

Asynchronous Team Automata

Davide Basile

FMT, CNR-ISTI, Pisa, Italy

Maurice ter Beek

FMT, CNR-ISTI, Pisa, Italy

José Proença

CISTER & INESC TEC & U. Porto, Portugal



FM 2026, Tokyo, Japan, May 22nd, 2026

- 25+ Years of Team Automata
 - Motivation and development
 - Constrained multi-party synchronisations

- Asynchronous Team Automata
 - Inspiration and challenge
 - Syntax and semantics by example
 - Well-formedness and well-behavedness
 - Tool support: A-Team ([demo](#))

- Conclusion and Future Work

25+ Years of Team Automata

A formalism for interacting component-based systems, whereby multiple **sending** and **receiving** actions from concurrent automata can **synchronise** on certain executions

First proposed at the 1997 ACM SIGGROUP Conference on Supporting Group Work for modelling components of groupware systems and their interconnections

[GROUP'97]



Inspired by Input/Output (I/O) automata, inheriting the distinction between **internal** and external (**input** and **output**) actions used for communication with the environment

A formalism for interacting component-based systems, whereby multiple **sending** and **receiving** actions from concurrent automata can **synchronise** on certain executions

First proposed at the 1997 ACM SIGGROUP Conference on Supporting Group Work for modelling components of groupware systems and their interconnections

[GROUP'97]



Inspired by Input/Output (I/O) automata, inheriting the distinction between **internal** and external (**input** and **output**) actions used for communication with the environment

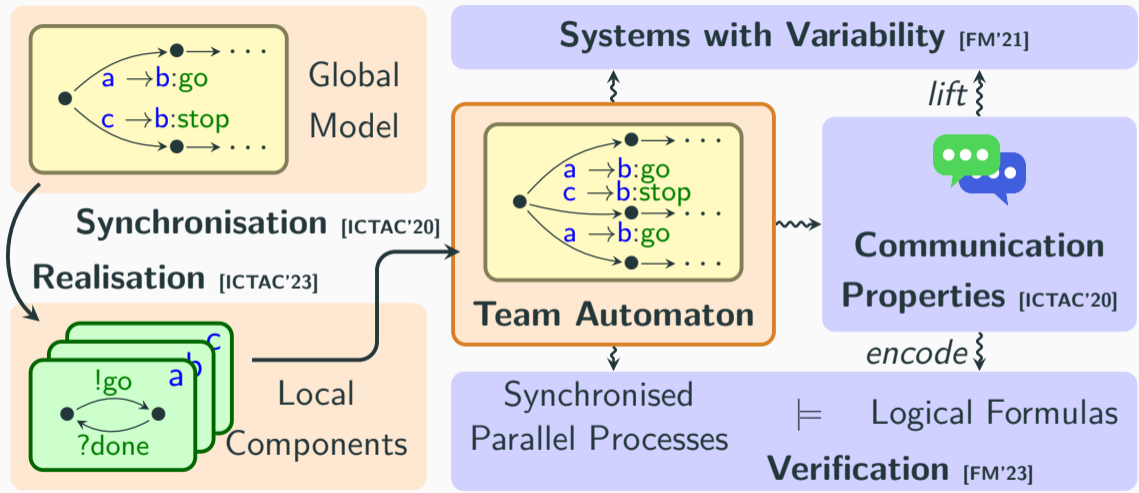
Formally defined in Computer Supported Cooperative Work (CSCW)—The Journal of Collaborative Computing, as composed by **component automata** that synchronise

[CSCW'03]



Technically an extension of I/O automata, imposing **hardly any restrictions on the role of actions** in components, while **composition is not limited to the synchronous product**

[IPL'05]

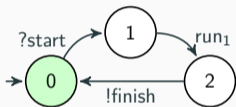


[FACS'23]: Overview on Constrained Multiparty Synchronisation in Team Automata

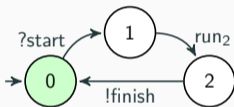
[COORDINATION'24]: Team Automata: Overview and Roadmap.

Constrained Multi-party Synchronisations

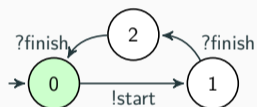
Team Automata: **not all system transitions are meaningful!**



Runner₁



Runner₂



Controller

Team Automata

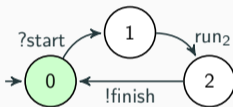
[CSCW'01'03] [FM'03,'21,'23] [TCS'12'13]

[COORDINATION'17,'20,'24] [ICTAC'20,'23]

Team Automata: **not all system transitions are meaningful!**



$Runner_1$



$Runner_2$



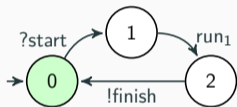
$Controller$

Team Automata

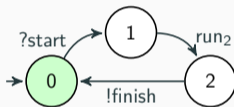
[CSCW'01'03] [FM'03,'21,'23] [TCS'12'13]

[COORDINATION'17,'20,'24] [ICTAC'20,'23]

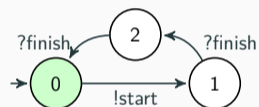
Team Automata: **not all system transitions are meaningful!**



Runner₁



Runner₂



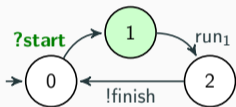
Controller

Team Automata

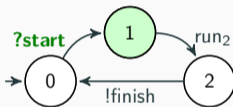
[CSCW'01'03] [FM'03,'21,'23] [TCS'12'13]

[COORDINATION'17,'20,'24] [ICTAC'20,'23]

Team Automata: **not all system transitions are meaningful!**



$Runner_1$



$Runner_2$



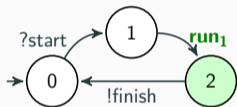
$Controller$

Team Automata

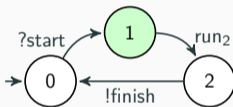
[CSCW'01'03] [FM'03,'21,'23] [TCS'12'13]

[COORDINATION'17,'20,'24] [ICTAC'20,'23]

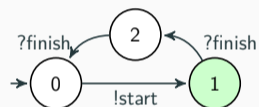
Team Automata: **not all system transitions are meaningful!**



$Runner_1$



$Runner_2$



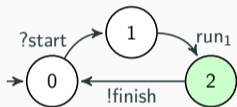
$Controller$

Team Automata

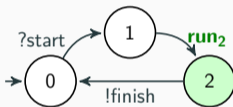
[CSCW'01'03] [FM'03,'21,'23] [TCS'12'13]

[COORDINATION'17,'20,'24] [ICTAC'20,'23]

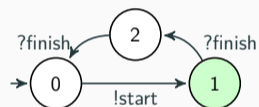
Team Automata: **not all system transitions are meaningful!**



$Runner_1$



$Runner_2$



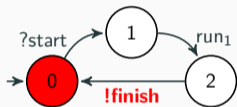
$Controller$

Team Automata

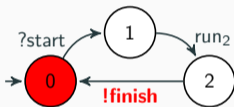
[CSCW'01'03] [FM'03,'21,'23] [TCS'12'13]

[COORDINATION'17,'20,'24] [ICTAC'20,'23]

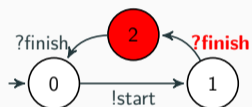
Team Automata: **not all system transitions are meaningful!**



$Runner_1$



$Runner_2$



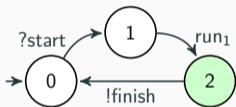
$Controller$

Team Automata

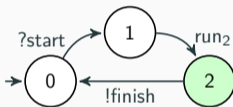
[CSCW'01'03] [FM'03,'21,'23] [TCS'12'13]

[COORDINATION'17,'20,'24] [ICTAC'20,'23]

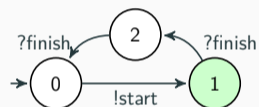
Team Automata: **not all system transitions are meaningful!**



$Runner_1$



$Runner_2$



$Controller$

Team Automata

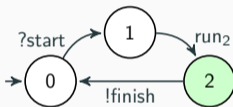
[CSCW'01'03] [FM'03,'21,'23] [TCS'12'13]

[COORDINATION'17,'20,'24] [ICTAC'20,'23]

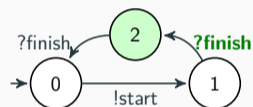
Team Automata: **not all system transitions are meaningful!**



$Runner_1$



$Runner_2$



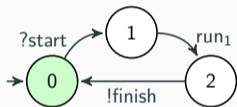
$Controller$

Team Automata

[CSCW'01'03] [FM'03,'21,'23] [TCS'12'13]

[COORDINATION'17,'20,'24] [ICTAC'20,'23]

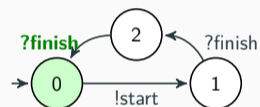
Team Automata: **not all system transitions are meaningful!**



$Runner_1$



$Runner_2$



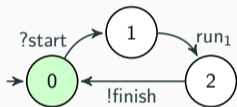
$Controller$

Team Automata

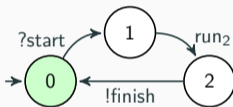
[CSCW'01'03] [FM'03,'21,'23] [TCS'12'13]

[COORDINATION'17,'20,'24] [ICTAC'20,'23]

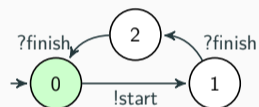
Extended Team Automata: Constrained Multi-party Synchronisations



Runner₁



Runner₂



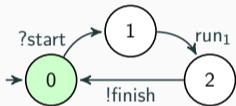
Controller

Extended TA synchronisations

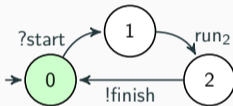
[CSCW'01'03] [FM'03,'21,'23] [TCS'12'13]

[COORDINATION'17,'20,'24] [ICTAC'20,'23]

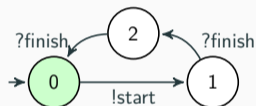
Extended Team Automata: Constrained Multi-party Synchronisations



$Runner_1$



$Runner_2$



$Controller$

Extended TA synchronisations

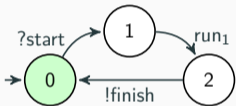
[CSCW'01'03] [FM'03,'21,'23] [TCS'12'13]

[COORDINATION'17,'20,'24] [ICTAC'20,'23]

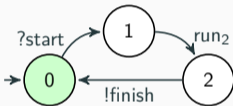
multi-party

$Ctr \rightarrow \{R1, R2\}: start$

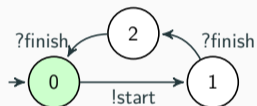
Extended Team Automata: Constrained Multi-party Synchronisations



$Runner_1$



$Runner_2$



$Controller$

Extended TA synchronisations

[CSCW'01'03] [FM'03,'21,'23] [TCS'12'13]

[COORDINATION'17,'20,'24] [ICTAC'20,'23]

multi-party

$Ctr \rightarrow \{R1, R2\}: start$

constrained

$start: 1 \rightarrow 2$

$finish: 1 \rightarrow 1$

Asynchronous Communication in Team Automata



Inspiration from **multi-composition** of asynchronous systems of CFSMs

D. Brand and P. Zafiropulo, On Communicating Finite-State Machines. *Journal of the ACM* 30 (1983)

F. Barbanera and R. Hennicker, Safe Composition of Systems of Communicating Finite State Machines @ ICE'24



Inspiration from **multi-composition** of asynchronous systems of CFSMs

D. Brand and P. Zafiropulo, On Communicating Finite-State Machines. *Journal of the ACM* 30 (1983)

F. Barbanera and R. Hennicker, Safe Composition of Systems of Communicating Finite State Machines @ ICE'24

Basically finite-state I/O-automata that communicate by asynchronous exchanges of messages via buffered FIFO channels, but

- CFSMs use potentially infinite FIFO buffers as message channels
- CFSMs use **binary peer-to-peer communication** with 'rich' local actions

Behaviour of CFSMs formalised as a transition relation on **configurations** (i.e., pairs of a tuple of component states and a tuple of channel buffers)



Inspiration from **multi-composition** of asynchronous systems of CFSMs

D. Brand and P. Zafiropulo, On Communicating Finite-State Machines. *Journal of the ACM* 30 (1983)

F. Barbanera and R. Hennicker, Safe Composition of Systems of Communicating Finite State Machines @ ICE'24

Basically finite-state I/O-automata that communicate by asynchronous exchanges of messages via buffered FIFO channels, but

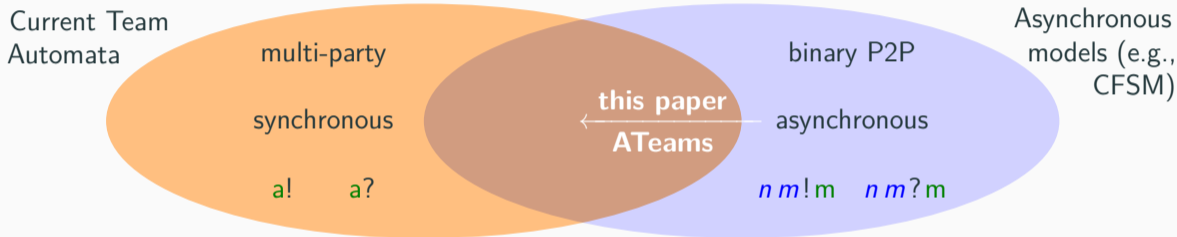
- CFSMs use potentially infinite FIFO buffers as message channels
- CFSMs use **binary peer-to-peer communication** with 'rich' local actions

Behaviour of CFSMs formalised as a transition relation on **configurations** (i.e., pairs of a tuple of component states and a tuple of channel buffers)

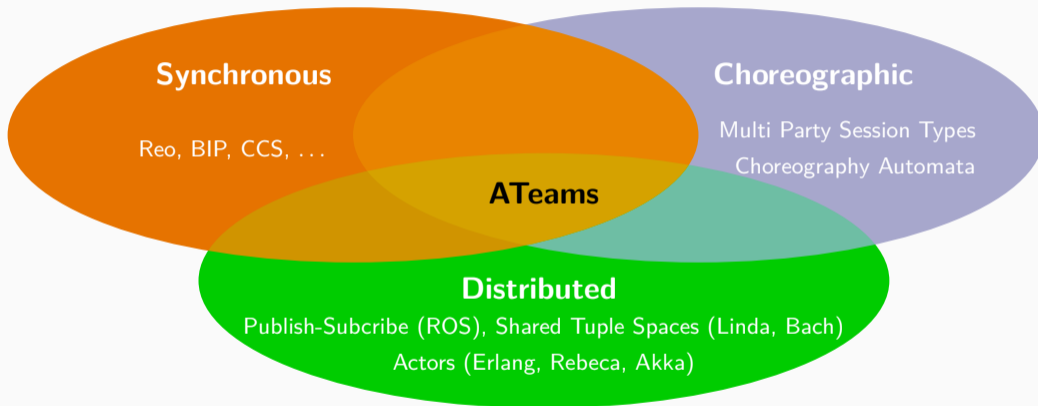


Generalise to asynchronous **multi-party communication** for team automata (incl. **receptiveness** and **responsiveness** properties in asynchronous setting)

This paper's challenge



“Every good talk should include a Venn diagram” – Einar Broch Johnsen (Lima, Peru, 2023)



What is a channel? (e.g. order?)

How to find a channel/agent?

Variations of the Race example in ATeams

```
// Synchronous Race
```

```
acts
```

```
start: 1->2, sync;  
finish: 1->1, sync;
```

```
proc
```

```
Ctr = start!.  
    finish?.finish?.  
    Ctr
```

```
R = start?.finish!.R
```

```
init
```

```
Ctr || R || R
```



```
// Async. Race (sender)
```

```
acts
```

```
start: 1->2, fifo@snd;  
finish: 2->1, fifo@snd;
```

```
proc
```

```
Ctr = start!.  
    finish?r1,r2.  
    Ctr
```

```
R = start?c.finish!.R
```

```
init
```

```
c:Ctr || r1:R || r2:R
```



```
// Async. Race (global)
```

```
acts
```

```
start: 1->2, fifo@global;  
finish:2->1, fifo@global;
```

```
proc
```

```
Ctr = start!.  
    finish?.  
    Ctr
```

```
R = start?.finish!.R
```

```
init
```

```
Ctr || R || R
```



A System consists of:

Action's types
(*acts*)

+

Process' definitions
(*proc*)

+

Agents (processes)
(*init*)

$$P := K \mid \mathbf{0} \mid \alpha.P \mid P + P$$

(CCS-like process)

$$\alpha := a! \mid a? \mid a!\bar{n} \mid a?\bar{n}$$

(action)

$$st := (\text{sync}, \text{intrv}, \text{intrv}) \mid (\text{async}, \text{intrv}, \text{intrv}, \text{buft}, \text{loct})$$

(synchronisation type)

$$\text{intrv} := (i, i) \mid (i, \infty)$$

(interval)

$$\text{buft} := \text{fifo} \mid \text{bag} \mid \dots$$

(buffer type)

$$\text{loct} := @\text{snd} \mid @\text{rcv} \mid @\text{global} \mid @\text{snd-rcv}$$

(location type)

```
// Synchronous Race
```

```
acts
```

```
start: 1->2, sync;
```

```
finish: 1->1, sync;
```

```
proc
```

```
  Ctr = start!.
```

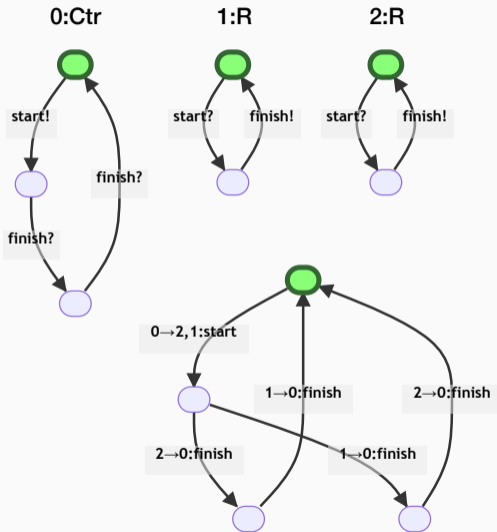
```
    finish?.finish?.
```

```
    Ctr
```

```
  R = start?.finish!.R
```

```
init
```

```
  Ctr || R || R
```



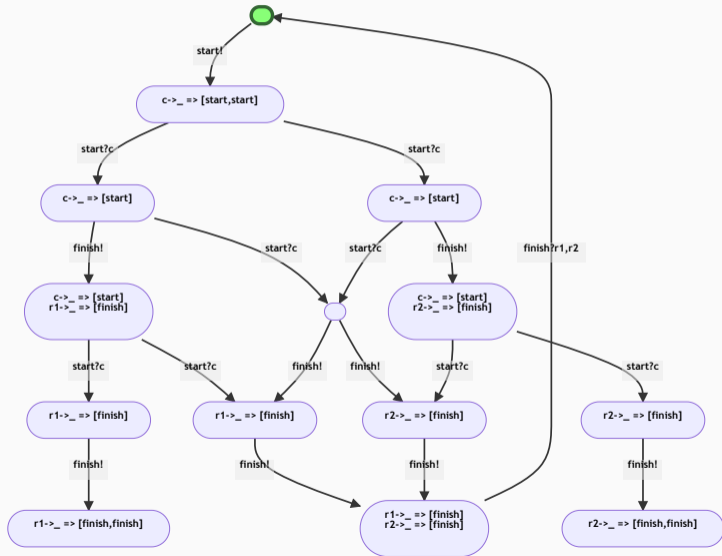
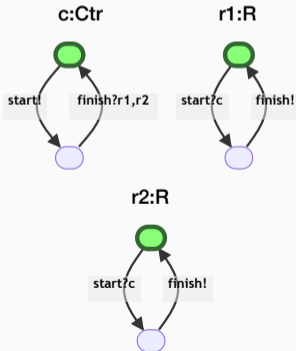
Semantics ATeams: asynchronous communication (sending)

```
// Async. Race (sender)
```

acts

```
start: 1->2, fifo@snd;
```

```
finish: 2->1, fifo@snd;
```



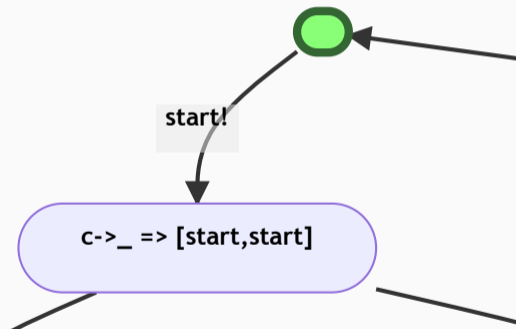
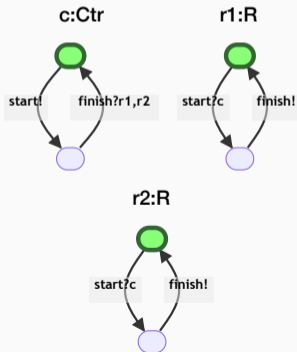
Semantics ATeams: asynchronous communication (sending)

```
// Async. Race (sender)
```

acts

```
start: 1->2, fifo@snd;
```

```
finish: 2->1, fifo@snd;
```



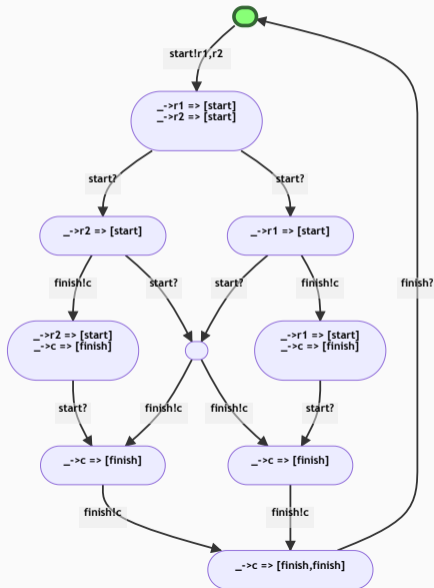
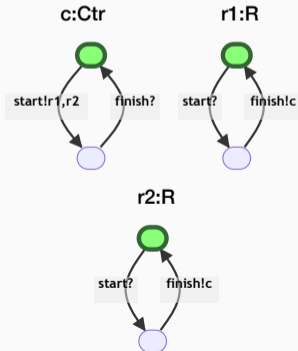
Semantics ATeams: asynchronous communication (receiving)

```
// Async. Race (receiver)
```

acts

```
start: 1->2, fifo@rcv;
```

```
finish: 2->1, fifo@rcv;
```

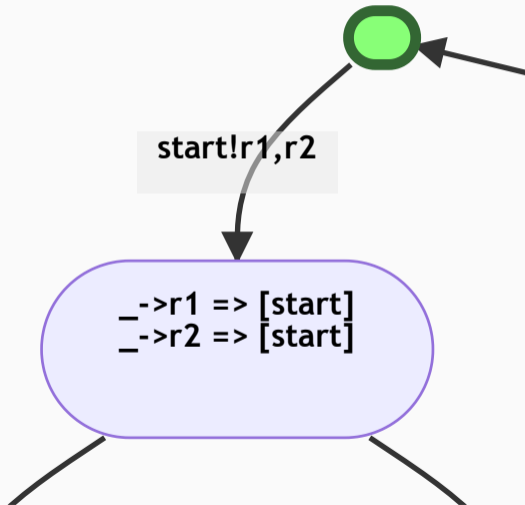
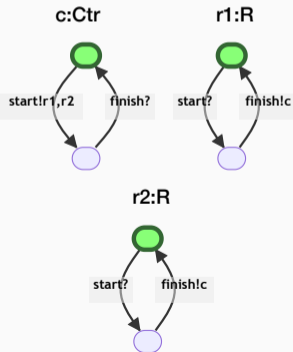


```
// Async. Race (receiver)
```

acts

```
start: 1->2, fifo@rcv;
```

```
finish: 2->1, fifo@rcv;
```



```
// Async. Race (global)
```

acts

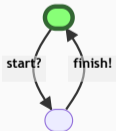
```
start: 1->2, fifo@global;
```

```
finish: 2->1, fifo@global;
```

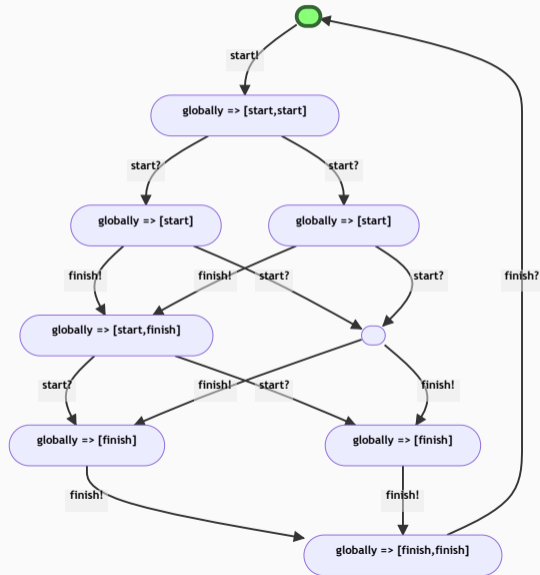
0:Ctr



1:R



2:R



```
// Async. Race (global)
```

```
acts
```

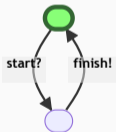
```
start: 1->2, fifo@global;
```

```
finish: 2->1, fifo@global;
```

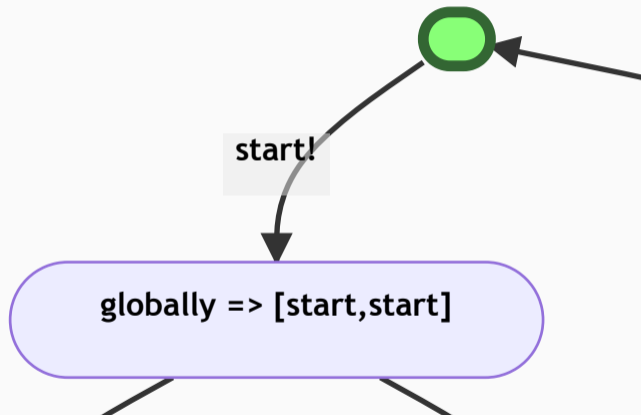
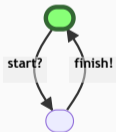
0:Ctr



1:R



2:R



```
// Async. Race (global)
```

acts

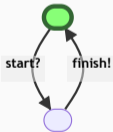
```
start: 1->2, fifo@global;
```

```
finish: 2->1, fifo@global;
```

0:Ctr



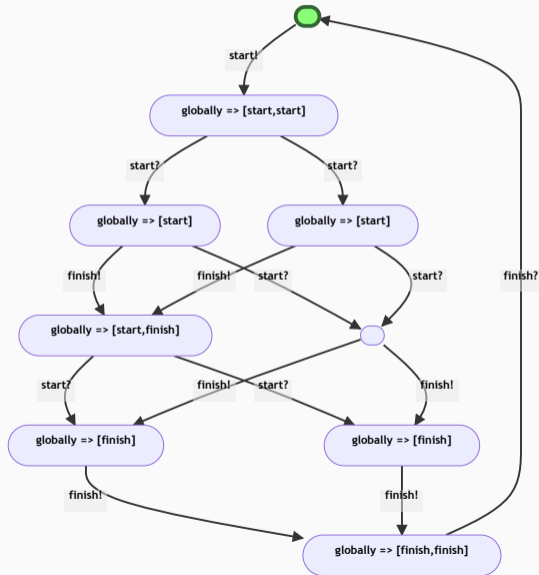
1:R



2:R



1:R can start twice before 2:R



Principle 1: *Asynchronous **sending** messages **should not block**.*

Principle 2: *The **sender or receiver** of an asynchronous action **must not be explicitly mentioned** when **this is disregarded** by the buffer location.*

Principle 3: *The **number** of asynchronous messages sent or received **must be deterministic**.*

Well-formedness

Syntactic checks: violations to the synchronisation types
(e.g., wrong numbers of targets or sources of an action)

Well-behavedness

Semantic checks: from traversing the full state-space
(e.g., deadlock, orphan messages found, or ever-growing buffers)

A-TEAM – Animator of Team Automata with asynchronous communication

Input program

```

1 acts
2   start: 1->2, fifo@snd;
3   finish: 2->1, fifo@snd;
4 proc
5   Ctr = start!.finish?r1,r2.Ctr
6   R = start?c.finish!.R
7 init
8   c:Ctr || r1:R || r2:R
    
```

Race async - both actions send asynchronously, with a buffer for each sender. This is problematic because a runner can start, finish, and start again, consuming a start message that was meant to the other runner (causing a deadlock).

Examples

View pretty data

Well-behaved?

Run semantics

Trace: start!

undo

Enabled transitions:

start?c
start?c

→
c: Ctr
r1: R
r2: R

start!
→
c: finish?r1,r2.Ctr
r1: R
r2: R
c->_ => [start,start]

Build LTS

Local components



Number of states and edges

A-TEAM - Animator of Team Automata with asynchronous communication

Input program

```
1 acts
2   start: 1->2, fifo@snd;
3   finish: 2->1, fifo@snd;
4 proc
5   Ctr = start!.finish?r1,r2.Ctr
6   R = start?c.finish!.R
7 init
8 c:Ctr || r1:R || r2:R
```

Race async - both actions send asynchronously, with a buffer for each sender. This is problematic because a runner can start, finish, and start again, consuming a start message that was meant to the other runner (causing a deadlock).

Examples

coffee-sync coffee-async
coffee-async-leave race-sync
race@rcv race@snd race@global
race@unbounded race@errors
race@bt-error healthcare-sync

View pretty data

Well-behaved?

Run semantics

Trace: start!

undo

Enabled transitions:
start?c
start?c

→ c: Ctr
r1: R
r2: R

start!
→ c: finish?r1,r2.Ctr
r1: R
r2: R
c->_ => [start,start]

Build LTS

Local components



Number of states and edges



Feedback in A-Team (ill-formed)

Input program

```
1 acts start: 1->2, sync;  
2   finish: 2->1, fifo@snd;  
3 proc Ctr = start!.finish?r1.Ctr  
4   R = start?.finish!c.R  
5 init c:Ctr || r1:R || r2:R
```

Error raised by 'Well-behaved?': Trying to send 'finish!c' to {c} but there is no guarantees that the receivers are the correct ones. You should either remove the explicit set of senders, or change the synchronisation type.

[Ctr] Trying to receive 'finish' from {r1} but expected $\#\{r1\} \in \{2\}$.

[R] Sending finish to c, but no destination should be specified.

Well-behaved?



Try A-Team yourself:

<https://fm-dcc.github.io/a-team/>

Feedback in A-Team (ill-behaved)

Input program

```
1 acts
2   default: fifo@rcv, 1->1;
3   coin; coffee;
4
5 proc
6   // User may leave
7   Usr = coin!m.coffee?.(leave+Usr)
8   Mach = coin?.coffee!u.Mach
9   // Add an extra coin
10  Usr2 = coin!m.Usr
11
12 init
13  u:Usr2 || m:Mach
```

Variation of the asynchronous coffee machine with an extra coin and a terminating option, to create orphan messages.

Examples

Hide comm-props

Max buffers' sizes

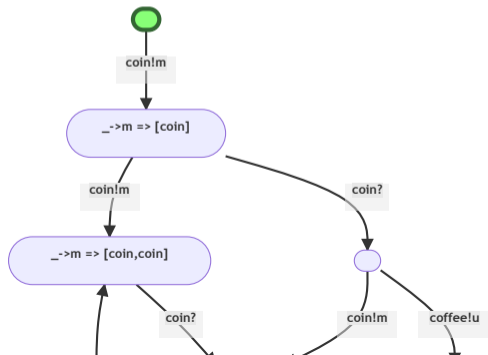
View pretty data

Well-behaved?

Deadlock found: {u: skip, m: Mach, _->u=>[coffee]}
[orphan-messages] Agent u terminated and has incoming messages: {u: skip, m: Mach, _->u=>[coffee]}
[strong-responsiveness] m can receive coin, but the system cannot (yet) send it: {u: skip, m: Mach, _->u=>[coffee]}
[strong-responsiveness] m can receive coin, but the system cannot (yet) send it: {u: coffee?.(leave+Usr)}
[strong-responsiveness] u can receive coffee, but the system cannot (yet) send it: {u: coffee?.(leave+Usr)}

Run semantics

Build LTS



Conclusion and Future Work

ATeams offer a unifying semantic foundation and a prototypical tool A-Team for the modelling and analysis of heterogeneous **synchronous–asynchronous multi-party interactions**

ATeams offer a unifying semantic foundation and a prototypical tool A-Team for the modelling and analysis of heterogeneous **synchronous–asynchronous multi-party interactions**

Future work

- Further analyses to guarantee a richer set of communication properties (e.g., **deadlock detection**) possibly with back-ends relying on tools such as mCRL2
- Encode CFSMs into ATeams (limited to actions whose synchronisation type has buffer type FIFO and location type SND-RCV): deadlock detection **undecidable**
- Suitable restrictions to recover decidability? (e.g., all buffers unordered or **finite**)

Input program

```
1 acts
2   start: 1->2, sync;
3   rest:  1->2, fifo@rcv;
4   finish: 2->1, fifo@rcv;
5 proc
6   Ctr = start!.finish?.Ctr
7       + rest!r1,r2.Ctr
8   R = start?.finish!c.R
9       + rest?.R
10 init
11 c:Ctr || r1:R || r2:R
```

Variation of the race example with an unbounded fifo growing forever.

Examples

Hide comm-props

Max buffers' sizes

View pretty data

Well-behaved?

Run semantics

Build LTS

Local components

Number of states and edges

Maximum buffers' sizes

```
== Traversed 251 edges (max: 7; still more to go) ==
- Buffer _->r1: max size 7
- Buffer _->r2: max size 6
- Buffer _->c: max size 2
== Traversed 503 edges (max: 11; still more to go) ==
- Buffer _->r1: max size 11
- Buffer _->r2: max size 10
- Buffer _->c: max size 2
== Traversed 1000 edges (max: 17; still more to go) ==
- Buffer _->r1: max size 17
- Buffer _->r2: max size 15
- Buffer _->c: max size 2
== Traversed 2000 edges (max: 25; still more to go) ==
```

Well-behaved

Well-behaved (simpler)

More...

Input program

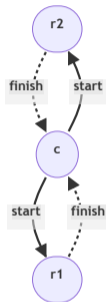
```
1 acts
2   start: 1->2, sync;
3   finish: 2->1, fifo@rcv;
4 proc
5   Ctr = start!r1,r2.finish?.Ctr
6   R = start?.finish!c.R
7 init
8   c:Ctr || r1:R || r2:R
```

Examples

coffee-sync coffee-async coffee-async-leave
race-sync race@rcv race@snd race@global
race@unbounded race@errors race@bt-error
priority-queue healthcare-sync

View pretty data

Architectural view (WiP)



Well-behaved?

Run semantics

Build LTS

Local components

Thanks for your attention!



<https://fm-dcc.github.io/a-team/>



Questions?