova rete di FA comportamentali (Behavioral Finite Automata Network) digitando 2 sulla tastiera. Se non è stata creata ancora alcun automa a stati finiti usato come modello comportamentale, allora il sistema annuncia all'utente l'impossibilità di creare una nuova rete di FA comportamentale, poiché una rete siffatta non sarebbe di alcuna utilità.
- Operare sulla rete di FA comportamentali attuale svolgendo alcune attività come il calcolo dello spazio comportamentale, la generazione dello spazio delle chiusure decorate, il calcolo della diagnosi relativa a un'osservazione lineare per mezzo del diagnosticare e tante altre funzionalità che saranno illustrare successivamente, digitando 3 sulla tastiera. Se il sistema non ha rilevato in precedenza alcuna rete di FA comportamentali e se l'utente non ne ha ancora creata una, l'applicazione segnala al fruitore la necessità di crearne una per poterne estrarre dei risultati.
- Mostrare i riassunti delle operazioni svolte passato dall'utente sulla rete di FA comportamentali, come il tipo di azione eseguita, il giorno e l'ora in cui ciò è accaduto e il tempo impiegato dal sistema per produrre il risultato, digitando 4 sulla tastiera. Nel caso in cui il fruitore non avesse ancora intrapreso alcuna operazione sulla rete, viene mostrato un messaggio che segnala la mancanza di riassunti disponibili.
- Tornare al menu precedente (relativo alla creazione o apertura di un progetto).

Menu di creazione di un automa a stati finiti usato come modello comportamentale

Quando l'utente decide di voler creare un nuovo BFA, innanzitutto gli viene richiesto di inserire un nome da attribuire all'automa. Se esiste già un BFA con tale nome, l'applicazione comunica questo fatto a schermo tramite un messaggio e riporta il fruitore sul menu del progetto corrente.

Quando il nome inserito è accettabile, appare il seguente menu:



```
Please make a selection:
1) Create a state
2) Create a transition
3) Put a transition between states
4) Select the initial state
5) Create the BFA
0) Cancel

Enter your choice: |
```

Questo menu consente di:

- generare un nuovo stato dell'automa digitando 1 sulla tastiera. Il sistema richiederà all'utente di inserire un nome da tastiera; se il nome inserito appartiene a uno stato già creato in precedenza, allora questa situazione sarà segnalata a video tramite un messaggio e la creazione dello stato verrà interrotta, riportando il fruitore al menu di creazione dell'automa.
- Creare una nuova transizione da aggiungere successivamente all'automa, digitando 2 sulla tastiera. In questo caso sullo schermo comparirà una sequenza di step relativi all'inserimento delle caratteristiche della transizione. Il primo step riguarda l'inserimento di un nome per la transizione; anche in questo caso, se il nome inserito è già stato assegnato a un'altra transizione generata in precedenza, l'applicazione segnala il conflitto e riporta l'utilizzatore del sistema al menu di creazione dell'automa. Il secondo step invece richiede di inserire una etichetta di osservabilità da associare alla transizione. Nel caso in cui non venga inserito nulla, il sistema assumerà che l'utente desideri associare alla transizione l'etichetta di osservabilità nulla ϵ . Il terzo step richiede l'inserimento di un'etichetta di rilevanza da associare alla transizione. Anche in questo caso, così come per l'etichetta di osservabilità,

nel caso in cui non venga inserito nulla, il sistema assumerà che l'utente desideri associare alla transizione l'etichetta di rilevanza nulla ϵ . Il quarto step prevede l'inserimento di un evento in ingresso per la transizione; premendo invio senza aver inserito alcun carattere, il sistema assumerà che l'utente desideri creare una transizione sprovvista di un evento in ingresso. L'ultimo step consente un inserimento ripetuto di eventi in uscita, fintantoché non venga premuto invio senza aver inserito alcun carattere. Giunti a tal punto, il processo di inserimento si arresterà, assegnando alla transizione un insieme (eventualmente vuoto) di eventi in uscita corrispondenti a quelli inseriti durante la procedura. Al termine di tutti e 5 gli step, l'applicazione genererà una transizione dotata delle caratteristiche specificate, la quale verrà immediatamente aggiunta tra quelle inseribili all'interno dell'automa. Un esempio di esecuzione di questa procedura è mostrato nella seguente figura:

```
Insert the name of the transition: t3a
Insert the transition observability label: o3
Insert the transition relevance label:
Insert the transition input event (press enter to skip this step):
Insert an output event to the transition (press enter to stop this process):e2(L2)
Insert an output event to the transition (press enter to stop this process):
```

Al termine del processo si viene rimandati al menu di creazione precedente di creazione dell'automa.

- Inserire una transizione precedentemente creata, digitando 3. Questa opzione permette all'utente di posizionare una transizione tra due stati (anche coincidenti), specificando prima lo stato sorgente e successivamente quello destinazione. Un esempio di scambi di interazione col sistema volti all'inserimento di una transizione di nome "t3a" uscente dallo stato "s1" entrante nello stato "s2" è il seguente:

```
Select the number of the transition:
0) t3a

Enter your choice: 0

Select the number of the source state:
0) s1
1) s2

Enter your choice: 0

Select the number of the destination state:
0) s1
1) s2
```

Inizialmente viene richiesto all'utente di selezionare di digitare il numero corrispondente a quello affiancato alla transizione che si desidera aggiungere nella rete. La lista di transizioni disponibili viene generata a partire dall'insieme di transizioni finora create manualmente dal fruitore. Nel caso in cui l'utente non abbia ancora creato alcuna transizione, il sistema stamperà a schermo un messaggio che invita alla creazione di almeno una transizione per poter usufruire dell'opzione, riportando l'utilizzatore al menu precedente relativo alla creazione dell'automa. Un'altra condizione necessaria è la presenza di almeno uno stato nella rete: per questo motivo, nel caso in cui non ne fosse presente alcuno, l'applicazione mostrerà un invito a crearne uno, riportando anche in questo caso il fruitore al menu di creazione dell'automa. Infine, vengono proposti due step per la selezione dello stato sorgente e dello stato destinazione in una modalità analoga a quella per la selezione della transizione nello step iniziale. Al termine del processo si viene rimandati al menu precedente di creazione dell'automa.

- Specificare uno stato iniziale dell'automa, digitando 4. Tramite questa opzione è possibile appunto impostare uno stato tra quelli finora creati come stato iniziale dell'automa. Nel caso in cui non si abbia ancora generato alcuno stato, verrà mostrato un messaggio che invita l'utente a crearne almeno uno. In tal caso l'utente verrà rimandato al menu precedente di creazione dell'automa.
- Creare un automa che rifletta le caratteristiche sinora imposte, digitando 5. Ciò permette di generare una BFA vera e propria, di aggiungerla tra quelle disponibili per formare una rete di FA comportamentali e di salvarla su file. Tutto ciò accade in un singolo step, in modo completamente trasparente. Affinché la creazione abbia successo, devono però essere soddisfatte tre condizioni: deve essere stato inserito almeno uno stato, deve essere stato impostato lo stato iniziale e non devono esistere stati isolati nella rete. Se anche una di queste condizioni non viene soddisfatta, la procedura viene annullata mostrando un generico messaggio d'errore e riportando l'utente al menu di creazione dell'automa. Quando invece la creazione avviene con successo, viene mostrato a schermo un riassunto dell'automa generato, ritornando poi automaticamente al menu del progetto corrente.
- Ritornare al menu precedente relativo al progetto corrente annullando la creazione, digitando 0.

Menu di creazione della rete di FA comportamentali

Se è stato creato almeno un automa a stati finiti usato come modello comportamentale, è possibile accedere al menu di creazione della rete di FA comportamentali. Tale menu offre le seguenti opzioni:

```
Please make a selection:
1) Create a Link
2) Put a Link between BFAs
3) Create the BFANetwork
0) Cancel

Enter your choice: █
```

- Creare un nuovo link digitando 1. Questa scelta porta alla generazione di un link che potrà essere inserito successivamente all'interno della rete di FA comportamentali che si vuole costruire, collegando due BFA. Per fare ciò viene richiesto all'utente di inserire un nome da assegnare al nuovo link; anche in questo caso, se viene inserito un nome che appartiene a un link generato in precedenza, la procedura viene annullata mostrando all'utente un messaggio che segnala il conflitto.
- Inserire un link creato precedentemente all'interno della rete, collegando due BFA, digitando 2. Non è possibile collegare una BFA con sé stessa tramite un link: ogniquale volta venga selezionata questa opzione senza aver generato almeno due BFA, viene mostrato un messaggio che invita il fruitore a creare almeno due automi a stati finiti comportamentali prima di procedere con l'inserimento del link. Se tale condizione è soddisfatta, allora verrà richiesto dapprima di selezionare il link che si vuole inserire e successivamente la BFA origine e quella destinazione. Un esempio di procedura in cui avvengono questi passi è la seguente, dove si desidera collegare l'automa "C2" con l'automa "C3" per mezzo del link "L2":

```
Select the number of the link:
0) L2

Enter your choice: 0

Select the number of the source bfa:
0) C2
1) C3

Enter your choice: 0

Select the number of the destination bfa:
0) C2
1) C3

Enter your choice: 1
```

- Creare una rete di FA comportamentali che rifletta le caratteristiche sinora imposte e salvarla su file (sovrascrivendo una eventuale rete già presente) digitando 3. Facendo questa scelta verrà mostrato a schermo un riassunto della rete generata, la quale sarà salvata automaticamente su file. Successivamente, il sistema mostrerà nuovamente il menu precedente relativo al progetto corrente, impostando la rete appena generata come rete attuale.
- Ritornare al menu precedente relativo al progetto corrente annullando la creazione, digitando 0.

Menu per operare sulla rete di FA comportamentali

Una volta creata o caricata una rete di FA comportamentali, è possibile accedere al seguente menu:

```
Please make a selection:
1) Compute Behavioral Space
2) Compute Decorated Silent Closure of a state
3) Compute Decorated Space of Closures
4) Compute Diagnostician
5) Change name of a State in the Behavioral Space
6) Change name of a Decorated Silent Closure
7) Create Linear Observation
8) Compute Behavioral Space related to a Linear Observation
9) Compute Linear Diagnosis with Diagnostician
0) Go back

Enter your choice: █
```

Le opzioni proposte permettono di:

- Calcolare lo spazio comportamentale della rete di FA, digitando 1. Così facendo, il sistema presenta la lista degli stati presenti nello spazio comportamentale, specificandone per ciascuno il nome, lo stato corrente di ciascuna BFA e il contenuto del buffer di ciascun link (al momento della creazione dello stato comportamentale) come nella seguente figura:

```

List of states in the behavioral space:
Name: 20 30 eps eps
STATES
    BFA: C3, state: 30
    BFA: C2, state: 20
LINKS
    Link: L2, event:  $\epsilon$ 
    Link: L3, event:  $\epsilon$ 

Name: 20 31 e2(L2) eps
STATES
    BFA: C3, state: 31
    BFA: C2, state: 20
LINKS
    Link: L2, event: e2(L2)
    Link: L3, event:  $\epsilon$ 

Name: 21 31 eps e3(L3)
STATES
    BFA: C3, state: 31
    BFA: C2, state: 21
LINKS
    Link: L2, event:  $\epsilon$ 
    Link: L3, event: e3(L3)

Name: 21 31 eps eps
STATES
    BFA: C3, state: 31
    BFA: C2, state: 21
LINKS
    Link: L2, event:  $\epsilon$ 
    Link: L3, event:  $\epsilon$ 

```

Subito dopo il sistema mostra la lista delle transizioni presenti nella rete, nella forma “*nome_stato_sorgente -> nome_transizione -> nome_stato_destinazione*”. Un esempio di output di transizioni è il seguente:

```

List of transitions in the behavioral space:
- 20 30 eps eps -> t3a -> 20 31 e2(L2) eps
- 20 31 e2(L2) eps -> t2a -> 21 31 eps e3(L3)
- 21 31 eps e3(L3) -> t3c -> 21 31 eps eps
- 21 31 eps e3(L3) -> t3b -> 21 30 eps eps
- 21 31 eps eps -> t2b -> 20 31 eps e3(L3)
- 20 31 eps e3(L3) -> t3c -> 20 31 eps eps
- 20 31 eps e3(L3) -> t3b -> 20 30 eps eps
- 21 30 eps eps -> t3a -> 21 31 e2(L2) eps
- 21 30 eps eps -> t2b -> 20 30 eps e3(L3)
- 20 30 eps e3(L3) -> t3a -> 20 31 e2(L2) e3(L3)
- 21 31 e2(L2) eps -> t2b -> 20 31 e2(L2) e3(L3)
- 20 31 e2(L2) e3(L3) -> t3c -> 20 31 e2(L2) eps
- 20 31 e2(L2) e3(L3) -> t3b -> 20 30 e2(L2) eps
- 20 30 e2(L2) eps -> t2a -> 21 30 eps e3(L3)
- 21 30 eps e3(L3) -> t3a -> 21 31 e2(L2) e3(L3)
- 21 31 e2(L2) e3(L3) -> t3c -> 21 31 e2(L2) eps

```


Infine, viene mostrato il nome dello stato iniziale e la lista dei nomi degli stati finali, come in questa figura:

```
Initial state of the behavioral space: 20 30 eps eps

List of final states of the behavioral space:
- 20 30 eps eps
- 20 31 eps eps
- 21 31 eps eps
- 21 30 eps eps
```

- Calcolare la chiusura decorata relativa a uno stato dello spazio comportamentale, digitando 2. L'applicazione mostra un menu di selezione proponendo tutti gli stati dello spazio comportamentali aventi in ingresso almeno una transizione osservabile assieme allo stato iniziale dello spazio, come nel seguente esempio:

```
Select an entry state:
0) 21 30 eps e3(L3)
1) 21 31 e2(L2) eps
2) 21 31 e2(L2) e3(L3)
3) 21 31 eps e3(L3)
4) 20 31 e2(L2) e3(L3)
5) 20 31 e2(L2) eps
6) 20 30 eps eps

Enter your choice: █
```

Dopo aver specificato da tastiera il numero corrispondente allo stato del quale si vuole calcolare la chiusura silenziosa decorata, il sistema mostra la chiusura stessa, in un formato di output del tutto analogo a quello dello spazio comportamentale.

- Costruire lo spazio delle chiusure decorato, digitando 3. Questa scelta permette di calcolare e visualizzare l'intero spazio delle chiusure silenziose decorate: in particolare il sistema mostra la lista dei nomi di ciascuna chiusura, la lista delle transizioni all'interno dello spazio nel formato *"nome_chiusura_sorgente -> nome transizione -> nome_chiusura_destinazione"*, il nome della chiusura iniziale e la lista dei nomi delle chiusure di accettazione (chiusure costituite da almeno uno stato finale dello spazio comportamentale). Un esempio di un possibile output è il seguente:

```
List of closures:
- x5
- x6
- x4
- x2
- x0
- x1
- x3

List of transitions:
- x5 -> t3a -> x6
- x4 -> t2a -> x2
- x4 -> t2a -> x5
- x6 -> t2a -> x5
- x6 -> t2a -> x2
- x0 -> t3a -> x1
- x2 -> t3a -> x1
- x2 -> t3a -> x4
- x2 -> t3a -> x3
- x3 -> t2a -> x5
- x3 -> t2a -> x2
- x1 -> t2a -> x2

Initial closure: x0

List of acceptance closures:
- x2
- x0
```

- Calcolare e visualizzare il diagnosticatore, digitando 4. Il formato di output del diagnosticatore è del tutto analogo a quello dello spazio delle chiusure silenziose decorate.
- Modificare il nome di uno stato appartenente allo spazio comportamentale, digitando 5. Così facendo, l'applicazione mostra la lista di tutti gli stati presenti per poter selezionare quello su cui si vuole operare, come nella seguente figura:

```
Select a state:
0) 20 30 eps eps
1) 20 31 e2(L2) eps
2) 21 31 eps e3(L3)
3) 21 31 eps eps
4) 21 30 eps eps
5) 20 31 eps e3(L3)
6) 20 31 eps eps
7) 21 31 e2(L2) eps
8) 20 30 eps e3(L3)
9) 20 31 e2(L2) e3(L3)
10) 20 30 e2(L2) eps
11) 21 30 eps e3(L3)
12) 21 31 e2(L2) e3(L3)

Enter your choice: 0
```

Dopodiché viene richiesto l'inserimento del nuovo nome da assegnare allo stato. Se il nuovo nome appartiene già a uno degli altri stati, l'applicazione segnala questo conflitto e impedisce la modifica. Subito dopo viene chiesto al fruitore se desidera continuare a modificare i nomi degli stati, digitando 1, oppure se desidera tornare al menu precedente, digitando 0.

- Modificare il nome di una chiusura presente all'interno dello spazio delle chiusure silenziose decorate, digitando 6. L'interazione con l'utente è del tutto analoga a quella relativa all'opzione precedente. Anche in questo caso, se si modifica il nome della chiusura tentando di assegnare un nome già appartenente ad un'altra chiusura, il sistema mostra il conflitto e impedisce la modifica.
- Aggiungere una nuova osservazione lineare, digitando 7. Questa opzione permette di inserire una nuova osservazione lineare da poter utilizzare nelle opzioni successive. Per fare ciò il sistema richiede ripetutamente all'utente di selezionare una etichetta di osservabilità, tra tutte quelle che sono state associate alle transizioni appartenenti alle BFA della rete. Questa procedura permette di generare una lista di etichette di osservabilità ordinata in base all'istante di inserimento di ciascuna etichetta. Per fermare la creazione e confermare la creazione dell'osservazione lineare attuale è sufficiente digitare 0. Se la nuova osservazione lineare così creata è vuota, oppure se è già stata creata un'osservazione identica in passato, l'applicazione mostra un messaggio che enuncia ciascuno di questi errori e riporta l'utente

al menu precedente. Un esempio di sequenza di passi volta alla creazione di una osservazione lineare è illustrato nella seguente immagine:

```
Current Linear Observation: []
Select the number of the observability label you want to add (type 0 to stop):
1) eps
2) o2
3) o3

Enter your choice: 3

Current Linear Observation: [o3]
Select the number of the observability label you want to add (type 0 to stop):
1) eps
2) o2
3) o3

Enter your choice: 2

Current Linear Observation: [o3, o2]
Select the number of the observability label you want to add (type 0 to stop):
1) eps
2) o2
3) o3

Enter your choice: 0
```

- Calcolare lo spazio comportamentale relativo a una osservazione lineare, digitando 8. Il sistema richiede innanzitutto di specificare quale osservazione lineare si desidera considerare, proponendo un menu costituito da tutte le osservazioni finora generate. Se non è stata creata alcuna osservazione, l'applicazione inviterà l'utente a crearne almeno una prima di poter procedere con il calcolo. Una volta selezionata l'osservazione, verrà mostrato a schermo lo spazio comportamentale risultante, in un formato di output del tutto analogo a quello per il calcolo dello spazio comportamentale illustrato nell'opzione 1. Infine, l'applicazione calcola la diagnosi lineare di questo spazio, mostrando il risultato in output. Se l'osservazione lineare considerata non è inerente ad alcuna traiettoria della rete, l'utente viene avvertito di questo fatto con un messaggio.
- Calcolare la diagnosi di una osservazione lineare attraverso il diagnosticare della rete, digitando 9. In questo caso, verrà richiesto di selezionare una osservazione lineare secondo la stessa procedura descritta per l'opzione precedente. Successivamente il sistema produrrà la diagnosi calcolata tramite il diagnosticare se l'osservazione è inerente ad almeno una traiettoria della rete. Se non è così, allora l'utente viene avvertito con un messaggio.

Sperimentazione

Verranno ora descritti alcuni dei risultati relativi alle sperimentazioni effettuate. Per ciascuno degli esempi proposti, è stato creato un progetto che può essere scelto utilizzando l'applicazione, come descritto quando si è parlato del [menù iniziale](#).

Esempio della consegna

Tale esempio consiste nella rete di FA comportamentali di pagina 26 della consegna del progetto, e di tutti i risultati che da essa possono essere ottenuti (spazio comportamentale, diagnosi, diagnosticatore...). Tutte le operazioni effettuabili su questa rete sono state testate nel dettaglio in fase di implementazione, che, [come abbiamo detto](#), ha seguito un approccio di tipo test-driven development. Tutti i test possono essere consultati nelle apposite classi situate all'interno della directory test/java/.

Questa rete può anche essere testata utilizzando l'applicazione e scegliendo il progetto già esistente chiamato "test". Al suo interno può essere trovata la rete, così come alcuni benchmark relativi alle nostre operazioni.

Mostreremo ora i tempi richiesti dagli algoritmi per ottenere i risultati e la memoria utilizzata per memorizzare le varie strutture dati. Questi valori sono stati ottenuti facendo una media su 1000 esecuzioni di ciascun algoritmo.

Il calcolo dello spazio comportamentale ha richiesto in media 0.24 ms con una deviazione standard sui dati di 2.62 ns circa. Per quanto riguarda la memoria necessaria per l'allocazione dello spazio comportamentale, l'applicazione ha richiesto in media circa 16 KB.

Il calcolo dello spazio delle chiusure silenziose decorate ha richiesto in media 0.70 ms con una deviazione standard sui dati di 7.99 ns circa. Per quanto riguarda la memoria necessaria per l'allocazione dello spazio delle chiusure silenziose decorate, l'applicazione ha richiesto in media circa 15 KB.

Infine, il calcolo del diagnosticatore ha richiesto in media 0.02 ms con una deviazione standard sui dati di 0.15 ns circa. Per quanto riguarda la memoria necessaria per l'allocazione del diagnosticatore, l'applicazione ha richiesto in media circa 4 KB.

Per quanto riguarda la diagnosi relativa all'osservazione lineare [o3, o2], il tempo medio richiesto dall'algoritmo che costruisce lo spazio comportamentale relativo all'osservazione per calcolare la diagnosi è stato pari all'incirca a 0.53 ms, mentre usando l'algoritmo che si appoggia sul diagnosticatore, il tempo medio richiesto è stato di 0.01 ms. Si nota dunque una chiara riduzione del tempo medio richiesto: molto probabilmente ciò è dovuto al fatto che il primo algoritmo ha la necessità di creare uno spazio comportamentale prima di eseguire l'algoritmo espressioni regolari, mentre il secondo si appoggia su un automa già costruito, ovvero il diagnosticatore, per produrre la diagnosi.

Secondo esempio della consegna

L'esempio benchmark consiste nella rete di FA comportamentali di pagina 73 della consegna del progetto, e di tutti i risultati che da essa possono essere ottenuti (spazio comportamentale, diagnosi, diagnosticatore...). Il progetto contenente questa rete di BFA comportamentali e i rispettivi BFA è il progetto "Network2".

Mostreremo anche in questo caso i tempi richiesti dagli algoritmi per ottenere i risultati e la memoria utilizzata per memorizzare le varie strutture dati. Questi valori sono stati ottenuti facendo una media su 1000 esecuzioni di ciascun algoritmo.

Il calcolo dello spazio comportamentale ha richiesto in media 0.21 ms con una deviazione standard sui dati di 0.37 ns circa. Per quanto riguarda la memoria necessaria per l'allocazione dello spazio comportamentale, l'applicazione ha richiesto in media circa 11 KB.

Il calcolo dello spazio delle chiusure silenziose decorate ha richiesto in media 0.97 ms con una deviazione standard sui dati di 2,38 ns circa. Per quanto riguarda la memoria necessaria per l'allocazione dello spazio delle chiusure silenziose decorate, l'applicazione ha richiesto in media circa 17 KB.

Infine, il calcolo del diagnosticatore ha richiesto in media 0.02 ms con una deviazione standard sui dati di 0.02 ns circa. Per quanto riguarda la memoria necessaria per l'allocazione del diagnosticatore, l'applicazione ha richiesto in media circa 6 KB.

Per quanto riguarda la diagnosi relativa all'osservazione lineare [act, sby, nop], il tempo medio richiesto dall'algoritmo che costruisce lo spazio comportamentale relativo all'osservazione per

calcolare la diagnosi è stato pari all'incirca a 0.71 ms, mentre usando l'algoritmo che si appoggia sul diagnosticatore, il tempo medio richiesto è stato di 0.02 ms. Anche in questo caso si nota una chiara riduzione del tempo medio richiesto.

Esempio benchmark

L'esempio benchmark consiste nella rete di FA comportamentali di pagina 93 della consegna del progetto, e di tutti i risultati che da essa possono essere ottenuti. Il progetto contenente questa rete di BFA comportamentali e i rispettivi BFA è il progetto "esempio_benchmark".

Come negli esempi precedenti, illustriamo ora i tempi richiesti dagli algoritmi per ottenere i risultati e la memoria utilizzata per memorizzare le varie strutture dati. Questi valori sono stati ottenuti facendo una media su 1000 esecuzioni di ciascun algoritmo.

Il calcolo dello spazio comportamentale ha richiesto in media 0.22 ms con una deviazione standard sui dati di 0.59 ns circa. Per quanto riguarda la memoria necessaria per l'allocazione dello spazio comportamentale, l'applicazione ha richiesto in media circa 8 KB.

Il calcolo dello spazio delle chiusure silenziose decorate ha richiesto in media 0.23 ms con una deviazione standard sui dati di 0.46 ns circa. Per quanto riguarda la memoria necessaria per l'allocazione dello spazio delle chiusure decorate silenziose, l'applicazione ha richiesto in media circa 4 KB.

Infine, il calcolo del diagnosticatore ha richiesto in media 0.002 ms con una deviazione standard sui dati di 0.0005 ns circa. Per quanto riguarda la memoria necessaria per l'allocazione del diagnosticatore, l'applicazione ha richiesto in media circa 1 KB.

Esempio creato dal gruppo

Di seguito sono mostrati i due FA comportamentali B1 e B2 da noi creati. Il contenuto delle transizioni in essi contenute è mostrato successivamente nella tabella delle transizioni, così come le rispettive etichette di osservabilità e rilevanza. I due BFA sono rappresentati in Figura 6.

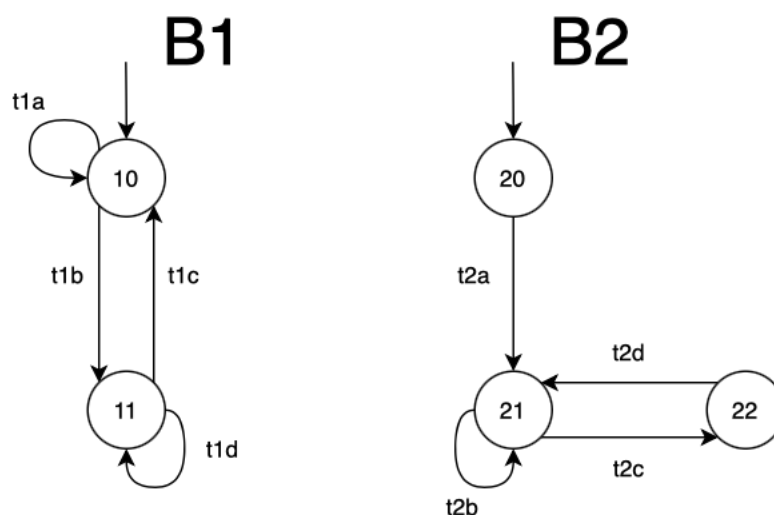


Figura 6: FA comportamentali che costituiscono la rete di BFA.

I due FA comportamentali sono interconnessi tra di loro all'interno della rete mostrata in Figura 7.

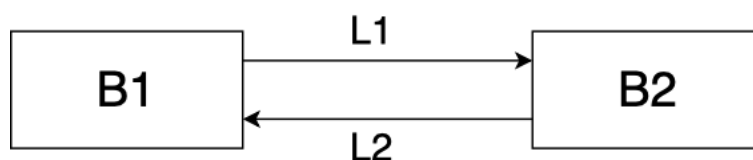


Figura 7: Rete di FA comportamentali creata dal gruppo di lavoro.

Il contenuto delle transizioni, così come le loro etichette di osservabilità e rilevanza è mostrato nelle seguenti tabelle:

Transizioni	
B1	B2
t1a /{e1(L1)}	t2a e1(L1)
t1b e1(L2)	t2b e1(L1)/{e1(L2)}
t1c	t2c /{e2(L2)}
t1d e2(L2)/{e2(L1)}	t2d /{e1(L2)}

Osservabilità	
B1	B2

t1a z1	t2a z2
t1b z2	t2b
t1c z3	t2c
t1d	t2d
Rilevanza	
B1	B2
t1a h	t2a
t1b	t2b
t1c h	t2c
t1d w	t2d w

Il calcolo dello spazio comportamentale ha richiesto in media 0.71 ms con una deviazione standard sui dati di 0.13 ns circa. Per quanto riguarda la memoria necessaria per l'allocazione dello spazio comportamentale, l'applicazione ha richiesto in media circa 6 KB.

Il calcolo dello spazio delle chiusure silenziose decorate ha richiesto in media 0.59 ms con una deviazione standard sui dati di 0.15 ns circa. Per quanto riguarda la memoria necessaria per l'allocazione dello spazio delle chiusure decorate silenziose, l'applicazione ha richiesto in media circa 7 KB.

Infine, il calcolo del diagnosticatore ha richiesto in media 0.03 ms con una deviazione standard sui dati di 0.02 ns circa. Per quanto riguarda la memoria necessaria per l'allocazione del diagnosticatore, l'applicazione ha richiesto in media circa 4 KB.

Per quanto riguarda la diagnosi relativa all'osservazione lineare $[z1, z2]$, il tempo medio richiesto dall'algoritmo che costruisce lo spazio comportamentale relativo all'osservazione per calcolare la diagnosi è stato pari all'incirca a 0.64 ms, mentre usando l'algoritmo che si appoggia sul diagnosticatore, il tempo medio richiesto è stato di 0.03 ms. Anche in questo caso si nota una chiara riduzione del tempo medio richiesto.