Team Automata@Work: On Safe Communication

Maurice H. ter Beek[1] Rolf Hennicker[2] Jetty Kleijn[3]

[1]ISTI–CNR, Pisa, Italy

[2]Ludwig-Maximilians-Universität München, Germany

[3]LIACS, Leiden University, The Netherlands

COORDINATION 2020
16 June 2020

Setting and topics of this talk

- Systems of communicating components which interact through message exchange
- *Here:* synchronous communication, i.e., outputs and inputs of the same message are performed simultaneously
- Consider a wide range of synchronisation types (peer-to-peer, master-slave, multicast, broadcast, gathering, . . .)

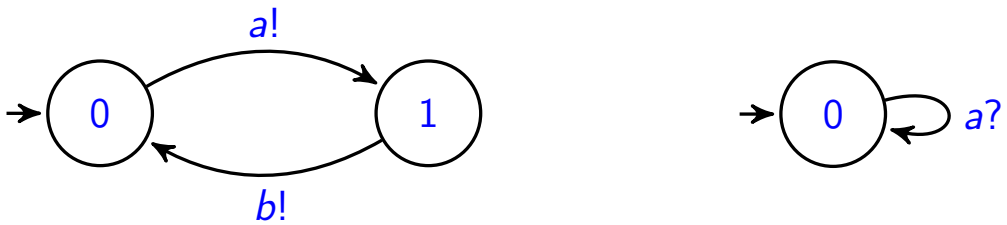
Overall goal:

Safe communication, i.e., avoidance of communication errors

Literature: [ter Beek, Carmona, Hennicker, Kleijn COORDINATION 2017] Communication Requirements for Team Automata

Typical communication errors

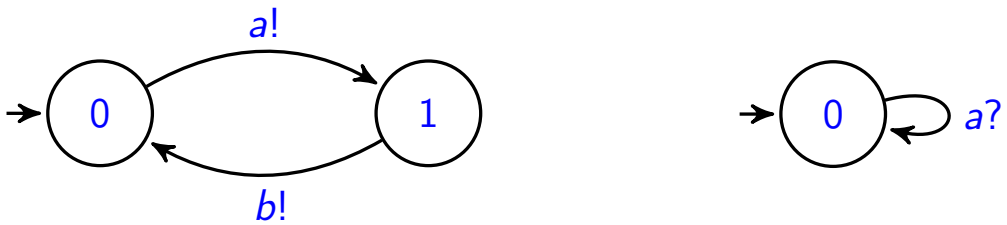
- A component sends a message to another component which is not ready to receive it (*message loss*)



Literature: [Brand, Zafiropulo 1983], [de Alfaro, Henzinger 2001], [Carmona, Cortadella 2002], [Larsen, Nyman, Wąsowski 2007], ...

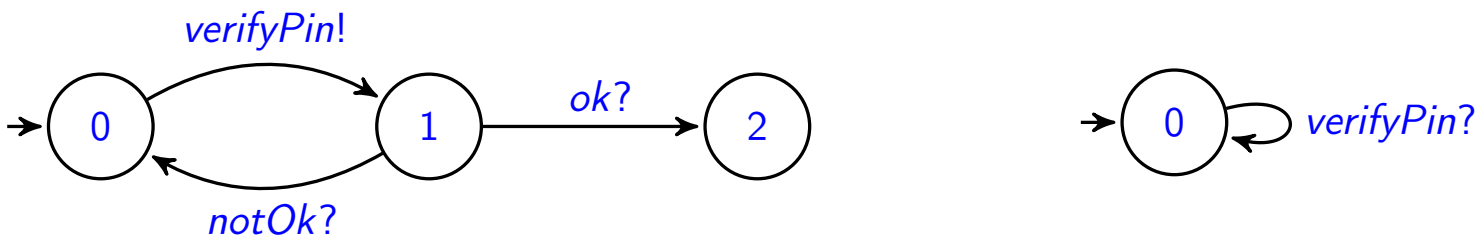
Typical communication errors

- A component sends a message to another component which is not ready to receive it (*message loss*)



Literature: [Brand, Zafiropulo 1983], [de Alfaro, Henzinger 2001], [Carmona, Cortadella 2002], [Larsen, Nyman, Wąsowski 2007], ...

- A component waits for a message from another component which does not send it (*indefinite waiting*)



Literature: [Carrez, Fantechi, Najm 2003], [Durán, Ouederni, Salaün 2012]

Contribution of our approach

Background from COORDINATION'17:

- Uniform formalism for specifying various synchronisation types
- Communication requirements: *receptiveness* and *responsiveness*
- Communication requirements depend on type of synchronisation
  - Generic theory for communication-safety (compatibility):

    synchronisation type A $\mapsto$ compatibility notion A

    synchronisation type B $\mapsto$ compatibility notion B

    ...
- Formalised in team automata:

  synchronisation policies not fixed a priori

Contribution of our approach

Background from COORDINATION'17:

- Uniform formalism for specifying various synchronisation types
- Communication requirements: *receptiveness* and *responsiveness*
- Communication requirements depend on type of synchronisation
  - Generic theory for communication-safety (compatibility):

    synchronisation type A $\mapsto$ compatibility notion A

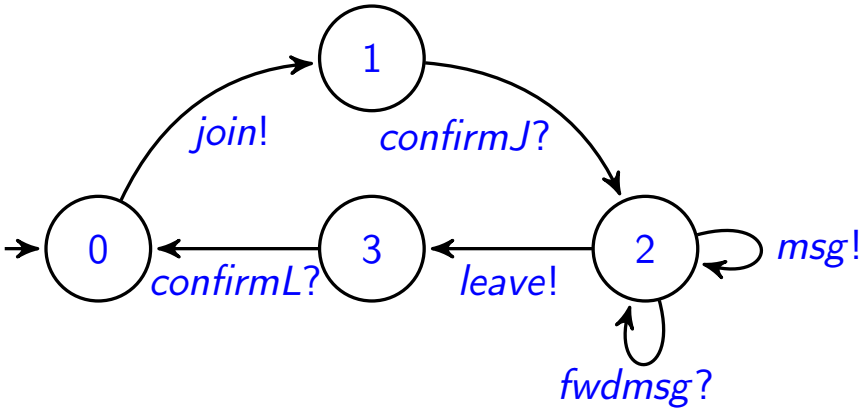
    synchronisation type B $\mapsto$ compatibility notion B

    ...
- Formalised in team automata:

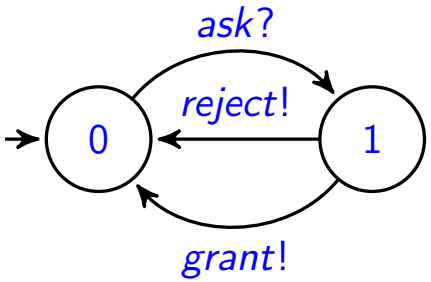
  synchronisation policies not fixed a priori

! In this paper, we propose three extensions

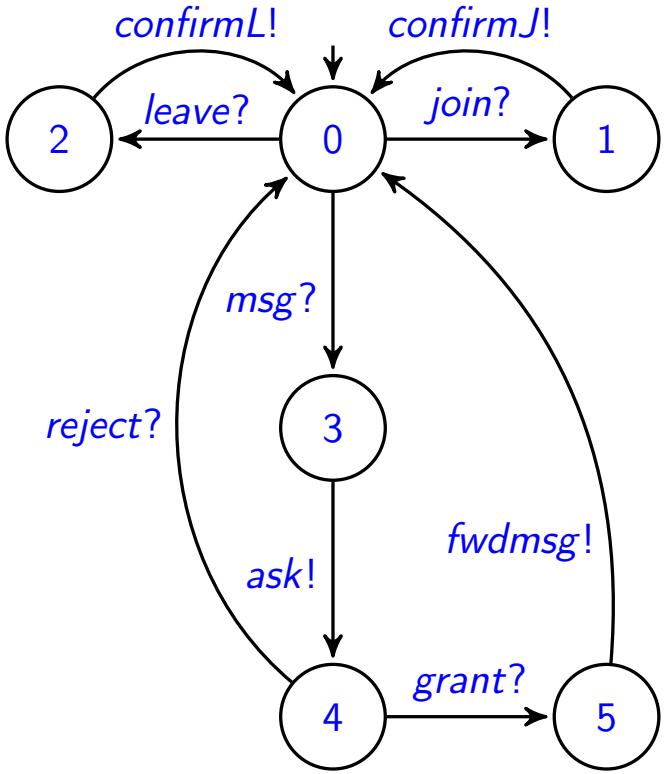
Example: a distributed chat system



(a) *Client*



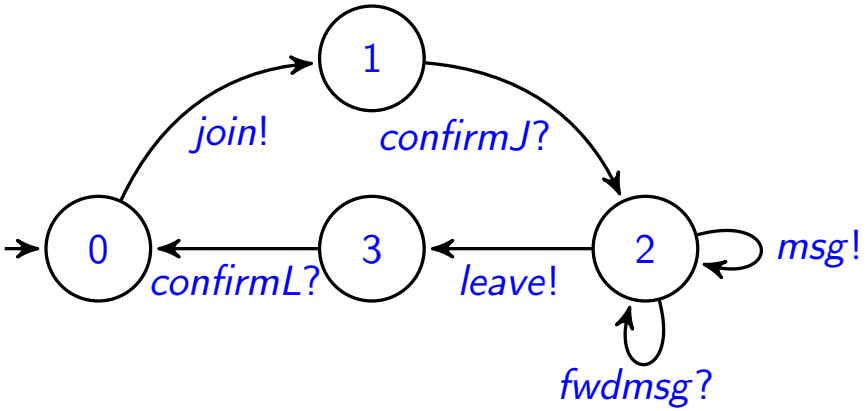
(b) *Arbiter*



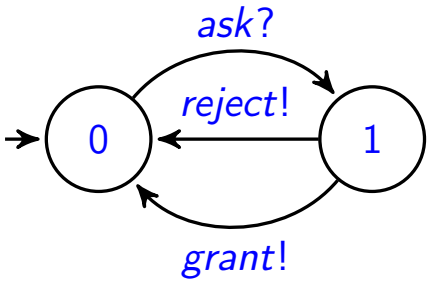
(c) *Server*

Messaging protocol: clients communicate messages (*msg*) to server which, upon arbiter approval, broadcasts the messages (*fwdmsg*) to all clients in the chat

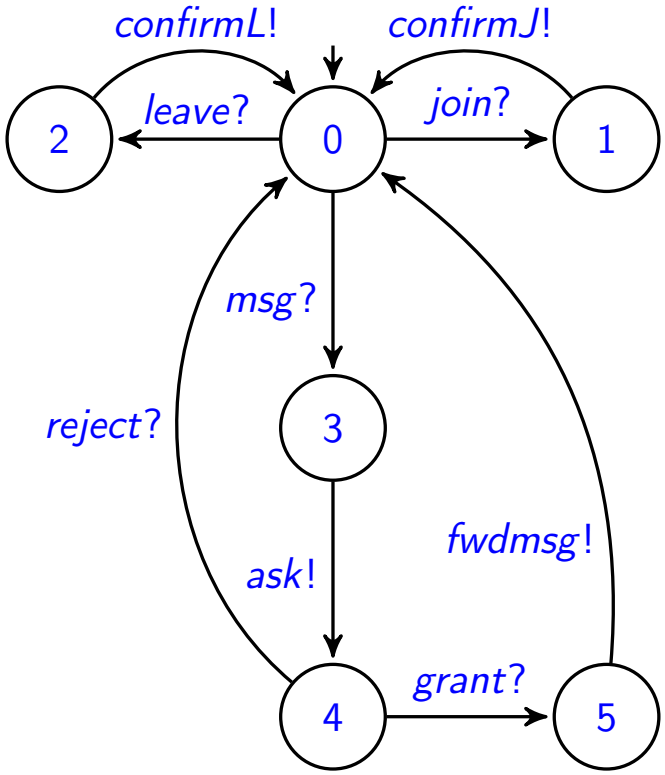
Example: a distributed chat system



(a) *Client*



(b) *Arbiter*

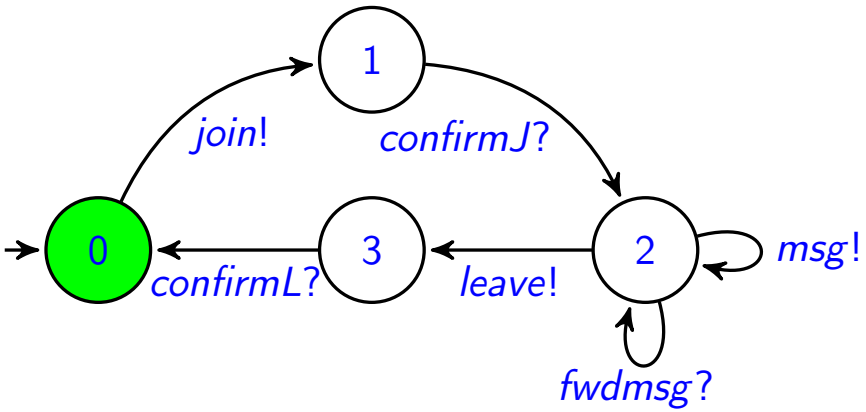


(c) *Server*

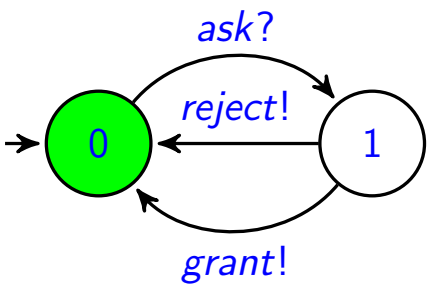
Messaging protocol: clients communicate messages (*msg*) to server which, upon arbiter approval, broadcasts the messages (*fwdmsg*) to all clients in the chat

Here: a system with $\mathcal{C}lient_1$, $\mathcal{C}lient_2$, $\mathcal{S}erver$, $\mathcal{A}rbiter$

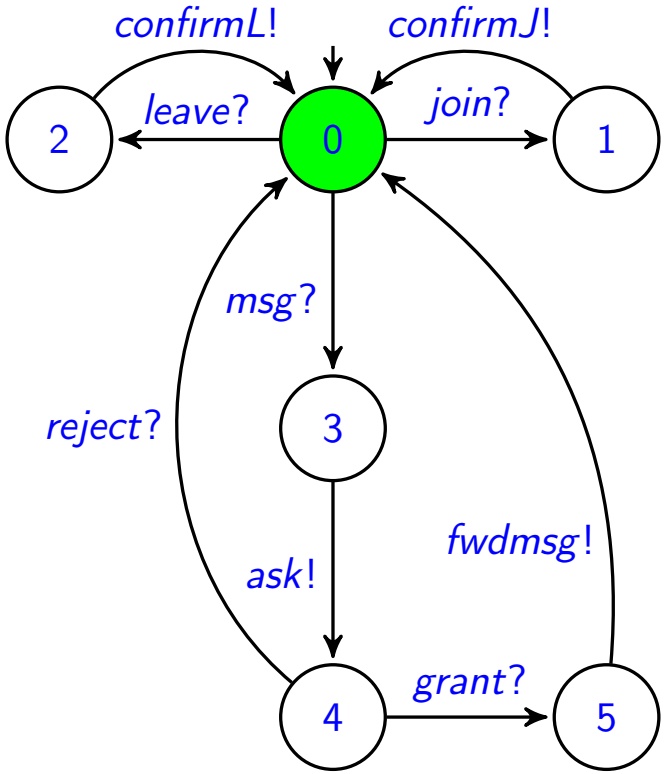
Not all system transitions are meaningful



(a) *Client*



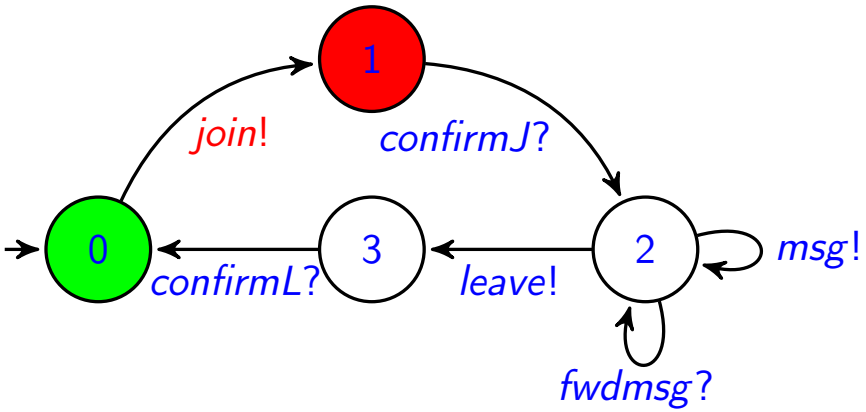
(b) *Arbiter*



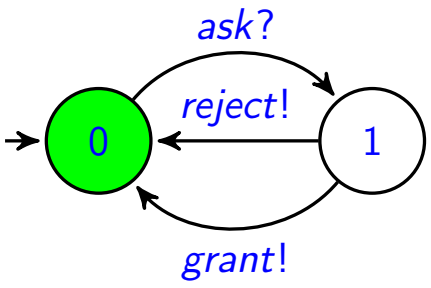
(c) *Server*

$(0, 0, 0, 0)$

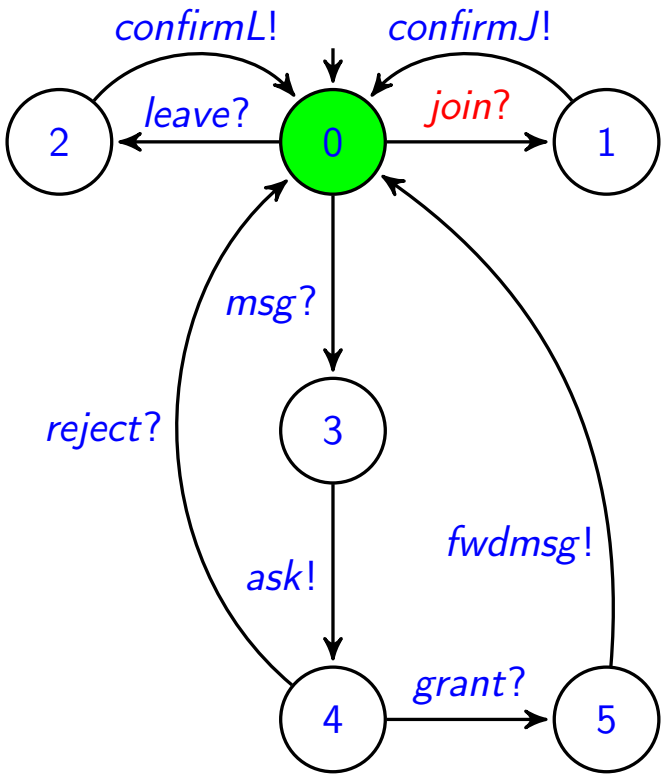
Not all system transitions are meaningful



(a) *Client*



(b) *Arbiter*

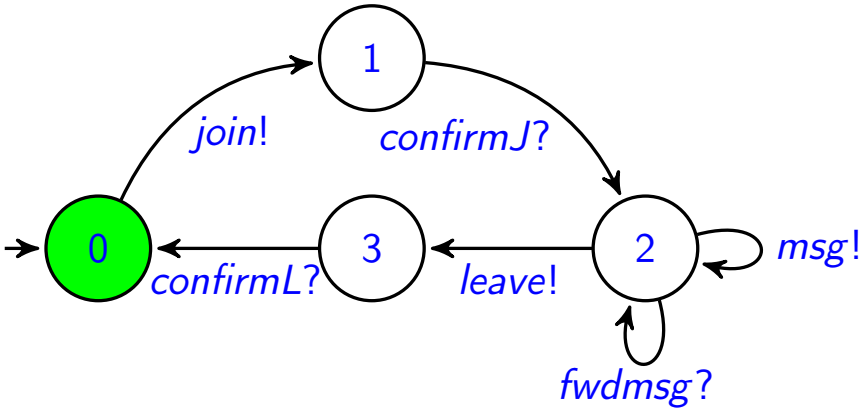


(c) *Server*

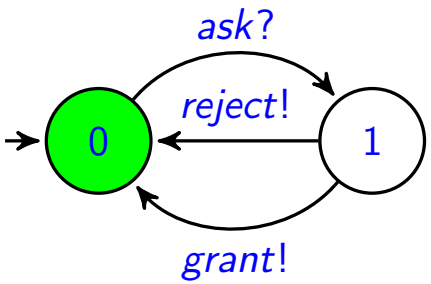
$(0, 0, 0, 0)$

✗ $(0, 0, 0, 0) \xrightarrow{join} (1, 0, 0, 0)$

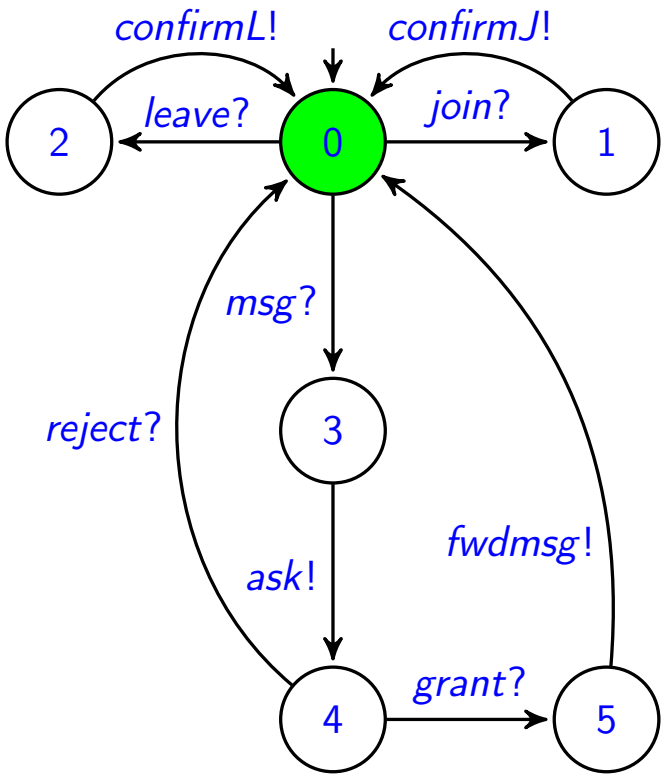
Not all system transitions are meaningful



(a) *Client*



(b) *Arbiter*

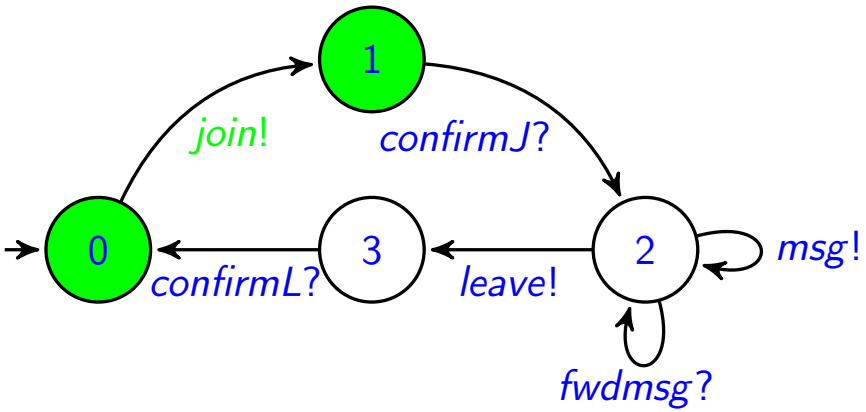


(c) *Server*

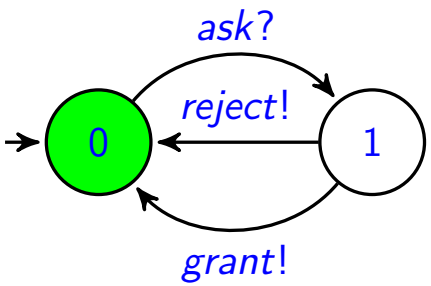
$(0, 0, 0, 0)$

✗ $(0, 0, 0, 0) \xrightarrow{join} (1, 0, 0, 0)$

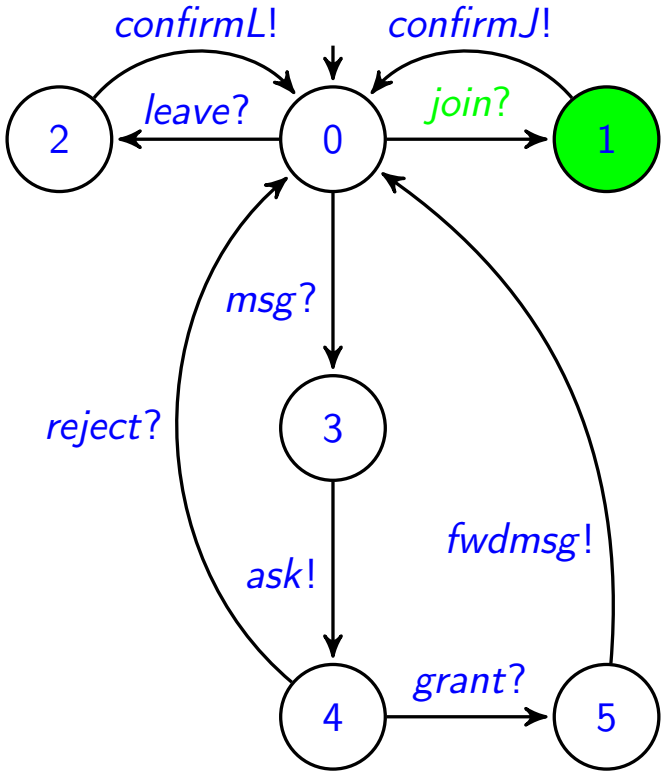
Not all system transitions are meaningful



(a) *Client*



(b) *Arbiter*



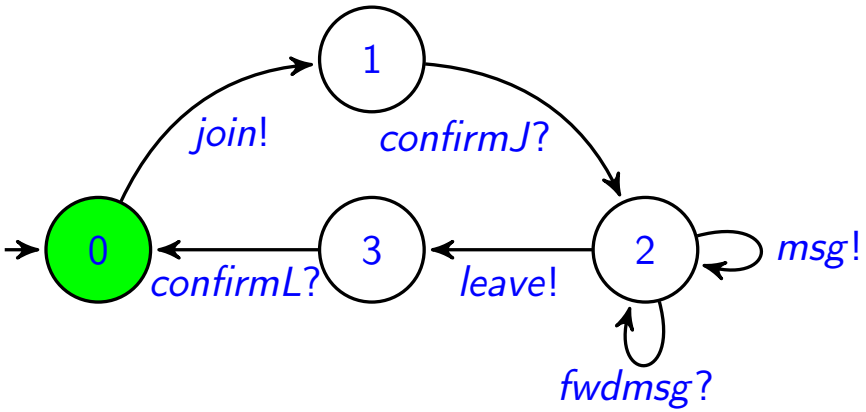
(c) *Server*

$(0, 0, 0, 0)$

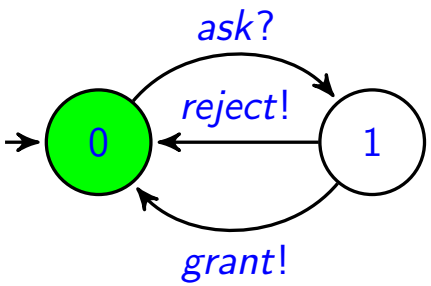
✗ $(0, 0, 0, 0) \xrightarrow{join} (1, 0, 0, 0)$

✓ $(0, 0, 0, 0) \xrightarrow{join} (1, 0, 1, 0)$

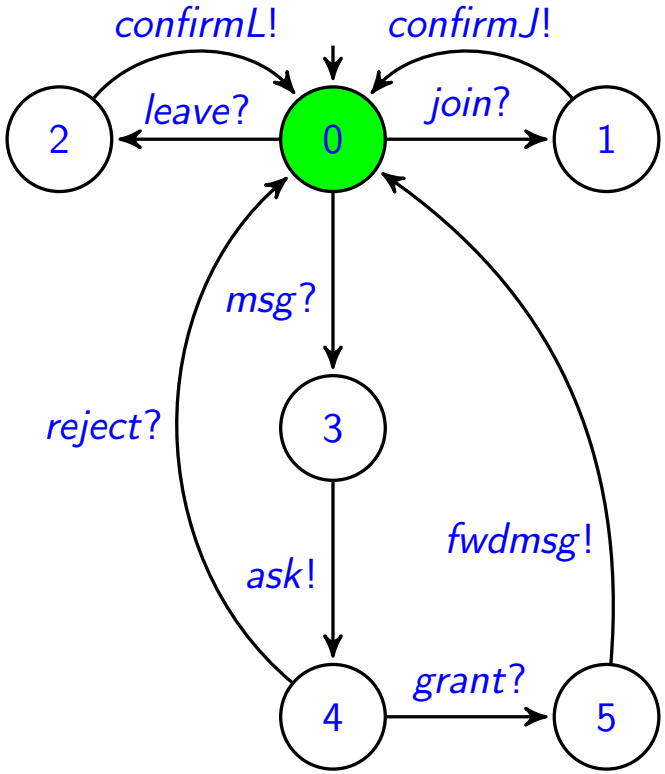
Not all system transitions are meaningful



(a) *Client*



(b) *Arbiter*



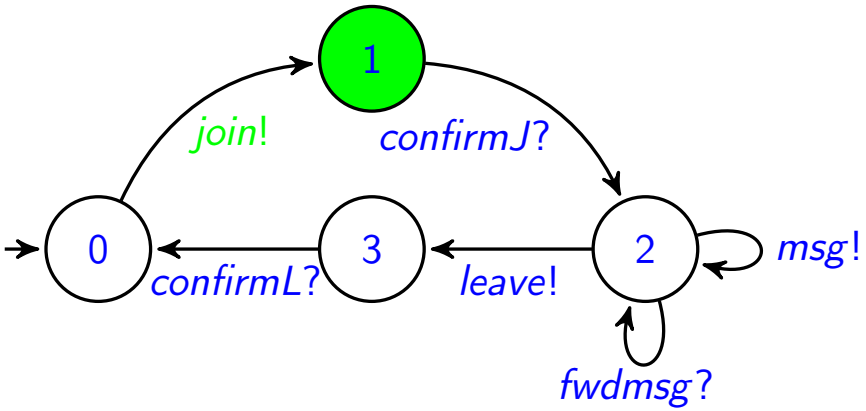
(c) *Server*

$(0, 0, 0, 0)$

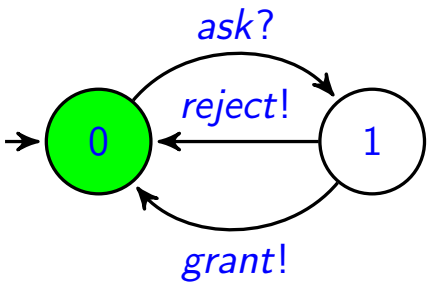
✗ $(0, 0, 0, 0) \xrightarrow{join} (1, 0, 0, 0)$

✓ $(0, 0, 0, 0) \xrightarrow{join} (1, 0, 1, 0)$

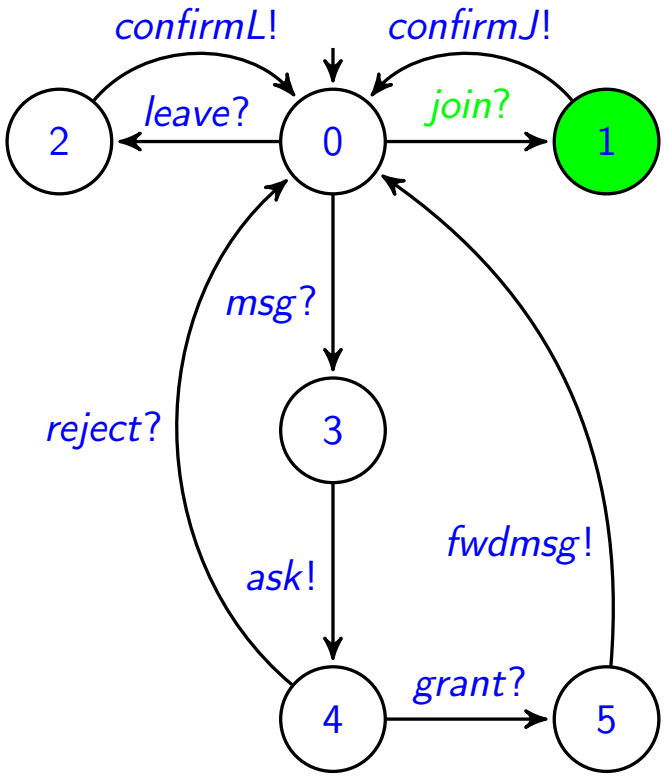
Not all system transitions are meaningful



(a) *Client*



(b) *Arbiter*



(c) *Server*

$(0,0,0,0)$

✗ $(0,0,0,0) \xrightarrow{join} (1,0,0,0)$

✓ $(0,0,0,0) \xrightarrow{join} (1,0,1,0)$

? $(0,0,0,0) \xrightarrow{join} (1,1,1,0)$

Synchronisation types

Idea from COORDINATION'17:

A synchronisation type is a pair $(\texttt{snd}, \texttt{rcv})$ that consists of a sending multiplicity $\texttt{snd}$ and a receiving multiplicity $\texttt{rcv}$

Synchronisation types

Idea from COORDINATION'17:

A synchronisation type is a pair $(\texttt{snd}, \texttt{rcv})$ that consists of a sending multiplicity $\texttt{snd}$ and a receiving multiplicity $\texttt{rcv}$

$([1,1],[1,1])$ binary, peer-to-peer communication

Synchronisation types

Idea from COORDINATION'17:

A synchronisation type is a pair $(\texttt{snd}, \texttt{rcv})$ that consists of a sending multiplicity $\texttt{snd}$ and a receiving multiplicity $\texttt{rcv}$

$([1,1],[0,1])$ binary, peer-to-peer communication, lossy

Synchronisation types

Idea from COORDINATION'17:

A synchronisation type is a pair $(\texttt{snd}, \texttt{rcv})$ that consists of a sending multiplicity $\texttt{snd}$ and a receiving multiplicity $\texttt{rcv}$

$([1,1],[0,1])$	binary, peer-to-peer communication, lossy
$([0,1],[0,1])$	non-blocking peer-to-peer (CCS)

Synchronisation types

Idea from COORDINATION'17:

A synchronisation type is a pair $(\texttt{snd}, \texttt{rcv})$ that consists of a sending multiplicity $\texttt{snd}$ and a receiving multiplicity $\texttt{rcv}$

$([1,1],[0,1])$	binary, peer-to-peer communication, lossy
$([0,1],[0,1])$	non-blocking peer-to-peer (CCS)
$([1,1],[0,*])$	multicast

Synchronisation types

Idea from COORDINATION'17:

A synchronisation type is a pair $(\texttt{snd}, \texttt{rcv})$ that consists of a sending multiplicity $\texttt{snd}$ and a receiving multiplicity $\texttt{rcv}$

$([1,1],[0,1])$	binary, peer-to-peer communication, lossy
$([0,1],[0,1])$	non-blocking peer-to-peer (CCS)
$([1,1],[1,*])$	multicast, strong

Synchronisation types

Idea from COORDINATION'17:

A synchronisation type is a pair $(\texttt{snd}, \texttt{rcv})$ that consists of a sending multiplicity $\texttt{snd}$ and a receiving multiplicity $\texttt{rcv}$

$([1,1],[0,1])$	binary, peer-to-peer communication, lossy
$([0,1],[0,1])$	non-blocking peer-to-peer (CCS)
$([1,1],[1,*])$	multicast, strong
$([1,1],\texttt{si})$	broadcast

Synchronisation types

Idea from COORDINATION'17:

A synchronisation type is a pair $(\texttt{snd}, \texttt{rcv})$ that consists of a sending multiplicity $\texttt{snd}$ and a receiving multiplicity $\texttt{rcv}$

$([1,1],[0,1])$	binary, peer-to-peer communication, lossy
$([0,1],[0,1])$	non-blocking peer-to-peer (CCS)
$([1,1],[1,*])$	multicast, strong
$([1,1],\texttt{ai})$	broadcast, strong

Synchronisation types

Idea from COORDINATION'17:

A synchronisation type is a pair $(\texttt{snd}, \texttt{rcv})$ that consists of a sending multiplicity $\texttt{snd}$ and a receiving multiplicity $\texttt{rcv}$

$([1,1],[0,1])$	binary, peer-to-peer communication, lossy
$([0,1],[0,1])$	non-blocking peer-to-peer (CCS)
$([1,1],[1,*])$	multicast, strong
$([1,1],\texttt{ai})$	broadcast, strong
$([1,*],[0,*])$	master-slave communication, 'slaves never proceed alone'

Synchronisation types

Idea from COORDINATION'17:

A synchronisation type is a pair $(\texttt{snd}, \texttt{rcv})$ that consists of a sending multiplicity $\texttt{snd}$ and a receiving multiplicity $\texttt{rcv}$

$([1,1],[0,1])$	binary, peer-to-peer communication, lossy
$([0,1],[0,1])$	non-blocking peer-to-peer (CCS)
$([1,1],[1,*])$	multicast, strong
$([1,1],\texttt{ai})$	broadcast, strong
$([1,*],[1,*])$	master-slave communication, strong, 'slaves must obey (at least one)'

Synchronisation types

Idea from COORDINATION'17:

A synchronisation type is a pair $(\texttt{snd}, \texttt{rcv})$ that consists of a sending multiplicity $\texttt{snd}$ and a receiving multiplicity $\texttt{rcv}$

$([1,1],[0,1])$	binary, peer-to-peer communication, lossy
$([0,1],[0,1])$	non-blocking peer-to-peer (CCS)
$([1,1],[1,*])$	multicast, strong
$([1,1],\texttt{ai})$	broadcast, strong
$([1,*],[1,*])$	master-slave communication, strong, 'slaves must obey (at least one)'
$(\texttt{ai},\texttt{ai})$	full synchronisation (FSP)

Synchronisation types

Idea from COORDINATION'17:

A synchronisation type is a pair $(\texttt{snd}, \texttt{rcv})$ that consists of a sending multiplicity $\texttt{snd}$ and a receiving multiplicity $\texttt{rcv}$

$([1,1],[0,1])$	binary, peer-to-peer communication, lossy
$([0,1],[0,1])$	non-blocking peer-to-peer (CCS)
$([1,1],[1,*])$	multicast, strong
$([1,1],\texttt{ai})$	broadcast, strong
$([1,*],[1,*])$	master-slave communication, strong, 'slaves must obey (at least one)'
$(\texttt{ai},\texttt{ai})$	full synchronisation (FSP)
$([1,*],[1,1])$	gathering

Synchronisation types

Idea from COORDINATION'17:

A synchronisation type is a pair $(\texttt{snd}, \texttt{rcv})$ that consists of a sending multiplicity $\texttt{snd}$ and a receiving multiplicity $\texttt{rcv}$

$([1,1],[0,1])$	binary, peer-to-peer communication, lossy
$([0,1],[0,1])$	non-blocking peer-to-peer (CCS)
$([1,1],[1,*])$	multicast, strong
$([1,1],\texttt{ai})$	broadcast, strong
$([1,*],[1,*])$	master-slave communication, strong, 'slaves must obey (at least one)'
$(\texttt{ai},\texttt{ai})$	full synchronisation (FSP)
$([1,*],[1,1])$	gathering
$([0,*],[0,*])$	all system transitions

Compliance with communication requirements

Basic ideas from COORDINATION'17:

Receptiveness requirement:

Whenever (a group of) components want to send a message, there should be (a group of) components ready to receive the message in conformance with the synchronisation type

Compliance with communication requirements

Basic ideas from COORDINATION'17:

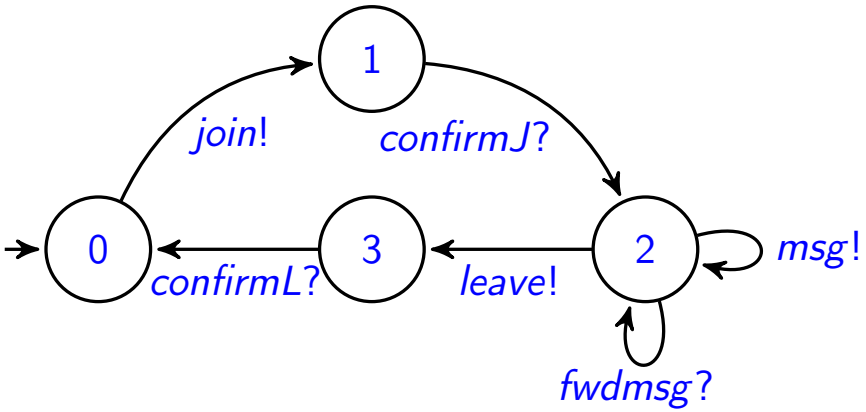
Receptiveness requirement:

Whenever (a group of) components want to send a message, there should be (a group of) components ready to receive the message in conformance with the synchronisation type

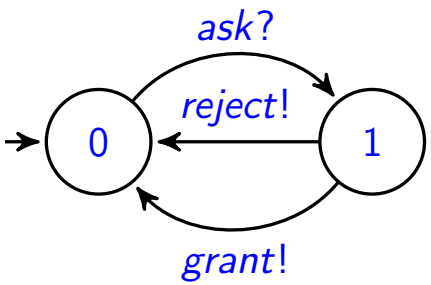
Responsiveness requirement:

Whenever (a group of) components wait to receive one of the messages $a_1, \ldots, a_n$, there should be (a group of) components able to send (at least) one of the messages $a_1, \ldots, a_n$ in conformance with the synchronisation type (*external choice*)

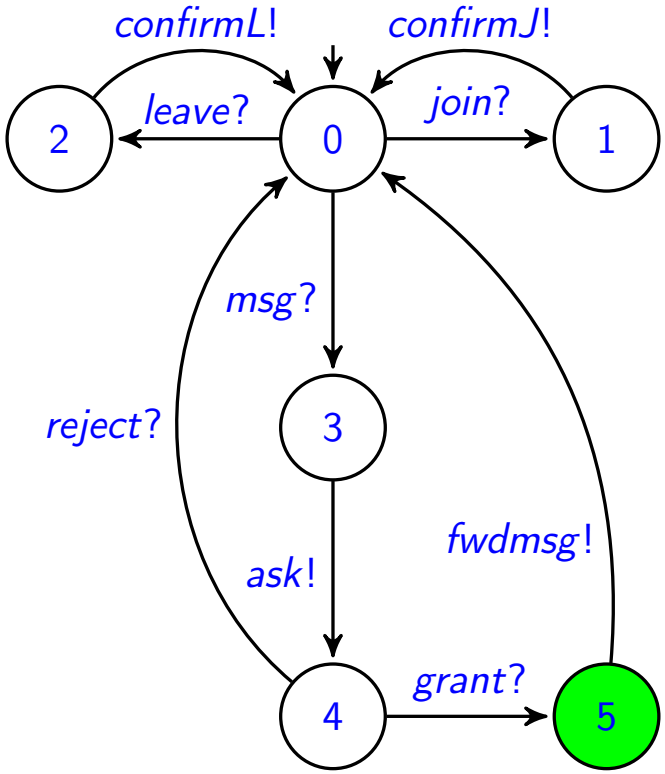
Example: communication requirements



(a) *Client*



(b) *Arbiter*

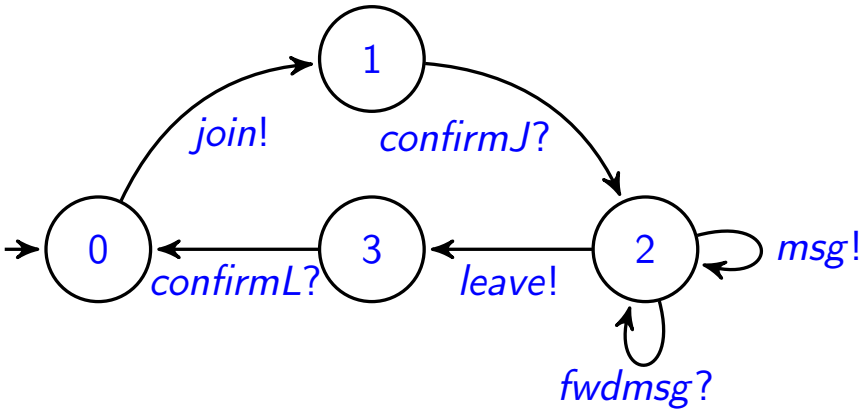


(c) *Server*

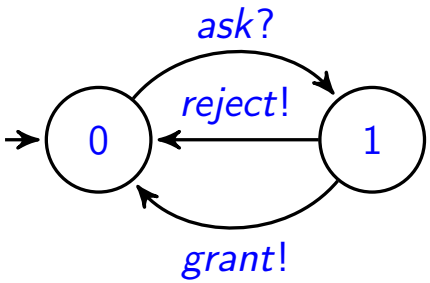
$([1,1],[1,*])$

✓ $\texttt{rcp}(\mathcal{S}erver, fwdmsg)@(q_1, q_2, 5, q_4)$

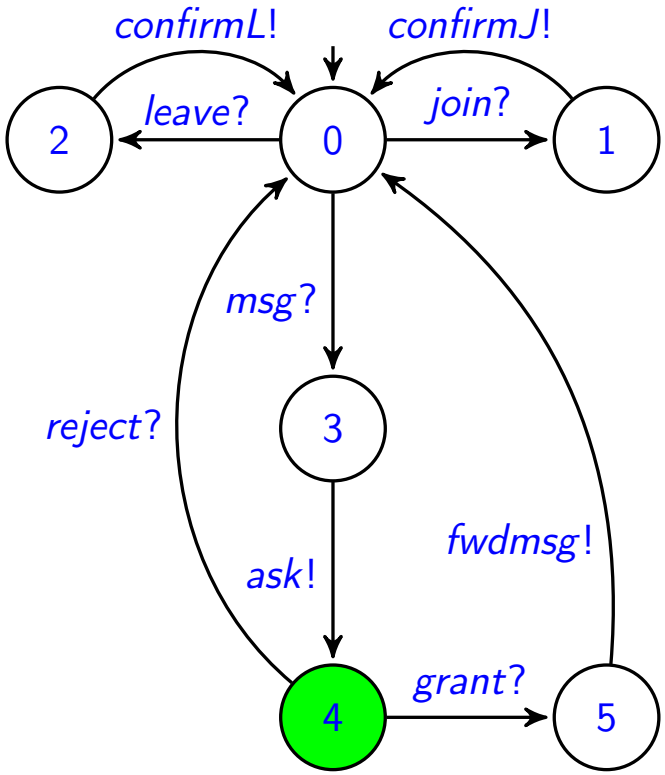
Example: communication requirements



(a) *Client*



(b) *Arbiter*



(c) *Server*

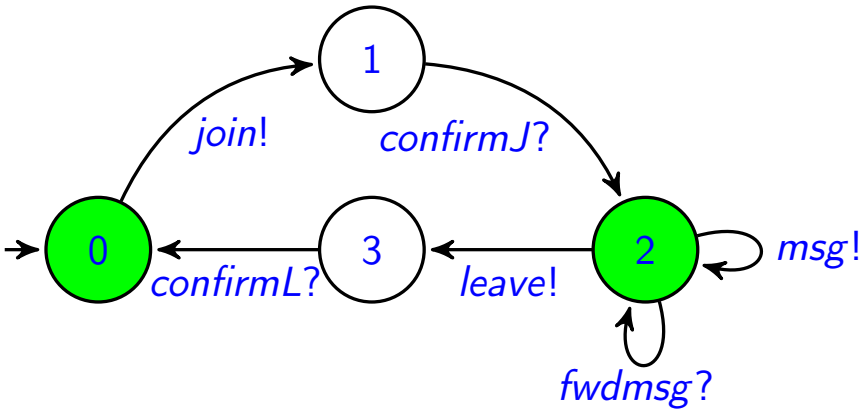
$([1,1],[1,*])$

- ✓ $\texttt{rcp}(\mathcal{Server}, \mathit{fwdmsg})@(q_1, q_2, 5, q_4)$
- ✓ $\texttt{rsp}((\mathcal{Server}, \mathit{reject}) \vee (\mathcal{Server}, \mathit{grant}))@(q_1, q_2, 4, q_4)$

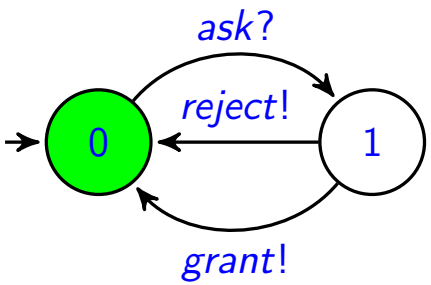
First extension

A less restrictive notion of compatibility (weak compliance), where intermediate communication between other components is allowed before a desired communication (reception or response) can occur (as opposed to only *internal actions*)

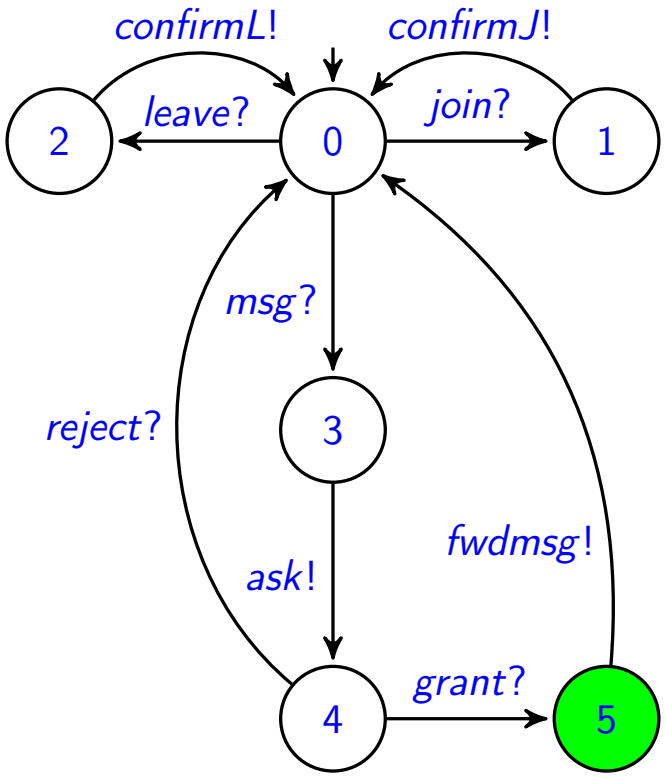
Compliance: intermediate communications



(a) *Client*



(b) *Arbiter*

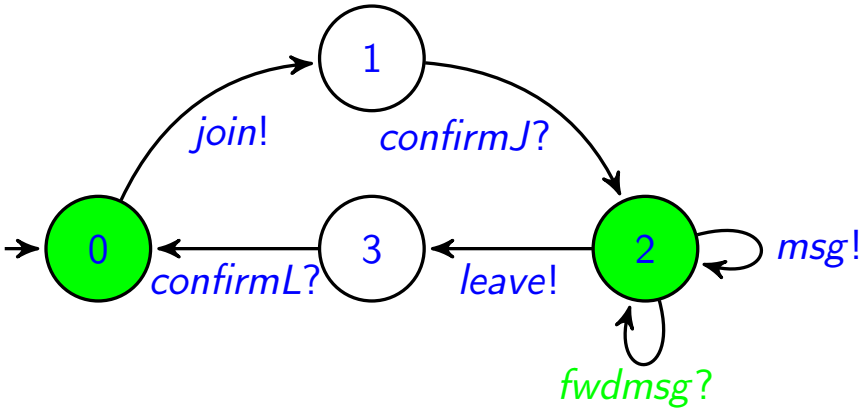


(c) *Server*

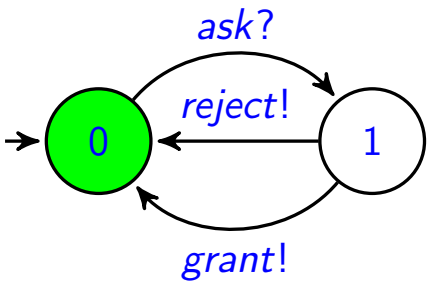
$([1,1],[1,*])$

✗ $\texttt{rcp}(\mathcal{C}lient_1, join)@(0,2,5,0)$

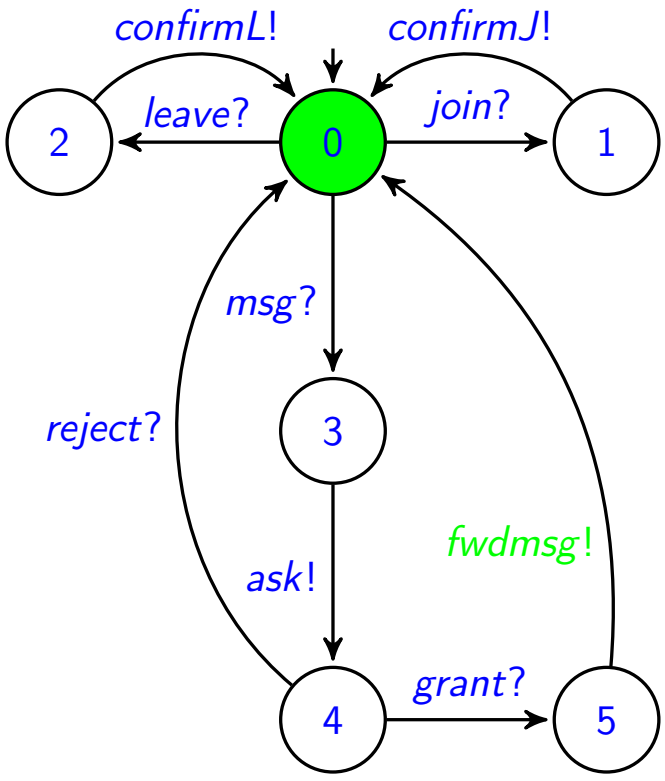
Compliance: intermediate communications



(a) *Client*



(b) *Arbiter*



(c) *Server*

$([1,1],[1,*])$

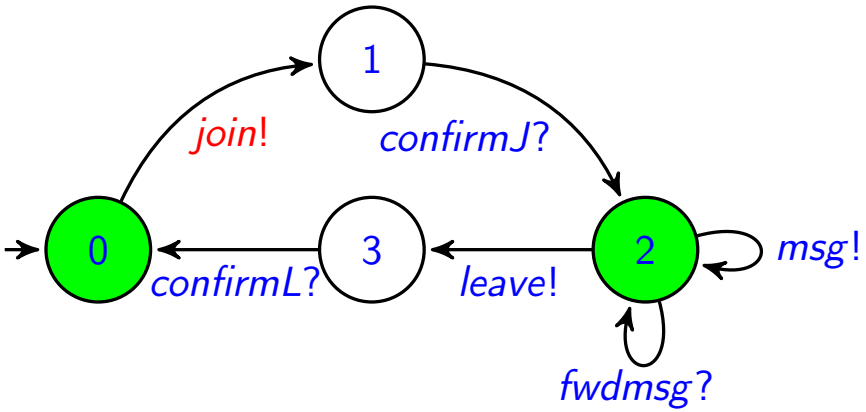
✗ $\texttt{rcp}(\mathcal{C}lient_1, join)@(0,2,5,0)$

✓ $\texttt{rcp}(\mathcal{C}lient_1, join)@(0,2,0,0)$

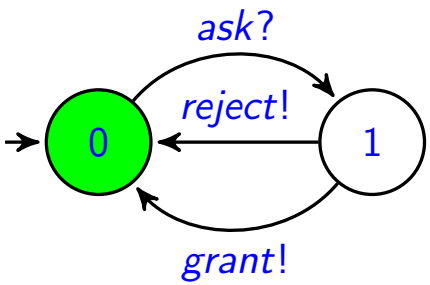
Second extension

Associate individual synchronisation types to each output/input action to fine tune the number of participating sending/receiving components *per action*

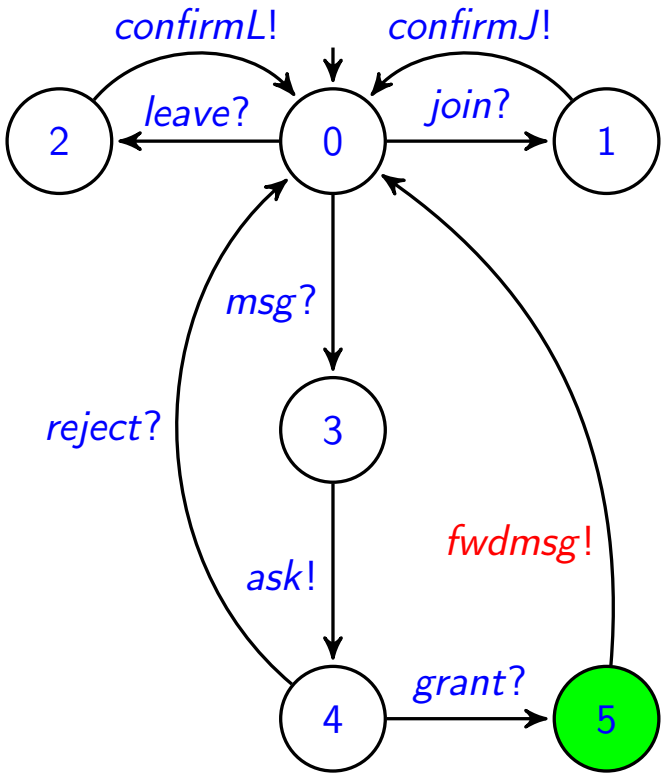
Actions: individual synchronisation types



(a) *Client*



(b) *Arbiter*

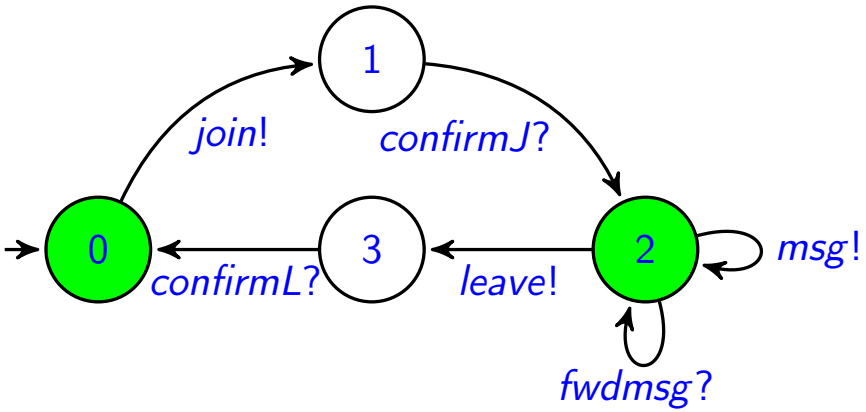


(c) *Server*

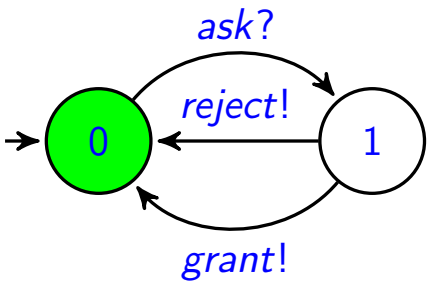
$([1,1],[1,*])$

✗ $\mathtt{rcp}((\mathcal{C}lient_2, leave) \wedge (\mathcal{C}lient_2, msg))@(0,2,5,0)$

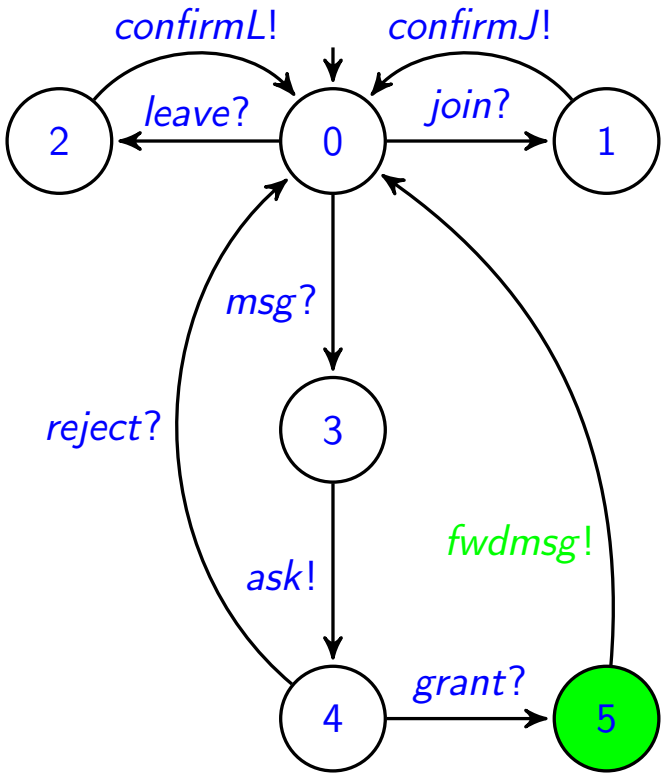
Actions: individual synchronisation types



(a) *Client*



(b) *Arbiter*

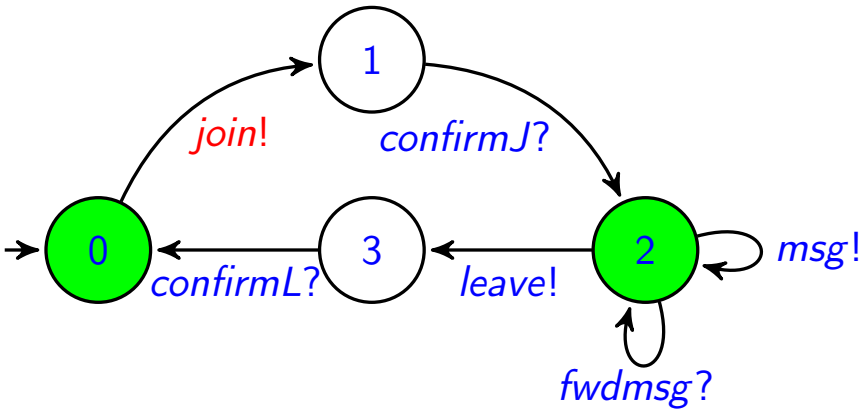


(c) *Server*

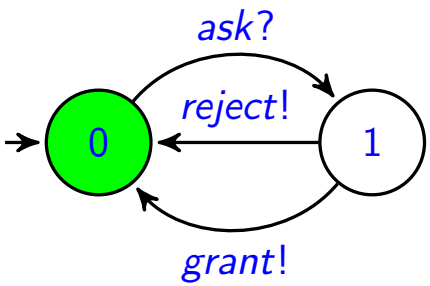
$([1,1],[0,*])$

✓ $\texttt{rcp}((\mathcal{C}lient_2, leave) \wedge (\mathcal{C}lient_2, msg))@(0,2,5,0)$

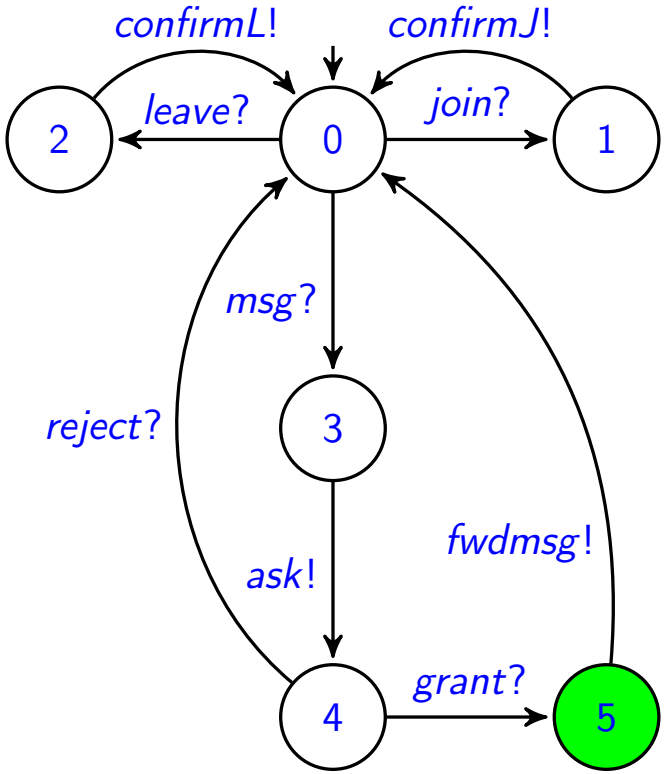
Actions: individual synchronisation types



(a) *Client*



(b) *Arbiter*



(c) *Server*

$([1,1],[0,*])$

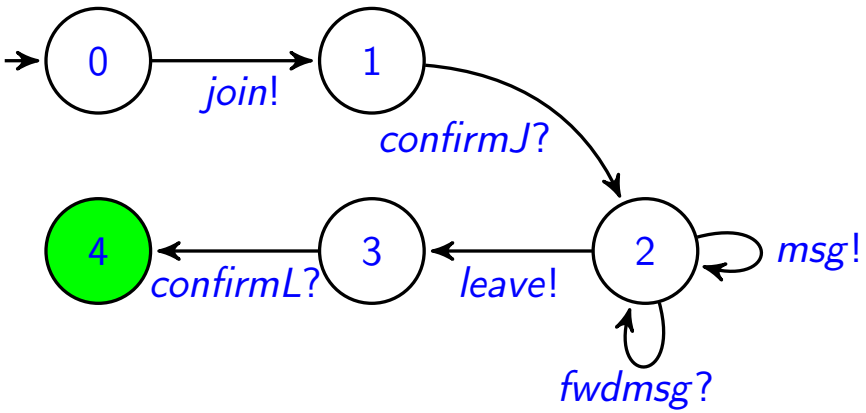
✓ $\mathtt{rcp}((\mathit{Client}_2, \mathit{leave}) \wedge (\mathit{Client}_2, \mathit{msg}))@(0,2,5,0)$

✗ $(0,2,5,0) \xrightarrow{\mathit{join}} (1,2,5,0)$

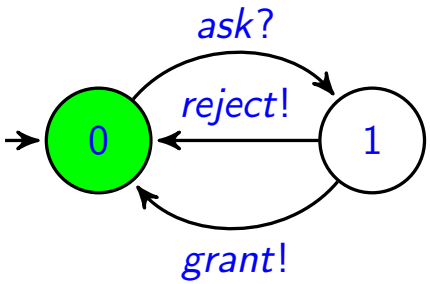
Third extension

A distinction between states were continuation is required and final states where it is possible yet not required

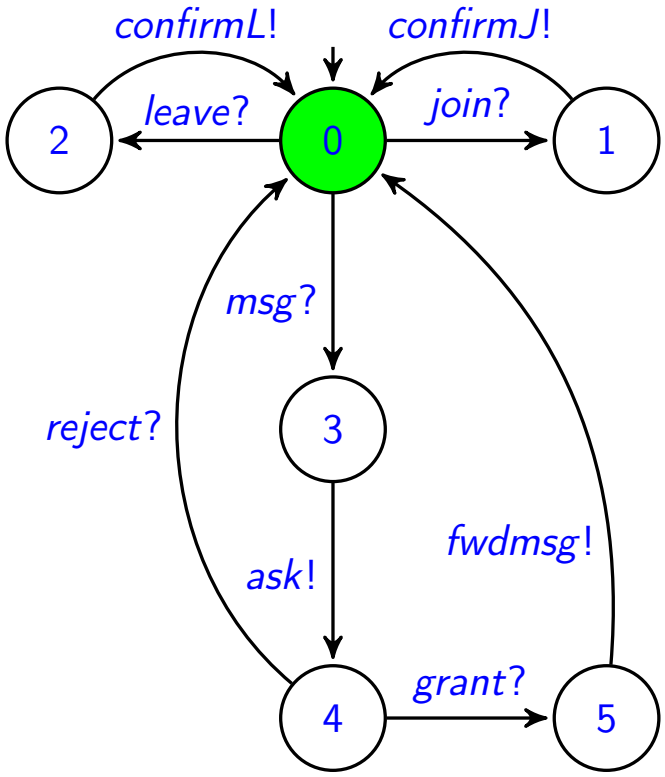
States: final states



(a) *Client*



(b) *Arbiter*

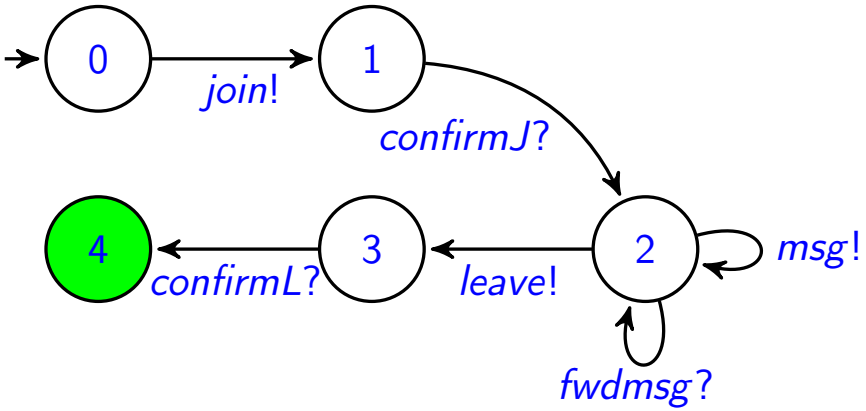


(c) *Server*

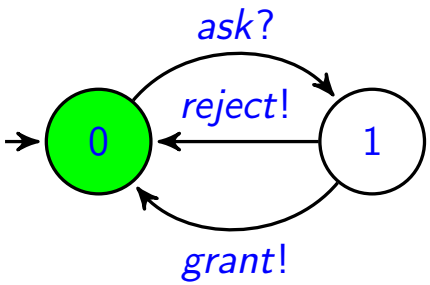
$([1,1],[1,*])$

✗ $\texttt{rsp}((\mathcal{S}erver, join) \vee (\mathcal{S}erver, leave) \vee (\mathcal{S}erver, msg))@(4,4,0,0)$

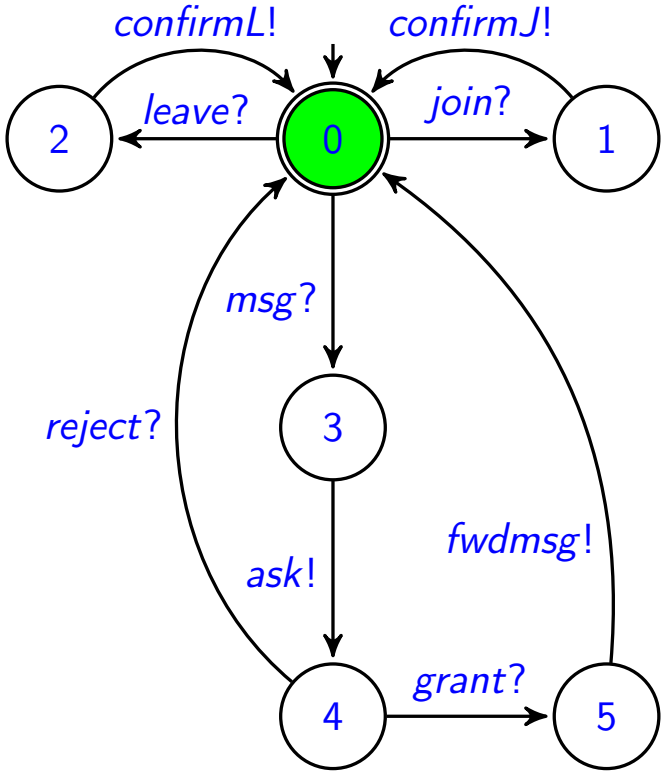
States: final states



(a) *Client*



(b) *Arbiter*

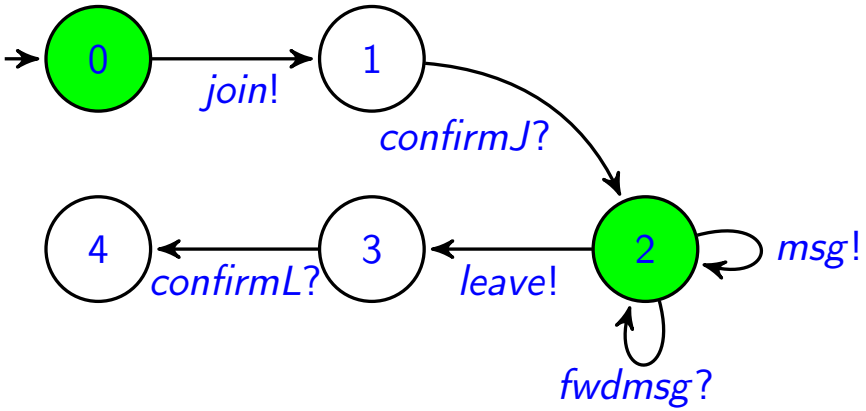


(c) *Server*

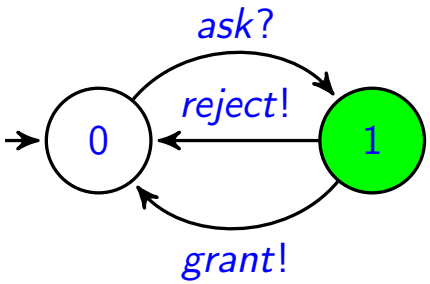
$([1,1],[1,*])$

~~✗ $\mathtt{rsp}((\mathit{Server},\mathit{join}) \vee (\mathit{Server},\mathit{leave}) \vee (\mathit{Server},\mathit{msg}))@(4,4,0,0)$~~

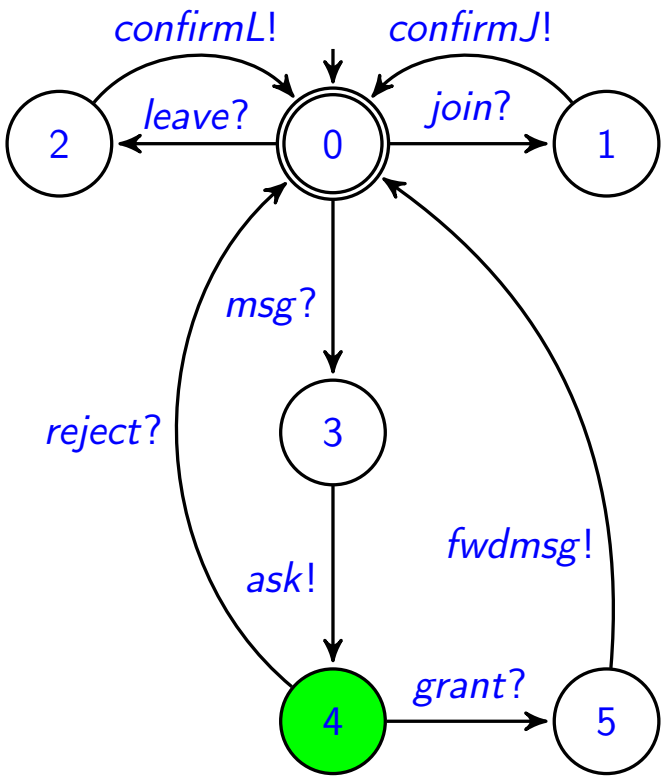
States: final states



(a) *Client*



(b) *Arbiter*



(c) *Server*

$([1,1],[1,*])$

~~✗ $\mathtt{rsp}((\mathit{Server},\mathit{join}) \vee (\mathit{Server},\mathit{leave}) \vee (\mathit{Server},\mathit{msg}))@(4,4,0,0)$~~

✓ $\mathtt{rsp}((\mathit{Server},\mathit{reject}) \vee (\mathit{Server},\mathit{grant}))@(0,2,4,1)$

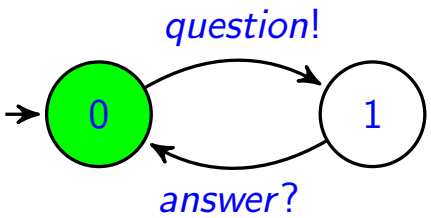
Conclusion

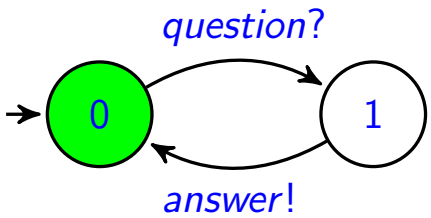
- Meta-theory for communication-safety (compatibility) in multi-component systems applicable to many concrete synchronisation types
- Means to compare compatibility notions from the literature (for closed/open systems, pessimistic/optimistic receptiveness, strong/weak compatibility)

Conclusion

- Meta-theory for communication-safety (compatibility) in multi-component systems applicable to many concrete synchronisation types
- Means to compare compatibility notions from the literature (for closed/open systems, pessimistic/optimistic receptiveness, strong/weak compatibility)
- Submitted:
  - ! powerful weak(er) compliance, individual synchronisation types
  - ! preservation of communication safety by composition of systems of team automata
- Future work:
  - final states
  - larger case studies
  - asynchronous communication

Thanks for your attention!





$\mathcal{S}peaker$ $\mathcal{A}udience$

$([1,1],[1,1])$

$\texttt{rsp}(\mathcal{S}peaker, question)@(0,0)$