Università degli Studi di Verona

Corso di laurea in ingegneria
e scienze informatiche

Progetto di Verifica Automatica di Sistemi

Destinato a:
Prof. Luca Geretti
Prof. Tiziano Villa
Dipartimento di
Informatica

Studente :
Raffaello Corsini
MAT: VR400386

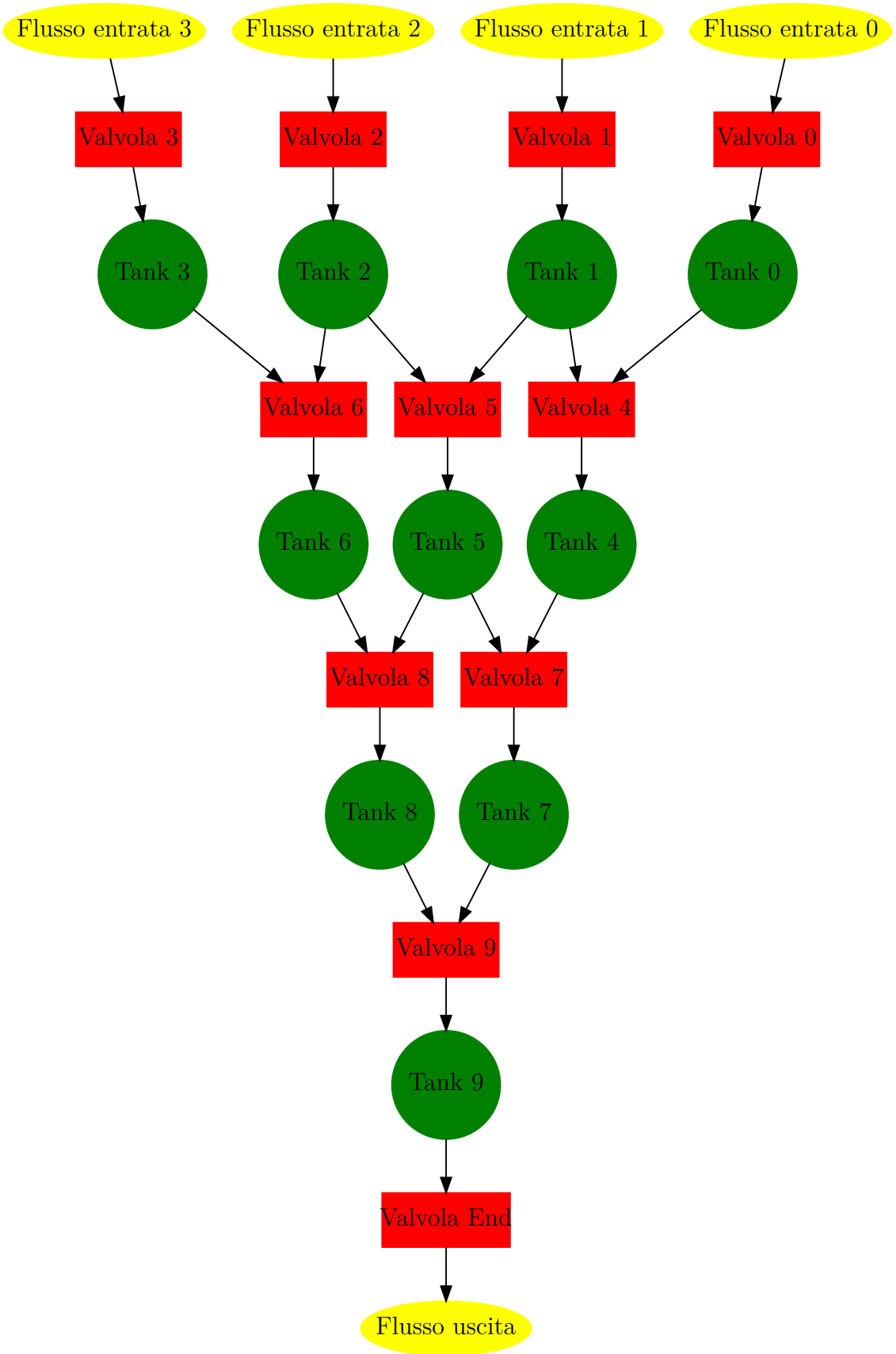
Indice

Sommario

Il seguente testo tratta di un progetto che si basa sulla libreria *Ariadne* per C++ per descrivere l'evoluzione di automi ibridi. Verranno presentate una serie di cisterne d'acqua (d'ora in poi *tank*) impilate a piramide inversa, con due tank in alto e una in basso. Nel progetto ci saranno alcune semplificazioni dell'ambiente reale per agevolare la rappresentazione. Verrà descritta l'idea di partenza, la sua concretizzazione e la realizzazione passo passo, con particolare enfasi sulle criticità superate.

1 Idea di partenza

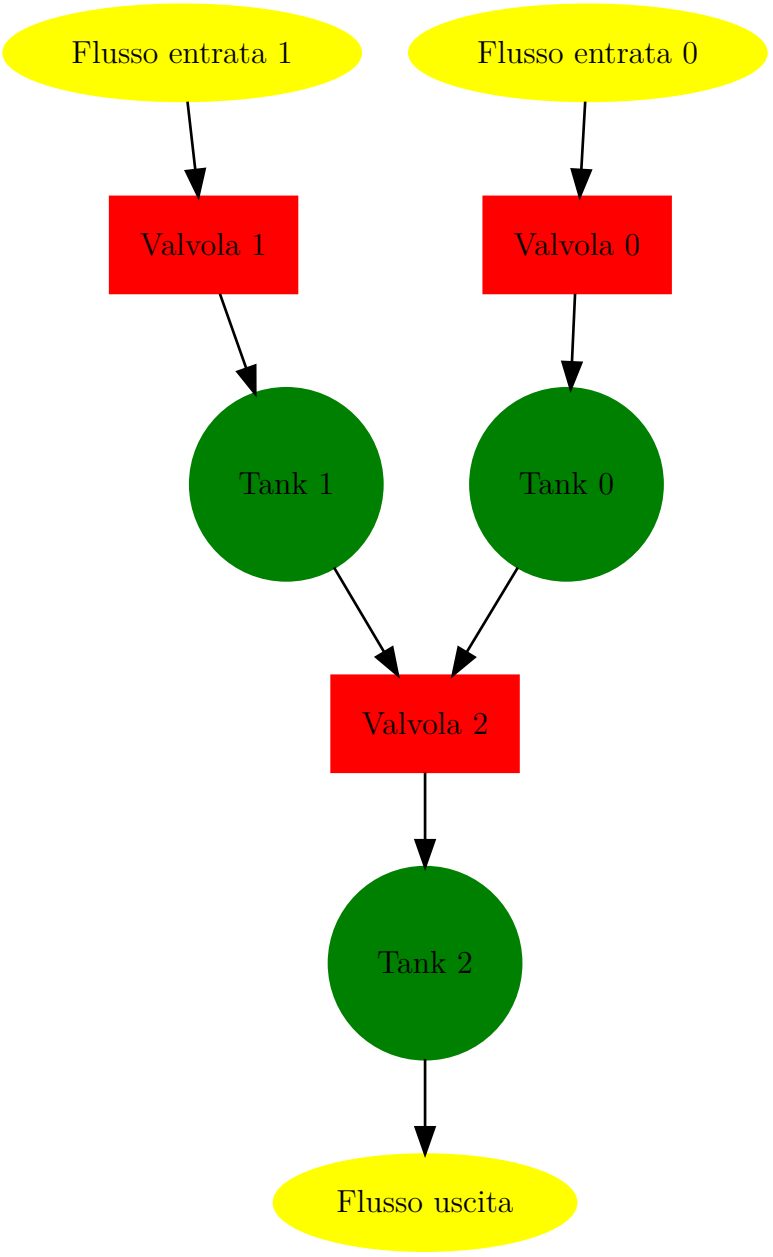
L'idea da cui si è partiti era quella di modellare un sistema che fosse composto da dieci tank e undici valvole, disposte a piramide inversa. Le prime quattro tank in alto avrebbero avuto ciascuna un flusso d'acqua esterno costante in entrata, mentre l'unica tank in basso avrebbe perso l'acqua uscente fuori dal sistema. A seconda del posto occupato nella piramide i flussi d'entrata e d'uscita delle varie tank avrebbero dovuto essere disposti secondo lo schema della pagina seguente. Inoltre ogni tank avrebbe avuto una valvola sopra di essa, comandata da un controllore, che si sarebbe chiusa al superamento di una certa soglia da parte dell'acqua contenuta. Le valvole avrebbero dovuto chiudersi nel caso venisse superata una certa quantità d'acqua contenuta e riaprirsi quando questa fosse tornata sotto la soglia desiderata. Per finire ci sarebbe stata un'ultima valvola posta sotto all'ultima tank, che nelle intenzioni si sarebbe dovuta aprire solo quando la quantità d'acqua presente nella tank soprastante avesse superato una certa soglia, per poi richiudersi una volta tornati sotto di essa.


Flusso entrata 3
Flusso entrata 2
Flusso entrata 1
Flusso entrata 0
Valvola 3
Valvola 2
Valvola 1
Valvola 0
Tank 3
Tank 2
Tank 1
Tank 0
Valvola 6
Valvola 5
Valvola 4
Tank 6
Tank 5
Tank 4
Valvola 8
Valvola 7
Tank 8
Tank 7
Valvola 9
Tank 9
Valvola End
Flusso uscita

2 Concretizzazione dell'idea di partenza

La creazione di un sistema come quello ideato, sebbene possibile tramite *Ariadne*, sarebbe stata poco utile perché non in grado di produrre risultati in tempo utile. Questo sistema avrebbe avuto ventuno variabili (dieci livelli d'acqua e undici livelli di apertura di valvola) e un numero molto grande di locazioni e transizioni, che avrebbero portato all'impossibilità di effettuare analisi sul sistema che convergessero in un tempo accettabile. [1] Per questo si è deciso di partire da un sistema più semplice che risultasse più facile da scrivere e da analizzare. Il sistema che si è quindi deciso di produrre è composto da tre tank, tre valvole e tre controllori. Le tank sono disposte in verticale a triangolo, con la punta rivolta verso il basso. Le due tank superiori hanno due ingressi regolati da valvole e ciò che esce da queste entra nella tank inferiore, anche qui attraverso una valvola. Ogni valvola è controllata da un controllore personale, che mantiene monitorato il livello d'acqua nella tank sottostante. Questo fa si che una valvola venga chiusa nel caso l'acqua contenuta nella tank sottostante superi una certa soglia, per poi invece riaprirla nel momento in cui il livello di riempimento torni al di sotto di un altro valore dato. L'acqua che esce dalla tank inferiore viene persa dal sistema.

[1] Per convergenza in tempo accettabile si intende d'ora in poi che arrivi ad una soluzione in meno di mezz'ora. Questo tenendo presente che la macchina utilizzata per i test è un PC del 2012, con sistema operativo Ubuntu 16.04.4 LTS e processore Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz.


Flusso entrata 1
Flusso entrata 0
Valvola 1
Valvola 0
Tank 1
Tank 0
Valvola 2
Tank 2
Flusso uscita

3 Gestione del codice

Il codice è stato scritto a partire dal tutorial allegato alla libreria *Ariadne*. Questo simulava una singola unità tank-valvola-controllore, descritto interamente nel file `system.h`. Ognuno di questo componenti veniva creato esplicitamente[2] all'interno del file per poi essere alla fine unito in un unico automa, restituito dall'unica funzione presente. Questo codice era diviso su tre file:

- `system.h` contenente la descrizione dei vari automi e della loro composizone;
- `analysis.h` contenente la descrizione dell'evoluzione del sistema e le varie analisi da eseguire;
- `project.cc` contenente il main che crea gli automi descritti in `system.h` ed esegue le analisi descritte in `analysis.h`.

Inizialmente sono stati aggiunti al file `system.h` dei comandi per creare esplicitamente una per una le variabili, le tank, le valvole, i controllori e comporre tutti gli automi nell'automa risultante che la funzione `getSystem()` restituisce. In seguito, per ragioni di praticità e scalabilità del codice si è deciso di optare per un approccio modulare. Ciò è consistito nel creare quattro file aggiuntivi.

- `side_tank.h` rappresentante una delle due tank poste al livello superiore;
- `bottom_tank.h` rappresentante la tank al livello inferiore;
- `valve.h` rappresentante una valvola;
- `controller.h` rappresentante un controllore.

In ogni file è stata scritta una funzione che in base a dei parametri forniti restituisse un oggetto di tipo `HybridIOAutomaton` con le caratteristiche desiderate. Questo ha permesso di creare automi dello stesso tipo tramite queste funzioni, quindi tramite l'invocazione con parametri adeguati al posto di dover scrivere esplicitamente ogni passaggio della creazione dell'automa. Questo torna particolarmente utile quando si presenta la necessità di produrre molti automi dello stesso tipo in serie, poiché questo può essere effettuato all'interno di un ciclo con l'ausilio di una variabile incrementale. In seguito gli automi cosi prodotti venivano composti esplicitamente nel file `system.h`, anche se questa risultava scomoda e difficilmente scalabile, in quanto per ogni nuovo automa creato che si fosse voluto aggiungere al sistema era necessario richiamare esplicitamente la composizione e fornire automi e locazioni iniziali. Per questo è stato scelto di aggiungere un nuovo file dal nome

- `automaton-composition.h`.

[2]Per creazione (o composizione) esplicita da qui in poi intendiamo che questo avviene passo dopo passo per ogni elemento, invece che essere automatizzato tramite funzioni o cicli.

Questo file conteneva inizialmente diverse funzioni. Le principali erano tre funzioni differenti per comporre dei vettori di automi. Inoltre erano presenti due funzioni ausiliarie. La prima aveva il compito di concatenare tre vettori di automi restituendone uno unico. Questo perché inizialmente gli automi venivano salvati in tre diversi vettori a seconda che fossero tank, valvole o controllori. La seconda funzione invece era stata inserita per migliorare la leggibilità della stampa a video degli automi, sostituendo i "`, `" con "`\n`". Ciò è stato fatto perché la stampa a video di un automa consisteva in un'unica riga di testo, rendendone difficile l'analisi e la comparazione. Andando a sostituire le virgole seguite da spazio con dei ritorni a capo invece il risultato si mostrava maggiormente comprensibile e utilizzabile. Si è scelto di usare come stringa da sostituire "`, `" perché la sola virgola è un carattere utilizzato per la composizione delle locazioni. Questo ci avrebbe portato ad un eccessivo numero di ritorni a capo con la conclusione che avremmo avuto un file con più righe che non ci avrebbe permesso un confronto rapido e agevole. In seguito alla riscrittura finale del codice sono state rimosse le funzioni di composizione non utilizzate e la funzione di concatenazione dei vettori, visto che dopo la scelta di salvare direttamente gli automi creati in un unico vettore non era più necessaria. È stata mantenuta invece la funzione che agevola la lettura della stampa di un automa perché potrebbe ancora risultare utile in eventuali prossime evoluzioni del progetto.

Infine è stato aggiunto al progetto è stato aggiunto un nuovo file dal nome

- `urgent-controller.h`.

che permetteva la creazione di un controllore con transizioni urgenti, eseguite immediatamente non appena si fosse raggiunta o oltrepassata una certa soglia.

L'ultima modifica effettuata è stato decidere di memorizzare la locazione di partenza di ogni automa creato, in maniera tale che in coppia con lo stesso automa venisse passata alla funzione di composizione. Questo per far sì che non si vadano mai a creare alla fine della composizione degli automi con locazioni irraggiungibili come locazioni iniziali.

4 Monitoraggio creazione automi e analisi di similarità

Durante la fase di scrittura e riscrittura del codice è stato necessario poter avere sempre sottocchio i sistemi creati in maniera tale da poter evidenziare il prima possibile eventuali anomalie. Questo sia per quanto riguarda automi creati ex-novo, per verificare che corrispondano a quanto desiderato, sia per quanto riguarda automi modificati o sostituiti con nuove versioni, per verificare che corrispondano alla versione precedente.

La libreria *Ariadne* non fornisce un sistema di comparazione fra due automi. Per questa ragione durante l'evoluzione del progetto si è sviluppato una sorta di protocollo per verificare che ogni modifica negli automi corrispondesse a quanto desiderato. Nel caso di automi creati ex novo questo è servito per verificarne la corretta creazione e un funzionamento plausibile, mentre nel caso di automi aggiornati o di metodi nuovi per la creazione degli stessi automi l'utilità è consistita nel poter verificare che i nuovi fossero identici ai precedenti.

Questo protocollo consisteva nell'analizzare la stampa testuale e i plot delle analisi effettuate, confrontandoli quando necessario con quelli prodotti dalle versioni meno recenti. Nel secondo caso se stampe e plot risultano identici a quelli delle versioni precedenti (con parametri identici) è ragionevole supporre che pure gli automi siano equivalenti. Le stampe testuali venivano fatte dal programma su file diversi a seconda dell'automa, con l'ausilio della funzione `easy_read_automaton` contenuta in `automaton-composition.h`. In questa maniera si ottenevano dei file di testo con la stampa degli automi desiderati. Inizialmente è stato usato il tool `diff` di *Linux* per confrontare i file di testo generati (talvolta in combinazione con `sort`), dopodiché si è passati a `Meld`, a causa delle maggiori funzionalità offerte e dell'interfaccia grafica, che agevolava l'individuazione delle righe difformi.

Questo tipo di monitoraggio è stato eseguito sia all'inizio, quando sono stati creati ex novo ed esplicitamente i nuovi componenti, sia in seguito quando è stata applicata la modularizzazione e la composizione sequenziale di una serie di automi. Nello specifico, per quanto riguarda la composizione degli automi, l'analisi testuale è stata utile per verificare che i primi due metodi del file `automaton-composition.h` producevano lo stesso automa, così come il terzo metodo ne produceva uno simile a quello ottenuto eseguendo la composizione esplicita all'interno del file `system.h`. Il confronto fra la stampa testuale dei due automi differenti, con controprova l'analisi dei grafici delle analisi, ha mostrato come fossero anche questi uguali, con la differenza che stava solo nel nome e nell'ordine delle locazioni. Ciò ha permesso di mantenere solo il secondo metodo introdotto per la composizione degli automi utilizzandolo nel file `system.h` al posto della composizione esplicita. Questo metodo è stato scelto perché dei tre provati era quello più compatto e di più facile lettura.

5 Monitoraggio analisi

Dapprima si sono eseguite diverse analisi mantenendo i vari parametri uguali alla tank del tutorial, per poter effettuare controlli con gli automi precedenti atti a scoprire eventuali anomalie. Inoltre i parametri passati come `Box` di inizializzazione sono stati scelti all'inizio e poi mantenuti costanti. In seguito sono stati modificati i parametri dei flussi iniziali e del delta per provare ad effettuare le analisi desiderate. Si è iniziato con la *finite time upper evolution* e la *finite time lower evolution*, notan-

do come non sempre il numero di traiettorie convergesse durante l'analisi. Spesso, nonostante venisse aumentato il passo d'integrazione e ridotti il tempo e il numero di eventi, l'analisi non convergeva in un lasso di tempo ragionevole. Per ovviare al problema si è provveduto ad impostare un delta molto piccolo. In questa maniera rimaneva presente un certo nondeterminismo sufficiente a permettere le transizioni non urgenti ma allo stesso tempo le analisi convergevano in un tempo ragionevole.

In seguito, una volta passati alla versione modulare, i parametri sono stati mantenuti uguali per poter effettuare dei confronti con le versioni precedenti. Sono stati modificati solamente in occasione dell'introduzione del controllore urgente, perché in quel caso un comportamento troppo conservativo delle librerie dava luogo a una certa incertezza nel sistema. Per risolverlo è stato reso necessario modificare il valore di inizializzazione di una tank affinché venisse simulato un comportamento corretto.

Da ultimo si è provveduto ad effettuare le analisi di *infinite time outer evolution* e e *infinite time lower evolution*. Si è notato che queste analisi per il valore di *accuracy* preimpostato (5) non riuscivano ad arrivare ad una soluzione in tempo ragionevole. Per la lower evolution si raggiungeva una soluzione impostando l'accuracy a 2, mentre per l'outer evolution nemmeno abbassare l'accuracy ad 1 ha portato ad una soluzione.

6 Monitoraggio versionamento e utilizzo di Github

Dato che questo progetto si prestava a continue riscritture del codice e aggiornamenti, e visto che il Prof. Geretti lo considerava utile, si è scelto di utilizzare un repository *git* per tenere traccia delle varie versioni e delle varie modifiche. Come hosting è stato scelto *Github*, per la gratuità e la semplicità d'interfacciamento. Dapprima è stato creato e aggiornato un repository chiamato *waterworld*[3], aggiornato costantemente. Questo è il codice che è stato usato per le prime esecuzioni delle analisi e in seguito è stato a suo tempo creato anche un nuovo branch, chiamato "invariant_initally_active", per salvare e rivedere meglio certe analisi che non raggiungevano una soluzione in un tempo soddisfacente. dubbi. Poi, una volta passati alla versione modulare del progetto, si è scelto di passare ad un nuovo repository chiamato *waterworld-modular*[4], che è quello poi mantenuto aggiornato fino alla versione finale del progetto. I file aggiunti al repository nella prima versione erano i tre componenti il programma più il file di configurazione per *cmake*. Nella versione modulare sono stati aggiunti i nuovi file creati per i vari componenti e per la com-

[3] `http://www.github.com/raffaello-corsini/waterworld`
[4] `http://www.github.com/raffaello-corsini/waterworld-modular`

posizione, per aggiungere una volta terminato il lavoro anche la presente relazione come documentazione del progetto.

7 Conclusioni finali e prospettive future

È stato creato un automa ottenuto dalla composizione di nove automi e che lavora su sei variabili. Già questo livello di complessità ci fa fatto notare come, nel caso di transizioni non urgenti, la quantità di non determinismo sia tale da portare spesso il sistema a non trovare soluzione in tempo ragionevole, anche nel caso di delta molto piccolo. Questo ci suggerisce che si possono ideare sistemi composti da più automi, ma a fronte delle capacità di calcolo di un normale personal computer la maggior parte delle analisi eseguibili da Ariadne potrebbero non terminare a meno di essere a tempo finito e con una terminazione delle tracce fissata molto presto o con un passo d'integrazione molto grande.

Inoltre sono stati scoperti una serie di comportamenti anomali o troppo conservativi nelle librerie, dove i manutentori dovranno scegliere se mantenerli o modificarli. Ci si è anche trovati di fronte a dei messaggi di errore poco chiari, che potrebbero essere esplicitati meglio nelle prossime versioni.

In linea teorica però, grazie alla modularizzazione, si potrebbero scrivere altri pezzi ed arrivare a sistemi come quello ipotizzato inizialmente o anche più grandi. Sarebbe necessario scrivere solamente altre tre classi per tre diversi tipi di tank e strutturare la composizione di conseguenza. Una delle ipotesi interessanti sarebbe che, una volta arricchito e automatizzato il sistema di composizione degli automi, si potrebbe scrivere una funzione che preso in input un intero `n` vada a creare una piramide rovesciata con lato `n`; al costo di una leggera semplificazione considerando identici caratteristiche e comportamenti di tank, valvole e controllori. Questo più che per esercizi di analisi potrebbe essere utile come esercizio di stile, per mostrare un sistema potenzialmente molto complesso.

Inoltre questo codice potrebbe essere distribuito fra gli esempi forniti al momento dello scaricamento della libreria, visto che quelli forniti in genere non componevano più di tre automi circa.

Per concludere potrebbe essere utile modificare i commenti nello stile richiesto da *Doxygen*, un tool utile per produrre della documentazione in maniera automatica a partire dal codice. Il codice è già riccamente commentato, ma un documento di questo tipo potrebbe aiutare eventuali soggetti terzi che volessero consultarlo.