

# Robots Cluster

Implementazione di un algoritmo di agreement  
per cluster di robots.

**Alessandro Mini - 7060381**

**Andrea Neri - 7060638**

**Mattia Bacci - 7060637**

Elaborato del corso di Software Dependability  
(9CFU)  
2021-2022



Magistrale di Ingegneria Informatica  
Università degli Studi di Firenze

## Indice

<b>1</b>	<b>Introduzione</b>	<b>2</b>
<b>2</b>	<b>Descrizione Problema</b>	<b>2</b>
2.1	Linear Two Phase-Commit Protocol . . . . .	3
<b>3</b>	<b>Implementazione</b>	<b>4</b>
3.1	Macchina a stati finiti "Coordinator" . . . . .	5
3.2	Macchina a stati finiti "Robot" . . . . .	7
3.3	Macchina a stati finiti "Tail" . . . . .	8
<b>4</b>	<b>Esempio di simulazione</b>	<b>9</b>
<b>5</b>	<b>Verifica Formale</b>	<b>11</b>
5.1	Proprietà di Agreement . . . . .	12
5.2	Proprietà di Liveness . . . . .	12
5.3	Proprietà di Validity . . . . .	13
5.4	Funzionamento della verifica . . . . .	13
5.4.1	Esempio di richiesta errata . . . . .	15
<b>6</b>	<b>Scalabilità</b>	<b>15</b>
6.1	Esempi di scalabilità . . . . .	16
<b>7</b>	<b>Coordinatore Multiplo</b>	<b>19</b>
<b>8</b>	<b>Failure</b>	<b>23</b>
8.1	Soluzione alternativa . . . . .	26
<b>9</b>	<b>Conclusioni</b>	<b>27</b>

## Elenco delle figure

1	Algoritmo Linear 2PC applicato a <i>Robots Cluster</i> . . . . .	4
2	Sistema Input/Output e sottosistema di verifica con le tre proprietà implementato in Simulink. . . . .	5
3	Robot (tra cui coordinator e Tail) implementati come charts (macchine a stati finiti estese) in grado di comunicare tra loro. . . . .	5
4	Macchina a Stati Finiti associata al Coordinatore . . . . .	7
5	Macchina a Stati Finiti associata al nodo Tail. . . . .	8
6	Macchina a Stati Finiti associata al Robot. . . . .	9
7	Visualizzazione delle variabili alla fine della simulazione, in questo caso si è supposto che $m > n$ , quindi $ACK = -1$ perchè di fatto non si può completare l'azione. . . . .	10
8	Circuito di implementazione della proprietà di Agreement. . . . .	12
9	Circuito di implementazione della proprietà di Liveness. . . . .	13
10	Circuito di implementazione della proprietà di Validity. . . . .	13
11	Sistema con tre verification subsystems, ovvero i sottosistemi di verifica delle proprietà sopracitati . . . . .	14
12	Le proprietà risultano verificate, i robot con indice 2,3 eseguono l'azione richiesta, $m = 2$ , pertanto il sistema conclude l'esecuzione con un acknowledgement positivo $ACK = 1$ . . . . .	14
13	Report di Matlab Design Verifier . . . . .	15
14	Configurazione del sistema nel caso sopracitato. . . . .	15
15	Stateflow nel caso di 5 Robot . . . . .	17
16	Modello per la verifica formale in presenza di 5 Robot e un numero di Robot necessari per portare a termine il task richiesto ( $m$ ) pari a 3 . . . . .	17
17	Stateflow nel caso di 7 Robot . . . . .	18
18	Modello per la verifica formale in presenza di 7 Robot e un numero di Robot necessari per portare a termine il task richiesto ( $m$ ) pari a 3 . . . . .	18
19	Individuazione Coordinatore e K-Means . . . . .	20
20	Verifica Formale con Coordinatori Multipli, si possono notare i due sottosistemi relativi ai due cluster (uno con 3 robots ed un coordinatore, l'altro con 7 robots ed un coordinatore), che prendono in ingresso azioni diverse. . . . .	21
21	Implementazione del sottosistema di verifica per proprietà <i>validity</i> in sistema con coordinatori multipli . . . . .	22
22	Implementazione del sottosistema di verifica per proprietà <i>liveness</i> in sistema con coordinatori multipli . . . . .	22
23	Modifica apportata al Robot Coordinatore in modo che riuscisse a gestire i due casi di Failure descritti precedentemente. . . . .	24
24	Modello modificato in modo da generare una situazione di Failure, rimuovendo tutti i collegamenti tra il Robot $R_2$ e il Robot Tail, simulando così un guasto. . . . .	24
25	Valorizzazione dei simboli nello statechart. . . . .	25

## 1 Introduzione

In questo elaborato si affronta il problema del *Robot Cluster*, ovvero strutturare un comportamento cooperativo tra Robot per muovere oggetti pesanti.

Il lavoro si colloca nell'ambito dello studio dei sistemi distribuiti, nello specifico in questo problema si deve raggiungere l'*agreement* tra i robot, per realizzare una pool caratterizzata da una certa azione da terminare.

Il lavoro viene presentato come segue: nella sezione 2 viene descritto il problema, seguiranno poi l'implementazione (sezione 3), simulazione (sezione 4) per poi passare alla verifica formale e alcune estensioni (Sezioni 5,6,7).

## 2 Descrizione Problema

Sia  $\mathcal{S}$  un insieme di Robot ognuno di questi esegue una certa azione in modo cooperativo per spostare oggetti pesanti.

Fornita una certa azione mediante un *action\_id*  $\in \mathcal{D}$  con  $\mathcal{D}$  set di azioni disponibili, si applica la versione lineare dell'algoritmo 2PC(Two Phase-Commit Protocol) per raggiungere *agreement* e *consistenza* sull'azione da eseguire tra tutti i partecipanti (cioè i robots).

La struttura lineare dell'algoritmo impone che i robot siano numerati  $R_0, \dots R_{n-1}$  e quindi che il Robot  $R_i$  comunichi soltanto con i robot  $R_{i+1}, R_{i-1}$ .

Nella catena che quindi si viene a generare viene eletto un nodo al ruolo di *coordinatore*, si adottano le seguenti assunzioni per semplificare la formalizzazione:

1. Per completare il task richiesto si assume che almeno  $m$  robot degli  $n$  totali si accordino tra di loro.
2. Ogni Robot può scegliere se aggregarsi o meno al gruppo di lavoro in base a una qualche politica di decisione (es. distanza dall'oggetto, batteria ecc.)
3. I messaggi vengono inviati e ricevuti sempre con successo.
4. Si considera solo l'implementazione del protocollo di agreement e non l'esecuzione dell'azione.
5. non si modellano le ragioni descritte al punto (2).
6. Il coordinatore, almeno all'inizio, è il Robot <sup>1</sup> con indice 0.

---

<sup>1</sup> Nella soluzione che verrà presentata il coordinatore è di fatto un Robot, cioè esegue un azione.

## 2.1 Linear Two Phase-Commit Protocol

Come algoritmo di *agreement* si è implementato, come sopracitato, il Linear 2PC, il quale prevede che:

1. Ogni partecipante abbia un indice progressivo.
2. La comunicazione sia soltanto con il precedente ed il successivo.
3. Il nodo finale retropropaga un messaggio "Ok" o "No" a seconda del fatto se i nodi della catena approvino o meno una certa azione.

Nella nostra implementazione si prevede la presenza di un *coordinatore* e un nodo *tail* che avvia la fase di "back-propagation" ovvero la retropropagazione dei messaggi.

Il nodo coordinatore di default ha indice 0, il nodo tail ha come indice  $(n - 1) \bmod n$  (con  $n$  numero di robots).

L'algoritmo linear 2pc come **soluzione** al problema dell'*agreement* nel sistema *Robots clusters* procede come segue:

1. Viene scelta (a priori) un'azione *action\_id*.
2. Il *coordinatore* prende in ingresso l'azione *action\_id* da eseguire nella pool dall'operatore.
3. Viene propagata la richiesta di eseguire l'azione *action\_id* in modo lineare da tutti i robot, il nodo di indice  $i$  legge la richiesta e:
  - (a) Se decide di eseguirla aumenta un contatore che viene passato tra i vari robot quindi  $m = m + 1$  (il set di robot attuatori ha acquisito un nuovo elemento).
  - (b) Se **non** si esegue l'azione si propaga comunque le informazioni su di essa, semplicemente non si attua l'azione e non viene incrementato  $m$ .
4. Il messaggio arriva quindi al nodo *Tail*, il quale verifica se  $m_{found} = m$  cioè se si sono trovati il numero di Robot richiesto per eseguire l'azione *action\_id*.
5. Il nodo tail genera quindi un messaggio: *ACK* se l'azione può essere eseguita dalla pool, *NACK* se **non** può essere eseguita.
6. In ogni caso il messaggio viene retropropagato al nodo controller il quale viene notificato dello stato di tutti gli altri, e può quindi intraprendere decisioni successive (es. interrompere l'azione attuale, proporre un'altra azione).

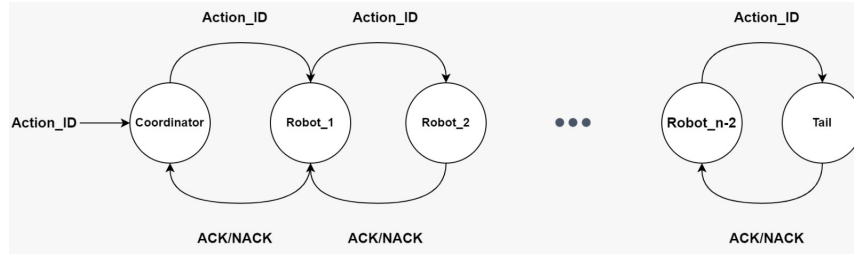


Figura 1: Algoritmo Linear 2PC applicato a *Robots Cluster*

### 3 Implementazione

Si è scelto di implementare il progetto con MatLab 2021b con i pacchetti Simulink, Stateflow e Design Verifier.

Ogni nodo viene descritto attraverso un costrutto *Chart*, che simula il comportamento di una *EFS* (Extended Finite States Machine).

Il flusso di Stateflow garantisce la riproducibilità dei risultati, permettendo di verificare le proprietà (di *safety* o *liveness*) attraverso il tool Design Verifier.

Il modello prevede i seguenti elementi:

1. Un subsystem che contiene  $n$  chart, uno per il coordinatore, altri  $n - 2$  robot e uno per il nodo Tail.
2. Sistemi di verifica automatica (Design Verifier).
3. Due input previsti dall'utente, il valore di robot da reclutare (cioè  $m$ , ed un  $id$  di azione da compiere.

Per garantire la propagazione (e soprattutto la retropropagazione) dei messaggi si era provata la strada di utilizzare il costrutto *messages*, tuttavia questo non è pienamente compatibile con le funzionalità del Design Verifier, quindi si è optato per l'utilizzo del metodo dei *delays*.

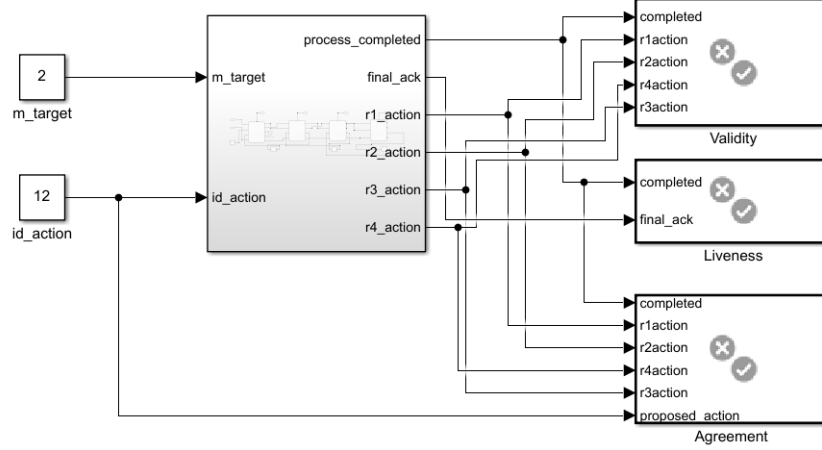


Figura 2: Sistema Input/Output e sottosistema di verifica con le tre proprietà implementato in Simulink.

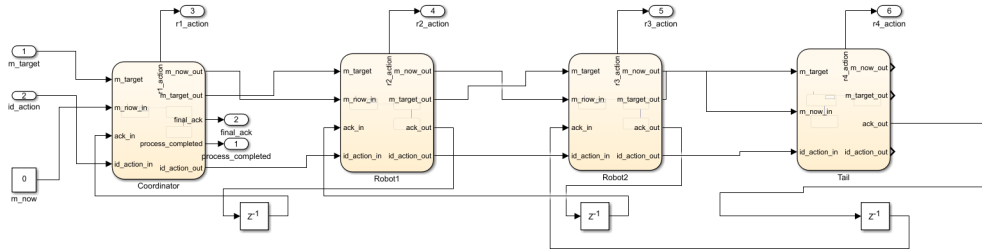


Figura 3: Robot (tra cui coordinator e Tail) implementati come charts (macchine a stati finiti estese) in grado di comunicare tra loro.

Nelle sezioni successive si analizzeranno le ESM dei vari componenti.

### 3.1 Macchina a stati finiti "Coordinator"

Un *Coordinator* è una macchina a stati finiti che ha i seguenti valori in ingresso ed in uscita:

- **Ingresso:**

1. *m\_target*: Variabile fornita dall'operatore che indica il numero di robot da reclutare.
2. *m\_now\_in*: Numero di robot reclutati al passo *i*, il coordinator ha questa variabile *costante* 0.

3. *id\_action\_in*: Id dell'azione da eseguire, in ingresso dall'operatore.
4. *ack\_in*: ACK/NACK ricevuto dal robot di indice 1, questo input porta il coordinatore nel suo stato finale, corrispondente alla fine di tutta la *run*.

• **Uscita:**

1. *m\_now\_out*: Variabile *m\_now* che viene propagata al Robot successivo.
2. *m\_target\_out*: Variabile *m\_target* che viene propagata al Robot successivo..
3. *r<sub>i</sub>\_action*: Variabile di output che indica l'azione intrapresa dal nodo *i*, si usa per verifica formale.
4. *final\_ack*: Ack finale emesso dal Coordinator, anche questa è una variabile di output usata per la verifica formale.
5. *process\_completed*: Variabile emessa dal Coordinator, utilizzata per verifica formale, indica il completamento di tutto il processo.
6. *id\_action\_out*: Id dell'azione che viene propagato al robot successivo.

Ed il seguente **Comportamento**:

Il *Coordinator* prende in Ingresso le variabili del problema dall'operatore, essendo comunque un Robot decide se fare l'azione o meno, procede quindi al Forward dei messaggi.

Lo stato *EnoughRobots* viene attivato alla ricezione di un messaggi di ACK/NACK ( ovvero al termine della retropropagazione).

$$ACK = \begin{cases} 1 & \text{(ACK) : Esito positivo: } m \text{ è sufficiente, l'azione viene eseguita.} \\ -1 & \text{(NACK) : Esito negativo: } m \text{ non è sufficiente.} \end{cases} \quad (1)$$



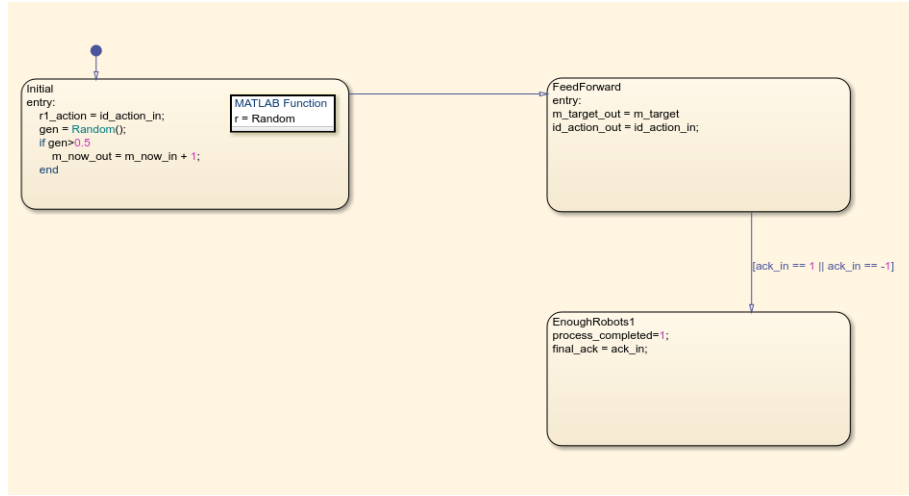


Figura 4: Macchina a Stati Finiti associata al Coordinatore

### 3.2 Macchina a stati finiti "Robot"

Un *Robot* è una macchina a stati finiti che ha i seguenti valori in ingresso ed in uscita:

- **Ingresso:**

1. *m\_target*: Variabile che indica il numero di robot da reclutare propagata dal Robot precedente  $e(i - 1)$ .
2. *m\_now\_in*: Numero di robot reclutati al passo  $i - 1$ .
3. *id\_action\_in*: Id dell'azione da eseguire, propagata dal robot precedente.
4. *ack\_in*: ACK/NACK ricevuto dal robot di indice  $i - 1$ .

- **Uscita:**

1. *m\_now\_out*: Variabile *m\_now* che viene propagata dal Robot al Robot  $i + 1$ .
2. *m\_target\_out*: Variabile *m\_target* che viene propagata dal Robot al Robot  $i + 1$ .
3. *r<sub>i</sub>\_action*: Variabile di output che indica l'azione intrapresa dal Robot  $i$ , si usa per verifica formale.
4. *ack\_out*: Ack finale emesso dal nodo Tail, che viene propagato indietro, anche questa è una variabile di output.
5. *id\_action\_out*: Id dell'azione che viene propagato al robot successivo.

Ed il seguente **Comportamento**:

Il *Robot* prende in Ingresso le variabili dal Robot precedente e decide se fare l'azione o meno.

Nel primo caso incrementa  $m\_now$ , altrimenti lo lascia invariato e procede quindi al Forward dei messaggi.

Lo stato *EnoughRobots* viene attivato alla ricezione di un messaggi di ACK/NACK, retropropagandoli al Robot precedente fino al Coordinatore.

Anche la decisione del Robot è potenzialmente random, anche se il seed che viene utilizzato è deterministico per ragioni di test e verifica formale.

Segue quindi il diagramma del Robot.

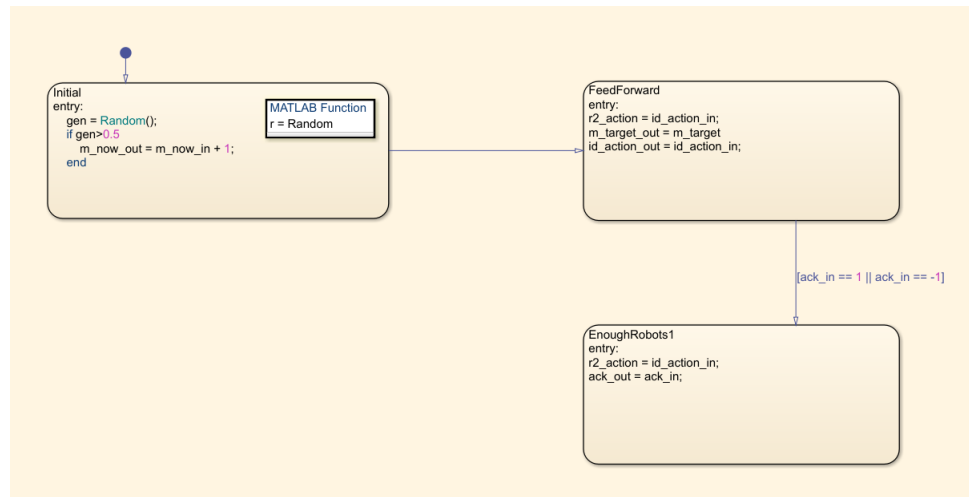


Figura 5: Macchina a Stati Finiti associata al nodo Tail.

### 3.3 Macchina a stati finiti "Tail"

Un sistema *Tail* è una macchina a stati finiti che ha i seguenti valori in ingresso ed in uscita:

- **Ingresso:**

1.  $m\_target$ : Variabile che indica il numero di robot da reclutare propagata dal Robot precedente.
2.  $m\_now\_in$ : Numero di robot reclutati al passo  $i - 1$ .
3.  $id\_action\_in$ : Id dell'azione da eseguire, propagata dal robot precedente.

- **Uscita:**

1.  $ack\_out$ : Ack finale emesso dal nodo Tail, che viene propagato indietro, anche questa è una variabile di output.

Ed il seguente **Comportamento**:

Il nodo *Tail* prende in Ingresso le variabili dal Robot precedente, essendo anch'esso (come il Coordinatore) comunque un Robot, decide se eseguire l'azione o meno.

Nel primo caso incrementa  $m\_now$ , altrimenti lo lascia invariato, anzichè eseguire un forward dei messaggi, si esegue un backward, avviando la retropropagazione.

Lo stato "EnoughRobots" si occupa di stabilire se il processo è stato concluso con successo ( $ACK = 1$ ) oppure con fallimento ( $ACK = -1$ ) cioè *NACK*, avviando la retropropagazione.

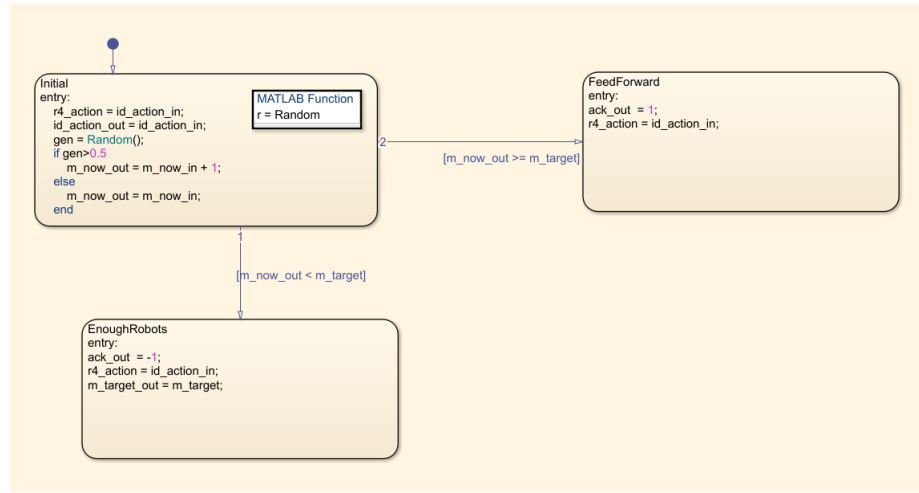


Figura 6: Macchina a Stati Finiti associata al Robot.

## 4 Esempio di simulazione

Il funzionamento (complessivo) di tutto il sistema ponendo ad esempio il caso in cui  $m = 2, n = 4, action\_id = 12$  può essere descritto come segue:

1.  $m, n$  vengono inviati al coordinator il quale crea uno stato composto dalle variabili che verranno propagate, inizializzate come segue:
  - (a)  $m\_now\_out = 0$ .
  - (b)  $m\_target\_out = 2$ .
  - (c)  $r_i\_action = 12$ .
  - (d)  $id\_action\_out = 12$ : Id dell'azione che viene propagato al robot successivo.
2. Le variabili vengono propagate, dato che  $m=2$  si arriva al nodo Tail passando per i robot 2,3, ad esempio il robot 3, supponendo che l'azione sia eseguita dai robot 2,3 avrà il seguente stato:

- (a)  $m\_now\_out = 2$ : Variabile  $m\_now$  che viene propagata dal Robot al Robot  $i + 1$ .
- (b)  $m\_target\_out = 4$ : Variabile  $m\_target$  che viene propagata dal Robot al Robot  $i + 1$ .
- (c)  $r_i\_action = 12$ : Variabile di output che indica l'azione intrapresa dal Robot  $i$ , si usa per verifica formale.
- (d)  $ack\_out = ?$ : Verrà retropropagato dopo la decisione del nodo Tail.
- (e)  $id\_action\_out$ : Id dell'azione che viene propagato al robot successivo.

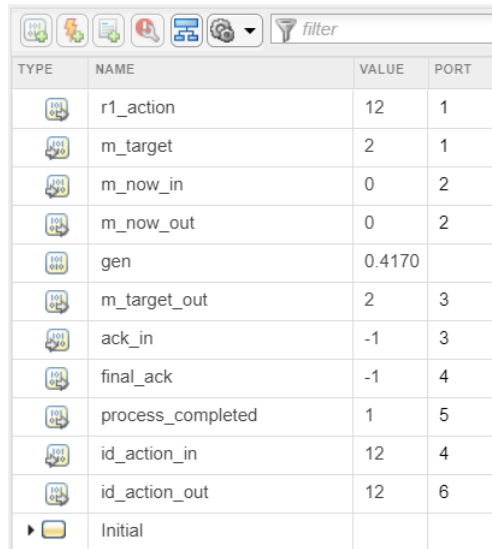
Si passa quindi al nodo Tail con il seguente stato in uscita:

- (a)  $ack\_out = 1$  (perchè  $m < n$  quindi si può fare l'azione).

Il valore verrà poi propagato al Coordinatore, il quale avrà il seguente stato in uscita (finale):

- (a)  $final\_ack = 1$ .
- (b)  $process\_completed = 1$ .

Sancisce cioè la fine del processo con esito positivo, i Robot 2,3 eseguiranno l'azione con id 12.



TYPE	NAME	VALUE	PORT
	r1_action	12	1
	m_target	2	1
	m_now_in	0	2
	m_now_out	0	2
	gen	0.4170	
	m_target_out	2	3
	ack_in	-1	3
	final_ack	-1	4
	process_completed	1	5
	id_action_in	12	4
	id_action_out	12	6
	Initial		

Figura 7: Visualizzazione delle variabili alla fine della simulazione, in questo caso si è supposto che  $m > n$ , quindi  $ACK = -1$  perchè di fatto non si può completare l'azione.

## 5 Verifica Formale

Il sistema Robots Clusters presenta la necessità di determinare se sono vere alcune proprietà che riguardano il suo funzionamento.

Per questo motivo si è eseguita un *analisi di verifica formale* attraverso *Design Verifier* (un componente di Matlab adibito alla verifica formale) delle seguenti proprietà:

1. **Validity** : Se una pool di robot raggiunge l'accordo, allora questo interessa tutti i robot della pool (ognuno di essi è effettivamente dentro la pool).
2. **Liveness** : Se un coordinatore riceve una richiesta di movimento, *prima o poi* lo stesso coordinatore riceverà un ack o un nack.
3. **Agreement** : Non è possibile separare le azioni in una pool, quando viene raggiunto l'accordo su una certa azione da eseguire  $d$ , ogni robot è concorde sul fatto che sia  $d$  l'operazione (eventualmente) da eseguire.

La verifica formale eseguita attraverso Design Verifier prevede l'implementazione di "*circuiti logici di check*" delle proprietà da verificare, il nome di questi elementi è "*Verification Subsystem*" e si comportano a tutti gli effetti come dei sottosistemi, avendo input e output (assertion).

Prima di implementare le proprietà, a titolo di maggior chiarezza, si sono trascritte in logica proposizionale, vengono inoltre introdotte le seguenti variabili "ausiliari" in output dal sistema Cluster Robots per portare a termine la fase di verifica:

1. **Coordinatore:**
  - (a) *final\_ack* : è il valore finale dell'ack che riceve il coordinatore.
  - (b) *process\_completed*: è la variabile che ha valore 1 quando il processo è concluso, ovvero l'*agreement run* di linear pc è conclusa.
2. **Robot:**
  - (a)  $R_i\_action\_id$  : è il valore dell'azione del robot  $i$ -esimo, avere questa variabile in output permette di verificare le proprietà di *agreement* e di *validity*.

Le proprietà vengono formalizzate come segue:

1. **Validity** : Considerando ogni coppia di robot precedente-successivo  $R_i, R_{i+1}$  la proprietà di agreement si formalizza come segue:

$$(process\_completed == 1) \longrightarrow \left( \bigwedge_{i=1}^{n-1} R_i\_action\_id == R_{i+1\_action\_id} \right)$$

2. **Liveness** : Si formalizza come segue:

$$(process\_completed == 1) \longrightarrow \{(final\_ack == 1) \vee (final\_ack == -1)\}$$

3. **Agreement** : Considerando *action\_id* come l'id dell'azione scelto dall'operatore e *R<sub>i</sub>\_action\_id*, l'azione acquisita dal robot *R<sub>i</sub>* al termine della run, la proprietà di agreement si formalizza come segue:

$$(process\_completed == 1) \longrightarrow \left( \bigwedge_{i=1}^n R_i\_action\_id == action\_id \right)$$

Rimane fondamentale il ruolo della variabile *process\_completed* in quanto permette di verificare facilmente tutte le proprietà come semplici *implicazioni* del tipo "se l'agreement run è conclusa, allora si verifica la validità della proprietà *P<sub>i</sub>*".

Viene quindi presentata l'implementazione delle proprietà come circuiti logici nel dettaglio.

### 5.1 Proprietà di Agreement

La proprietà di Agreement, nel caso in cui si abbia  $n = 4$  viene implementata con il seguente circuito:

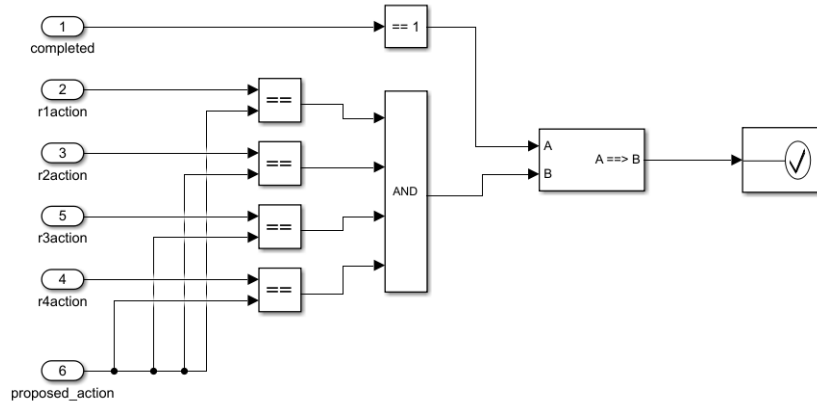


Figura 8: Circuito di implementazione della proprietà di Agreement.

### 5.2 Proprietà di Liveness

La proprietà di Liveness, sempre nel caso in cui  $n = 4$ .

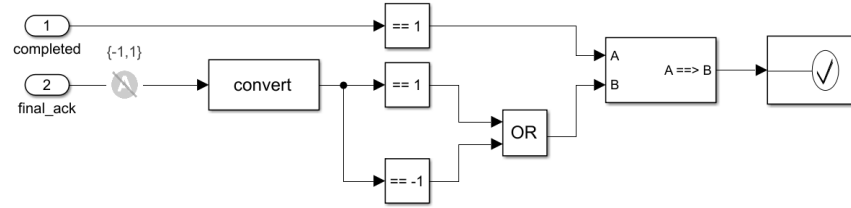


Figura 9: Circuito di implementazione della proprietà di Liveness.

### 5.3 Proprietà di Validity

La proprietà di Validity invece, viene implementata come segue:

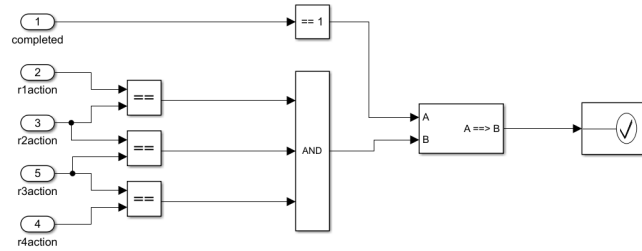


Figura 10: Circuito di implementazione della proprietà di Validity.

### 5.4 Funzionamento della verifica

Infine si mostra il sistema completo, sempre nel caso in cui  $n = 4, m = 2$  in cui si evidenziano anche i blocchi di verifica:

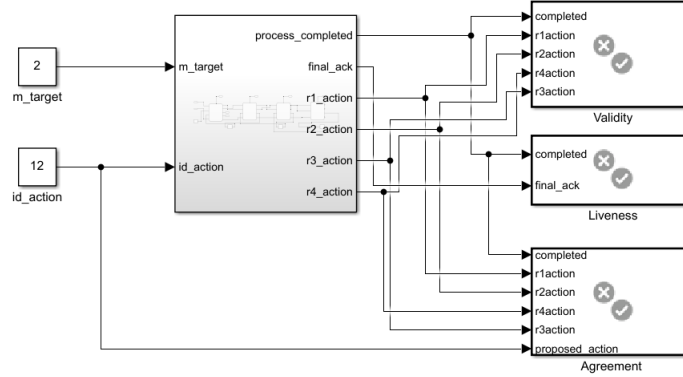


Figura 11: Sistema con tre verification subsystems, ovvero i sottosistemi di verifica delle proprietà sopracitati

A titolo di esempio si riporta un esecuzione della verifica delle proprietà, sempre nel caso sopracitato:

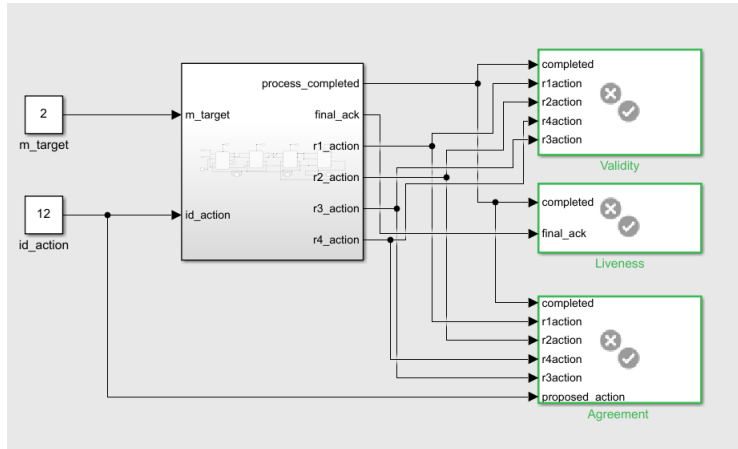


Figura 12: Le proprietà risultano verificate, i robot con indice 2,3 eseguono l'azione richiesta,  $m = 2$ , pertanto il sistema conclude l'esecuzione con un acknowledgement positivo  $ACK = 1$ .

In seguito si riporta l'esito del report generato da Design Verifier:



#	Type	Model Item	Description	Analysis Time (sec)
1	Assert	Agreement /Assertion	Assert	2
2	Assert	Liveness/Assertion	Assert	2
3	Assert	Validity/Assertion	Assert	2

Figura 13: Report di Matlab Design Verifier

#### 5.4.1 Esempio di richiesta errata

Si esegue infine un ulteriore test in cui dato  $n = 4$ , quindi 4 robot disponibili a svolgere una certa azione (con  $action\_id = 12$ ), si cerca di reclutare 7 robots, effettuando una richiesta che **non** potrà essere portata a termine.

Ci si aspetta che la richiesta venga propagata fino al nodo Tail, il quale concludendo che  $n > m$  l'esito sia  $ACK = -1$ , ovvero un nack, non eseguendo quindi l'azione ma non violando le 3 proprietà richieste, nello specifico quella di Liveness.

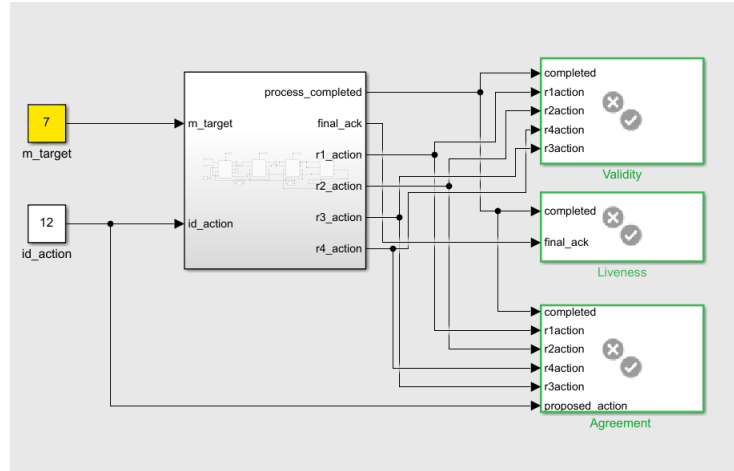


Figura 14: Configurazione del sistema nel caso sopracitato.

Come si può osservare nell'immagine precedente, le proprietà vengono comunque rispettate.

## 6 Scalabilità

Il sistema presentato nelle sezioni precedenti è stato modellato con un numero di Robot pari a 4, in cui è possibile distinguere 3 componenti principali: Coordinator, Robot (nell'esempio in figura 3 sono presenti 2 Robot) e un Tail.

Nel caso si volesse scalare il modello in modo da includere ulteriori Robot sarà necessario:

- Aggiungere ulteriori Robot in una qualunque posizione ad eccezione del nodo Coordinator e del nodo Tail, in quanto, per come è stato progettato, il modello questi due nodi hanno dei comportamenti e la posizione predefinita all'interno del modello.
- Effettuare i collegamenti input/output tra i nuovi Robot aggiunti e i Robot già presenti nel modello, in modo da non interrompere la catena di propagazione/retropropagazione dei messaggi.

Mentre per quanto riguarda la parte di verifica formale sarà necessario apportare le seguenti modifiche:

- Nella verifica dell'Agreement sono stati aggiunti i controlli sulla variabile  $r_i\_action$ , in modo da controllare se effettivamente tutti i Robot nella Pool raggiungono un accordo, anche con l'aggiunta di nuovi Robot.
- In modo analogo, per verificare la Validity del sistema, sono stati aggiunti i controlli sulla variabile  $r_i\_action$ . In modo da controllare se al momento che viene raggiunto l'accordo su una certa azione da eseguire  $d$ , ogni robot è concorde sul fatto che sia  $d$  l'operazione (eventualmente) da eseguire.

## 6.1 Esempi di scalabilità

Si riportano in questa sezione due esempi di scalabilità del modello e la relativa verifica formale.

Nel primo caso è stata creata una simulazione in cui operano 5 Robot a quali viene sottoposto un lavoro per cui sono richiesti 3 Robot. Nel secondo caso è stata creata una simulazione in cui operano 7 Robot a quali viene sottoposto un lavoro per cui sono richiesti 3 Robot.

Per entrambe le realizzazioni sono riportati gli screenshot degli StateFlow e una visione dall'alto del sistema in cui è possibile vedere Input/Output e sottosistema di verifica con le tre proprietà

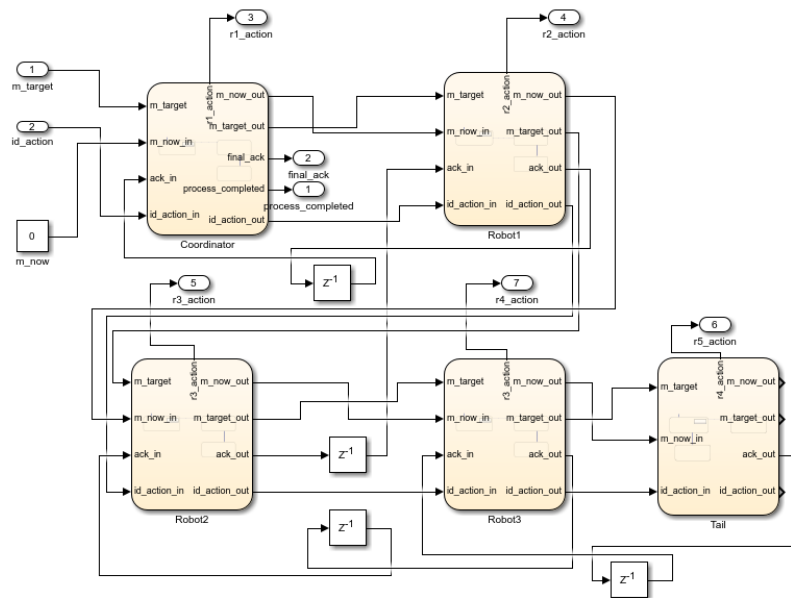
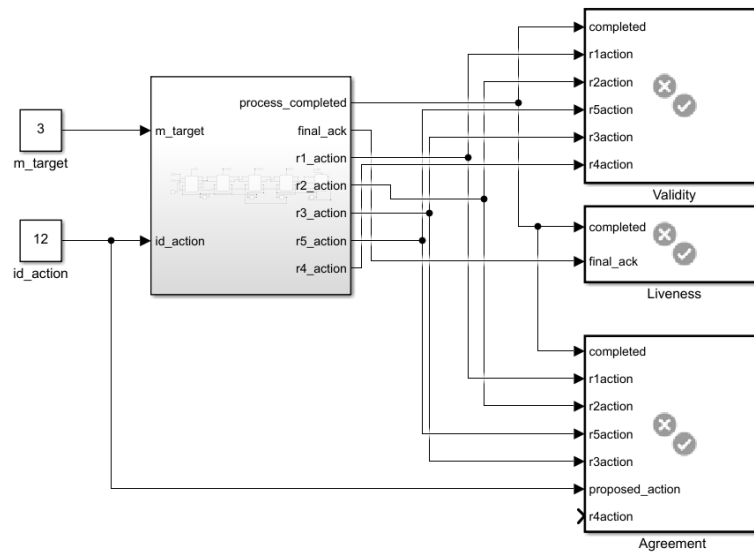


Figura 15: Stateflow nel caso di 5 Robot

Figura 16: Modello per la verifica formale in presenza di 5 Robot e un numero di Robot necessari per portare a termine il task richiesto ( $m$ ) pari a 3

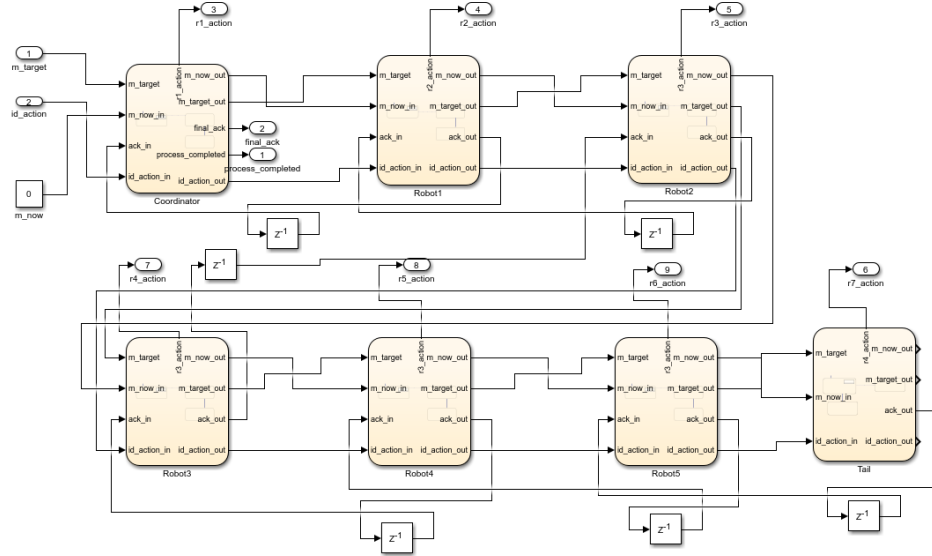
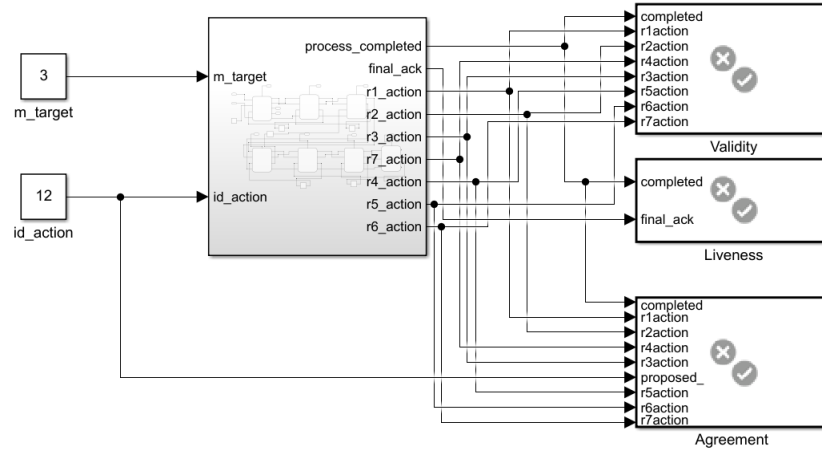


Figura 17: Stafeflow nel caso di 7 Robot

Figura 18: Modello per la verifica formale in presenza di 7 Robot e un numero di Robot necessari per portare a termine il task richiesto ( $m$ ) pari a 3

Si riportano i valori ottenuti alla conclusione della verifica formale nel caso dei due modelli precedenti.

Model Item	Property	Status	Analisis Time (Sec)
Agreement	Assert	Valid	2
Liveness	Assert	Valid	2
Validity	Assert	Valid	2

Tabella 1: Valori estrapolati dal report di verifica formale per il modello con  $n = 5$  e  $m = 3$

Model Item	Property	Status	Analisis Time (Sec)
Agreement	Assert	Valid	2
Liveness	Assert	Valid	2
Validity	Assert	Valid	2

Tabella 2: Valori estrapolati dal report di verifica formale per il modello con  $n = 7$  e  $m = 3$

Alla luce delle modifiche effettuate al modello di base ed ai risultati ottenuti nelle simulazioni e della verifica formale possiamo concludere che il modello continua a mantenere il comportamento desiderato. Inoltre, poiché il modello mantiene le tre proprietà di Validity, Liveness e Agreement, possiamo concludere che esso sia altamente scalabile con elevata semplicità.

## 7 Coordinatore Multiplo

Si vuole adattare il nostro modello per poter gestire il problema dei coordinatori multipli, ovvero la presenza di più di un coordinatore.

L'idea è quella di avere una prima agreement run Linear-2PC tra tutti gli agenti per individuare chi avrà il ruolo di *coordinatore*, secondo una certa *policy*.

La seconda fase prevede si assegnare i robots ai coordinatori, che assumono il ruolo di *cluster head*.

Un'idea potrebbe essere assegnare un robot ad un certo coordinatore basandosi su una metrica di *distanza*, mettendo in atto un comportamento di assegnazione simile all'utilizzo di un algoritmo di Clustering (es. K-Means).

Un agente esterno assegna un set di azioni  $S$  con  $|S| = |Coordinators|$  al sistema (coordinatori + robots), ogni coordinatore assumerà il comando di una certa azione, l'agreement di una sotto-pool di robot associati ad un coordinatore si raggiunge con una seconda agreement run di linear-2PC, relativa alla singola "catena" coordinatore + robots.

Si propone un esempio che vede l'utilizzo di due Coordinatori e due rispettivi cluster di robots.

Si suppone che:

1. Durante la fase di inizializzazione vengano fornite al sistema due azioni da eseguire.
2. Una agreement run linear-2PC viene eseguita per individuare due coordinatori tra tutti i robots, si assegnano le azioni da eseguire ai due coordinatori.
3. Un criterio (potrebbe essere il suddetto algoritmo di clustering) associa ogni nodo non-coordinatore ad un coordinatore.

In questa situazione si hanno quindi due *catene* di agenti composte da un coordinatore e più robots.

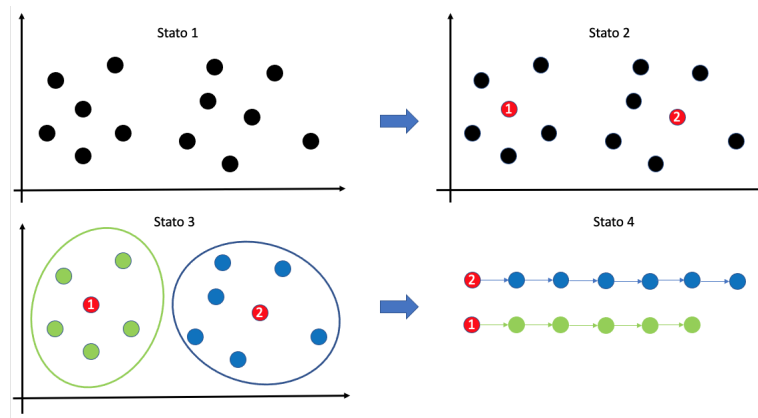


Figura 19: Individuazione Coordinatore e K-Means

Fornendo una descrizione più dettagliata:

1. Nello Stato 1 tutti i Robot sono in attesa di ricevere un comando.
2. La ricezione di due azioni fa partire una run di Agreement 2PC che terminerà con l'individuazione di due Coordinatori, nello Stato 2 i Coordinatori sono evidenziati dal Colore Rosso.
3. Per passare allo Stato 3 un Algoritmo di Clustering K-Means utilizzando i coordinatori scelti come centroidi, suddivide i Robot in due gruppi.
4. Si crea una catena per ogni cluster con il coordinatore e ciascun robot (Stato 4).

5. Il sistema procede in tutti i sotto-sistemi come nel caso singolo sistema a singolo controllore analizzato precedentemente.
6. I vari sottosistemi eseguiranno o meno le varie azioni, nel caso in cui non si incontreranno problemi tutte le azioni verranno eseguite.

Nel caso in cui le azioni non verranno eseguite, alcuni coordinatori riporteranno lo stato finale ABORT.

Si è quindi simulato il sistema, sotto l'assunzione che le catene di agenti (robots + coordinatore) siano già state create come descritto precedentemente, si è modificata la simulazione in modo da gestire il coordinamento tra 2 cluster che eseguono azioni diverse.

I passaggi successivi riprendono i sottosistemi StateFlow sviluppati nei capitoli precedenti, il sistema di Verifica Formale viene adattato al contesto.

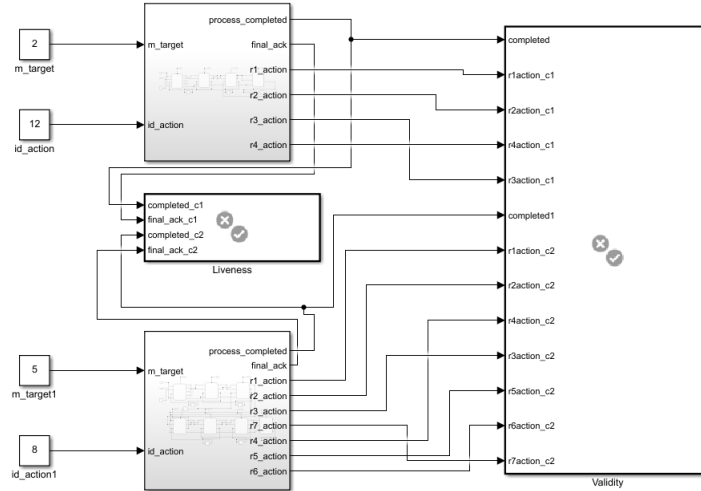


Figura 20: Verifica Formale con Coordinatori Multipli, si possono notare i due sottosistemi relativi ai due cluster (uno con 3 robots ed un coordinatore, l'altro con 7 robots ed un coordinatore), che prendono in ingresso azioni diverse.

I due sottosistemi che corrispondono ai Cluster appena formati, dovranno rispettare le proprietà di *Validity* e *Liveness*, il rispetto della priorità di *Agreement* non è verificabile, data la presenza di più azioni.

Segue quindi la descrizione dell'implementazione delle proprietà da rispettare negli specifici sottosistemi di verifica (uno per *Validity*, uno per *Liveness*).

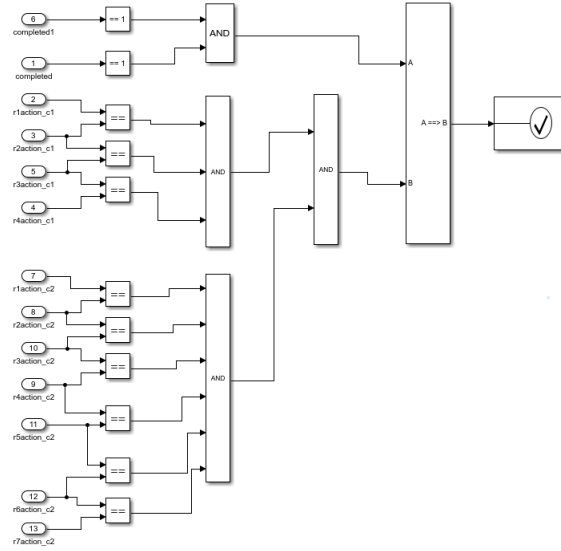


Figura 21: Implementazione del sottosistema di verifica per proprietà *validity* in sistema con coordinatori multipli

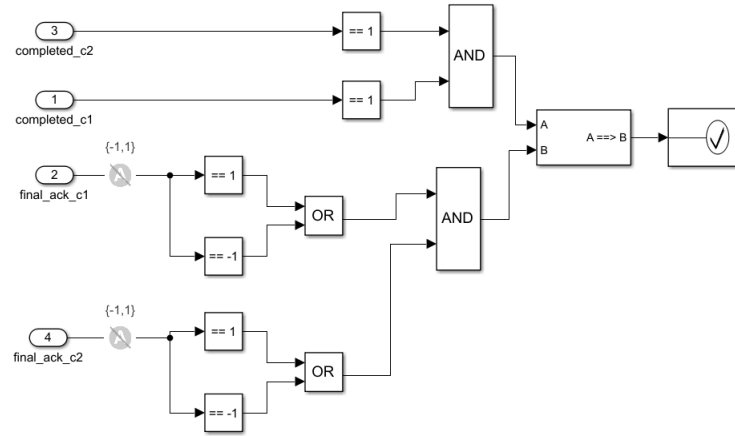


Figura 22: Implementazione del sottosistema di verifica per proprietà *liveness* in sistema con coordinatori multipli

Si riportano i valori ottenuti alla conclusione della verifica formale.



Model Item	Property	Status	Analisis Time (Sec)
Liveness	Assert	Valid	2
Validity	Assert	Valid	2

Tabella 3: Valori estrapolati dal report di verifica formale per il modello con Coordinatori Multipli

Si è verificato quindi che, sotto le precedenti assunzioni, il sistema *Robots Clusters* con *coordinatori multipli* rispetta le proprietà richieste.

## 8 Failure

In questo task si suppone che nella catena di agenti possano avvenire dei *failures*, dalla letteratura si possono distinguere le tipologie di *failure* secondo il dominio del fallimento:

- **Fallimenti nel valore:** Il valore del servizio fornito non è conforme alla specifica;
- **Fallimenti nel tempo:** La temporizzazione della fornitura del servizio non è conforme alla specifica.

Il modello base non era in grado di gestire situazioni di failure, è stata quindi apportata una modifica in modo che riuscisse a gestire questo problema.

Prima di vedere nel dettaglio la modifica effettuata è necessario discutere sulle tipologie di failure che si possono verificare in questo sistema:

- **Guasto di un Robot all'interno della comunicazione:** Ogni singolo Robot per un guasto sottosistema di comunicazione, per un guasto dell'interno del proprio sistema o per la scarica completa della propria batteria potrebbe *interrompere* la propagazione o la retropropagazione dei messaggi, mettendo in stallo tutto il sistema.
- **Ritardo nella propagazione o retropropagazione:** Ogni singolo Robot per simili cause interne potrebbe generare dei *ritardi* nella propagazione o retropropagazione dei messaggi all'interno del sistema.  
Questo porterebbe il sistema a dover attendere una risposta e non poter raggiungere un accordo/disaccordo e non poter eseguire/non eseguire il lavoro richiesto in tempi brevi.
- **Valorizzazione errata degli ACK:** Al controllore potrebbe arrivare un valore di acknowledgement finale fuori dal dominio  $ACK = [-1, 0, 1]$ .

Vediamo adesso la modifica effettuata al Robot Coordinator: Confrontando la macchina a stati finiti del Coordinator iniziale e quello modificato per riuscire a gestire le Failure è possibile vedere un nuovo stato Failure.

Il Coordinator entrerà in questo stato se e solo se passati 30 secondi il Robot successivo non avrà inviato  $ACK = 1$  o  $ACK = -1$ .

Grazie a questa modifica il Coordinator imposterà  $final\_ack = 0$  (Situazione di guasto) e  $process\_completed = 0$  (il lavoro richiesto non verrà portato a termine) dopo 30 secondi dall'avvio della procedura di accordo.

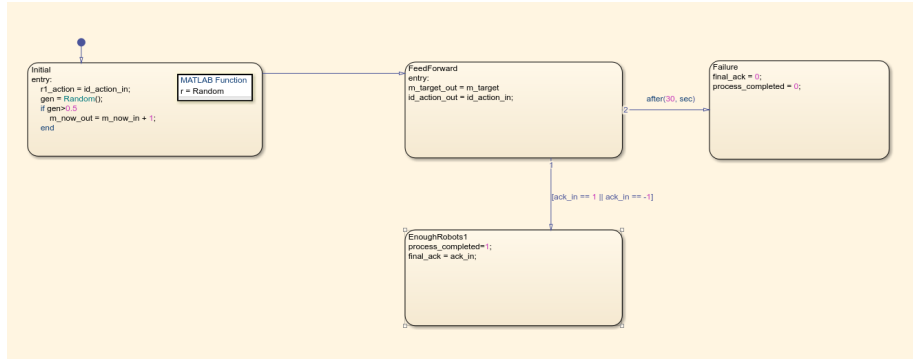


Figura 23: Modifica apportata al Robot Coordinatore in modo che riuscisse a gestire i due casi di Failure descritti precedentemente.

Per verificare che, anche nel caso di presenza di Failure, il modello garantisca il regolare funzionamento e rispetti le tre proprietà di Validity, Liveness e Agreement, si è eseguita una simulazione.

Per simulare un possibile guasto all'interno del sistema sono stati scollegati tutti i collegamenti tra in Robot  $R_2$  e il Robot Tail.

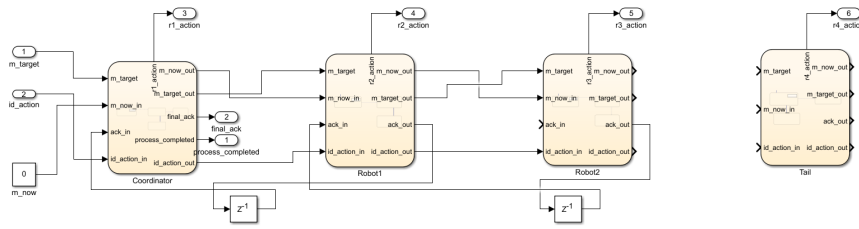
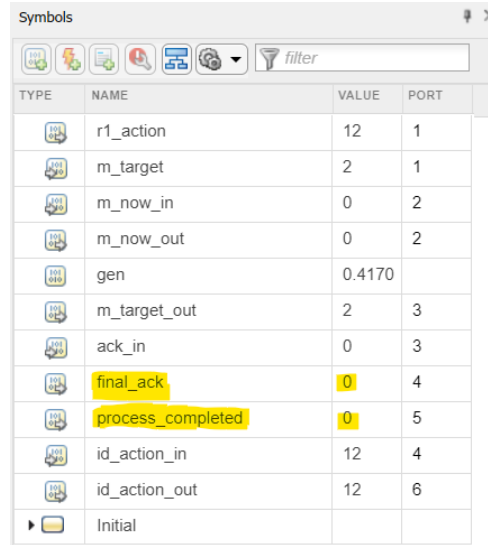


Figura 24: Modello modificato in modo da generare una situazione di Failure, rimuovendo tutti i collegamenti tra il Robot  $R_2$  e il Robot Tail, simulando così un guasto.

Conclusa la simulazione, si può notare in figura 14 come le variabile  $final\_ack$  e  $process\_completed$  vengono valorizzate entrambe a 0.

Il valore di queste variabili indicano che il compito richiesto **non** verrà portato a termine poiché il processo di accordo è stato terminato dal coordinator per un guasto o un blocco dello scambio dei messaggi tra i Robot.

Avviando il tool di verifica formale è possibile accertarsi che il modello continua a rispettare le proprietà di Validity, Liveness e Agreement anche in presenza di un fallimento.



TYPE	NAME	VALUE	PORT
	r1_action	12	1
	m_target	2	1
	m_now_in	0	2
	m_now_out	0	2
	gen	0.4170	
	m_target_out	2	3
	ack_in	0	3
	final_ack	0	4
	process_completed	0	5
	id_action_in	12	4
	id_action_out	12	6
	Initial		

Figura 25: Valorizzazione dei simboli nello statechart.

Model Item	Property	Status	Analisis Time (Sec)
Agreement	Assert	Valid	2
Liveness	Assert	Valid	2
Validity	Assert	Valid	2

Il lavoro realizzato non gestisce il caso in cui il coordinatore riceva un valore di ACK fuori dal dominio  $ACK = [-1, 0, 1]$ .

Al fine di colmare questa carenza, si attua banale modifica, sarà infatti necessario inserire un nuovo stato di errore che verrà raggiunto da una nuova transizione.

La nuova transizione andrà a controllare semplicemente il valore dell'ACK ricevuto, se questo valore sarà fuori dal dominio il Coordinator verrà portato nello stato di errore.

## 8.1 Soluzione alternativa

La soluzione proposta nella sezione precedente è tale che in presenza di un failure il modello termini in uno finale di errore non eseguendo la richiesta in corso.

In particolare, la presenza di un timeout nel nodo coordinatore interrompe tutto il processo di agreement dopo un certo intervallo di tempo di "attesa massima".

Una possibile soluzione alternativa, per la gestione dei fallimenti nel dominio del tempo, consiste nell'interrompere l'agreement run qualora si verifichi un fallimento, ed eseguire comunque l'azione nel caso in cui si abbia un numero di Robot "concordi" pari a  $m$ , sebbene ci sia stato un fallimento.

Il modello potrebbe essere rilassato modificando la condizione di agreement totale, ma garantendo un agreement *parziale* tra i Robot, in modo che il sistema non si fermi se un robot presenta un fallimento nel dominio del tempo (es. non risponde o risponde con un ritardo eccessivo).

Sarà necessario modificare il modello precedente, apportando le seguenti modifiche:

1. Aggiungere un timeout su ogni singolo Robot, invece che solo sul nodo Coordinator.
2. Il timeout partirà dal momento dell'invio del messaggio al Robot successivo.
3. Se il timeout dovesse scadere (mancata risposta del nodo successivo) entra in uno stato di controllo, in cui si va a verificare se si è raggiunto un numero di Robot  $m$  pari a quello richiesto dall'azione da intraprendere.
4. Nel caso si sia raggiunto un numero di Robot  $m$  pari a quello richiesto dall'azione da intraprendere si inizia a retropropagare il messaggio di ACK, in modo da iniziare il task con i soli Robot che hanno raggiunto l'accordo.
5. Nel caso non si sia raggiunto un numero di Robot  $m$  pari a quello richiesto dall'azione da intraprendere verrà retropropagato il messaggio NACK in modo da interrompere l'esecuzione della Run.

Con questa modifica (rilassando però il vincolo di un agreement totale) è possibile gestire i *guasti nel dominio del tempo* dovuti a robot che per un qualche motivo ritardano nella risposta, ma portare comunque a termine l'azione richiesta se il numero di Robot che hanno raggiunto l'accordo sia pari almeno a  $m$ .

Integrando questa potenziale soluzione con la gestione dei failures nel dominio del valore descritta nella sezione precedente, si ha una policy in grado di trattare in modo efficiente (preservando l'esecuzione dell'azione quando necessario) dei fallimenti.

## 9 Conclusioni

In sintesi il lavoro svolto può essere suddiviso nei seguenti punti che ricalcano gli obiettivi dell'elaborato:

- **Obj1 - Modello:** Si è scelto di implementare il progetto con MatLab 2021b con i pacchetti Simulink, Stateflow e Design Verifier. Ogni nodo è stato descritto attraverso un costrutto Chart, che simula il comportamento di una EFS (Extended Finite States Machine). Ogni EFS rappresenta un nodo di un sistema distribuito, la quale comunicazione segue l'algoritmo di *Agreement Two-Phase Linear Protocol*.
- **Obj2 - Verifica Formale:** La verifica formale è stata portata avanti attraverso l'utilizzo di Design Verifier.  
  
Si sono modellate le singole proprietà da verificare, le si sono quindi implementate come circuiti logici nei blocchi "Verification Subsystem" offerti dall'estensione di matlab.  
  
Infine si sono eseguiti test vari in cui si è dimostrata l'efficacia di queste tecniche di property proving.
- **Obj3 - Scalabilità:** Si è discusso della possibilità di espandere il modello aggiungendo Robot alla pool. Dopo aver discusso delle modifiche necessario per far sì che il sistema continuasse ad avere il comportamento richiesto, sono stati eseguiti due test con due modelli espansi per verificare che le tre proprietà di Validity, Liveness e Agreement venissero mantenute.
- **Obj4 - Coordinatori Multipli:** La possibilità di avere controllori multipli viene modellata come due Run di Agreement 2PC. La prima per individuare i Coordinatori tra i Robot Disponibili, la seconda in modo che ogni Coordinatore abbia un subset di Robot, opportunamente clusterizzati, ai quali richiedere una determinata azione.
- **Obj5 - Fallimento:** Dopo aver evidenziato due possibili casi di guasti del modello (Guasto di uno o più Robot, stallo del sistema dovuto a ritardi nelle comunicazioni) è stata proposta, modellata e verificata una possibile soluzione dove la decisione di Abort delle operazioni viene presa dal Coordinator. Si è discusso una seconda soluzione, nella quale i singoli Robot possono comunque procedere ad eseguire il Task se si è raggiunto il numero necessario di Robot anche se è presente uno stato di Failure.

Si riportano le ore impiegate nei vari obiettivi:

Obiettivo	Richiesta	Ore
//	Studio Preliminare	20
1	Modello	16
2	Verifica Formale	17
3	Scalabilità	4
4	Coordinatori Multipli	5
5	Fallimento	6
//	Finalizzazione e scrittura Report	10
//	Totale	78

Il lavoro è stato svolto sempre in gruppo, le ore riportate sono da intendersi come comuni a tutti e 3 gli studenti.

## Riferimenti bibliografici

- [1] *Property Proving with Multiple Properties - MATLAB & Simulink - MathWorks Italia*. URL: <https://it.mathworks.com/help/sldv/ug/property-proving-with-multiple-properties.html>.
- [2] *Property Proving Workflow for Cruise Control - MATLAB & Simulink - MathWorks Italia*. URL: <https://it.mathworks.com/help/sldv/ug/property-proving-workflow-for-cruise-control.html>.
- [3] *Property Proving Workflow for Thrust Reverser - MATLAB & Simulink - MathWorks Italia*. URL: <https://it.mathworks.com/help/sldv/ug/property-proving-workflow-for-thrust-reverser.html>.
- [4] *Prove Properties in a Model - MATLAB & Simulink - MathWorks Italia*. URL: <https://it.mathworks.com/help/sldv/ug/prove-properties-in-a-model.html>.
- [5] *Validate Requirements by Analyzing Model Properties - MATLAB & Simulink - MathWorks Italia*. URL: <https://it.mathworks.com/help/sldv/ug/requirement-set-validation-property-analysis.html>.
- [6] *What Is Property Proving? - MATLAB & Simulink - MathWorks Italia*. URL: <https://it.mathworks.com/help/sldv/ug/what-is-property-proving.html>.