

Can we Communicate?

Using Dynamic Logic to Verify Featured Team Automata

Maurice H. ter Beek¹ Guillermina Cledou² Rolf Hennicker³ José Proença⁴

¹ISTI-CNR, Pisa, Italy

²University of Minho, Portugal

³LMU Munich, Germany

⁴Polytechnic Institute of Porto, Portugal

Pre-VaMoS Workshop at ITU, Copenhagen, 24 January 2023

Before we start, and I forget ...

The Formal Methods and Tools lab at ISTI–CNR in Pisa, Italy, offers 1 grant for a 3-year PhD programme at the University of Pisa on “*Formal modelling and analysis of critical software systems*”, financed by the PNRR project “*Railway Transport*” of the Sustainable Mobility Center



Istituto di Scienza e Tecnologie
dell'Informazione “A. Faedo”
Consiglio Nazionale delle Ricerche



The deadline for applications is: **February 1st, 2023, at 1pm**

Requirement for participation: master's degree by February 28th

If you're interested, feel free to contact me for more information!

References

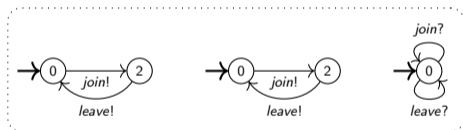
- FM21** M.H. ter Beek, G. Cledou, R. Hennicker & J. Proença, *Featured Team Automata*. In Proceedings of the 24th International Symposium on Formal Methods (FM'21) (M. Huisman, C. Păsăreanu, and N. Zhan, eds.), Lecture Notes in Computer Science 13047, Springer, 2023.
- FM23** M.H. ter Beek, G. Cledou, R. Hennicker & J. Proença, *Can we Communicate? Using Dynamic Logic to Verify Team Automata*. In Proceedings of the 25th International Symposium on Formal Methods (FM'23) (M. Chechik, J.-P. Katoen, and M. Leucker, eds.), Lecture Notes in Computer Science 14000, Springer, 2023.



Background

Team Automata:

- Systems of communicating components: synchronise over shared actions
- Synchronisation types per action: *peer-2-peer*, *broadcast*, ...



Goal: **safe communication** – *no message loss, no indefinite waiting, ...*

ter Beek, *Team Automata*. Ph.D. thesis, Leiden University, 2003

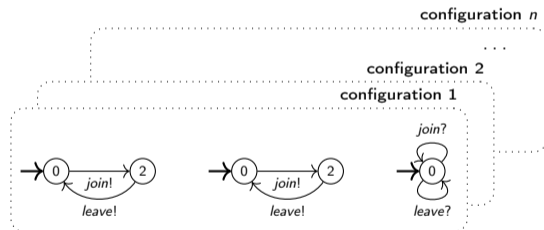
ter Beek, Ellis, Kleijn & Rozenberg, *Synchronizations in team automata for groupware systems*. Comput. Sup. Coop. Work 12, 2003

ter Beek, Hennicker & Kleijn, *Compositionality of Safe Communication in Systems of Team Automata*. ICTAC 2020

Motivation

Many systems today are **highly configurable** (in terms of features):

- Large sets of similar systems that share a lot of behaviour but differ in other

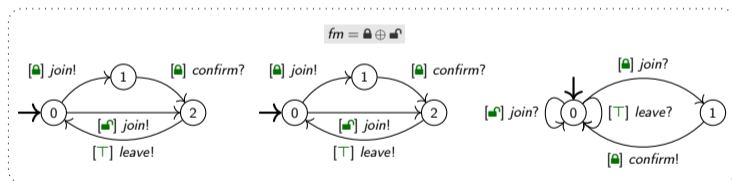


Challenge: system-by-system analysis of safe communication quickly becomes **unfeasible**

Approach

Featured Team Automata:

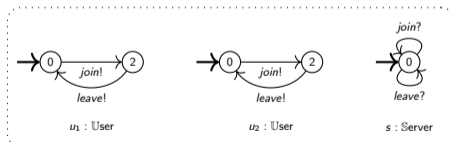
- Families (sets) of Team Automata model as a Software Product Line
- Single model parametrised by **features** (e.g.: lock , unlock), and a **feature model** ($\text{lock} \oplus \text{unlock}$)



Goal: **family-based analysis** of safe communication

Team Automata

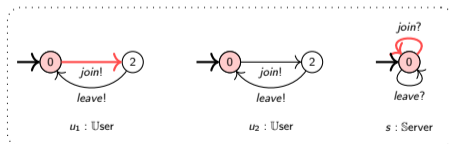
ICTAC'20:



Systems

Team Automata

ICTAC'20:

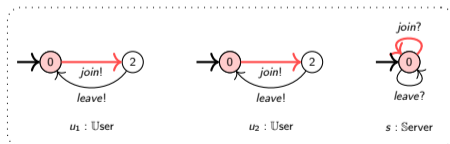


Systems

$$(0, 0, 0) \xrightarrow{\{\{u_1\}, \text{join}, \{s\}\}} (2, 0, 0)$$

Team Automata

ICTAC'20:

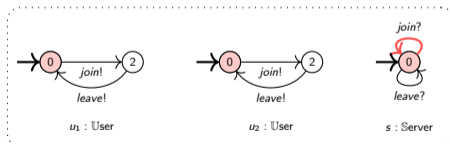


Systems

$$(0, 0, 0) \xrightarrow{\langle \{u_1, u_2\}, join, \{s\} \rangle} (2, 2, 0)$$

Team Automata

ICTAC'20:

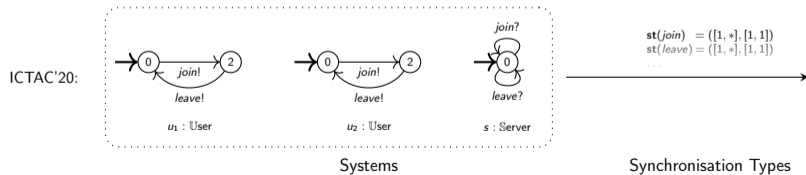


Systems

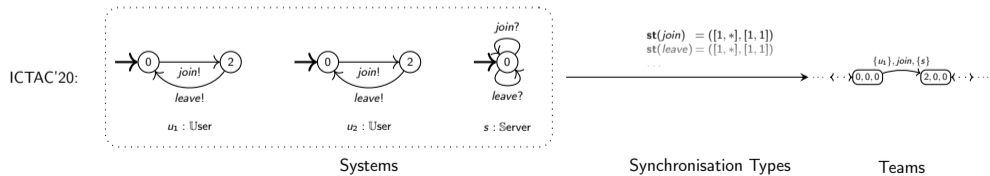
$$(0, 0, 0) \xrightarrow{\{\}, \text{join}, \{s\}} (0, 0, 0)$$

might not be desirable

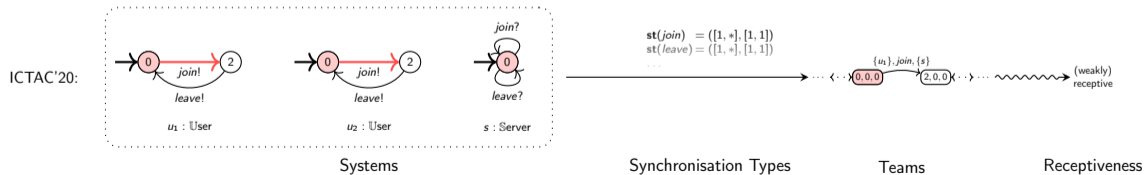
Team Automata



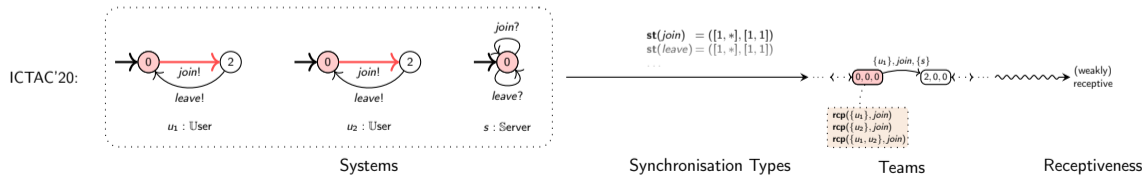
Team Automata



Team Automata

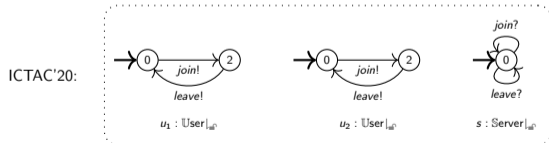
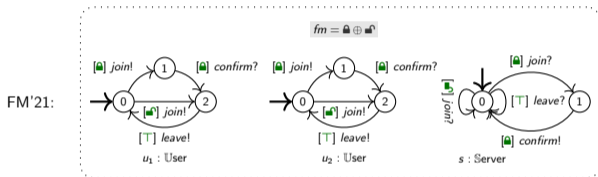


Team Automata



Featured Team Automata

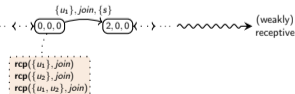
Featured Systems



Systems

$st(\text{join}) = ([1, *], [1, 1])$
 $st(\text{leave}) = ([1, *], [1, 1])$

Synchronisation Types

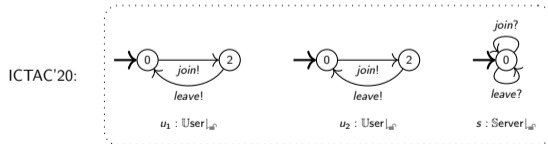
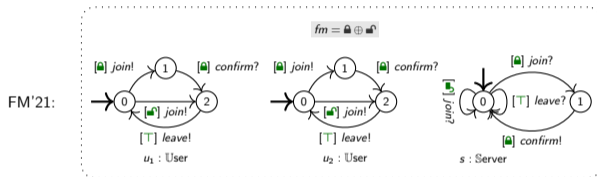


Teams

Receptiveness

Featured Team Automata

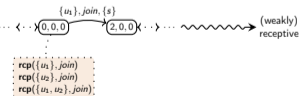
Featured Systems



Systems

$st(join) = ([1, *], [1, 1])$
 $st(leave) = ([1, *], [1, 1])$

Synchronisation Types



Teams

Receptiveness

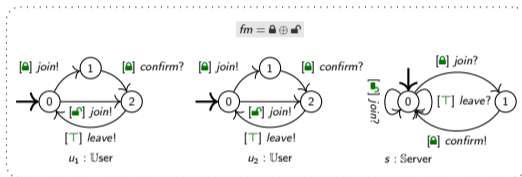
(weakly) receptive

Featured Team Automata

Featured Systems

Featured Synchronisation Types

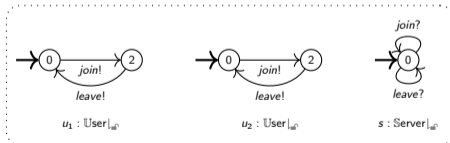
FM'21:



fst($\{lock\}$, join) = ([1, *], [1, 1])
fst($\{lock\}$, leave) = ([1, *], [1, 1])
...



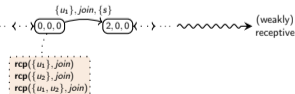
ICTAC'20:



Systems

st(join) = ([1, *], [1, 1])
st(leave) = ([1, *], [1, 1])
...

Synchronisation Types



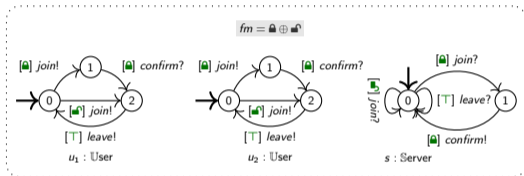
Teams

Receptiveness

Featured Team Automata

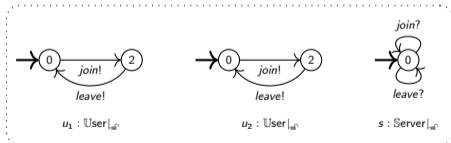
Featured Systems

FM'21:



Systems

ICTAC'20:



Featured Synchronisation Types

$$fst(\{a\}, join) = ([1, *], [1, 1])$$

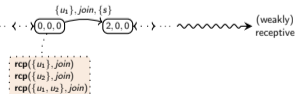
$$fst(\{a\}, leave) = ([1, *], [1, 1])$$

$$st(join) = ([1, *], [1, 1])$$

$$st(leave) = ([1, *], [1, 1])$$

Synchronisation Types

Teams

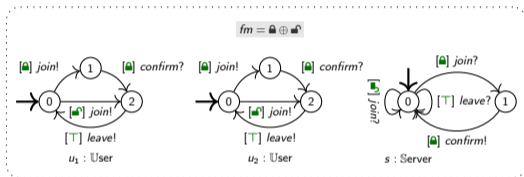


Receptiveness

Featured Team Automata

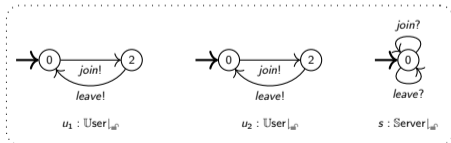
Featured Systems

FM'21:



Systems

ICTAC'20:



Featured Synchronisation Types

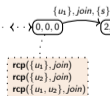
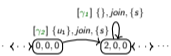
$fst(\{u_1\}, join) = ([1, *], [1, 1])$
 $fst(\{u_1\}, leave) = ([1, *], [1, 1])$

$st(join) = ([1, *], [1, 1])$
 $st(leave) = ([1, *], [1, 1])$

Synchronisation Types

Featured Teams

$fm = u_1 \oplus u_2$



Teams

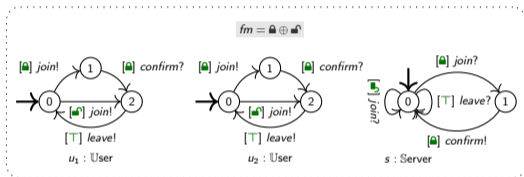
Receptiveness

(weakly) receptive

Featured Team Automata

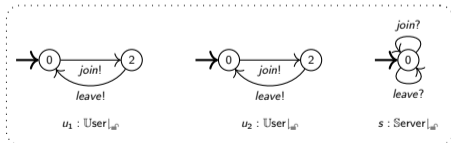
Featured Systems

FM'21:



Systems

ICTAC'20:



Featured Synchronisation Types

fst($\{a\}$, join) = ([1, *], [1, 1])
fst($\{a\}$, leave) = ([1, *], [1, 1])

st(join) = ([1, *], [1, 1])
st(leave) = ([1, *], [1, 1])

Synchronisation Types

Featured Teams

$fm = \{a, b, c\}$

$[a] \{u_1\}.join, \{s\}$

$[a] \{u_2\}.join, \{s\}$

$\{u_1\}.join, \{s\}$

rcp($\{u_1\}.join$)
rcp($\{u_2\}.join$)
rcp($\{u_1, u_2\}.join$)

Teams

Featured Receptiveness

featured (weakly) receptive

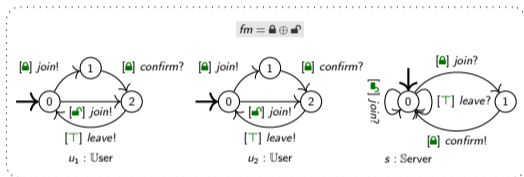
(weakly) receptive

Receptiveness

Featured Team Automata

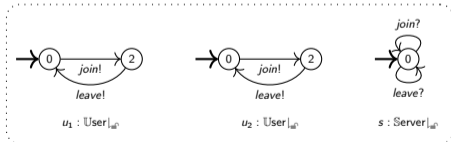
Featured Systems

FM'21:



Systems

ICTAC'20:



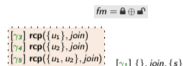
Featured Synchronisation Types

fst($\{u_1\}, join$) = $([1, *], [1, 1])$
fst($\{u_1\}, leave$) = $([1, *], [1, 1])$

st(join) = $([1, *], [1, 1])$
st(leave) = $([1, *], [1, 1])$

Synchronisation Types

Featured Teams



Teams

Featured Receptiveness

featured (weakly) receptive

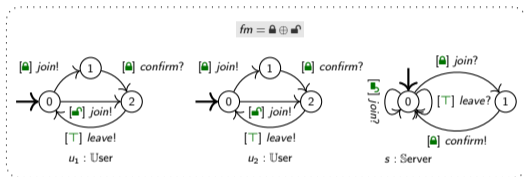
(weakly) receptive

Receptiveness

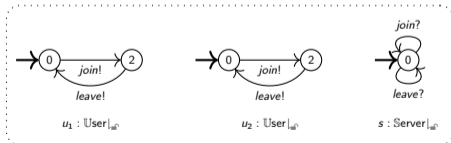
Featured Team Automata

Featured Systems

FM'21:



ICTAC'20:



Systems

Featured Synchronisation Types

$$\text{fst}(\{a\}, \text{join}) = ([1, *], [1, 1])$$

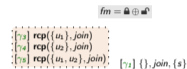
$$\text{fst}(\{a\}, \text{leave}) = ([1, *], [1, 1])$$

$$\text{st}(\text{join}) = ([1, *], [1, 1])$$

$$\text{st}(\text{leave}) = ([1, *], [1, 1])$$

Synchronisation Types

Featured Teams



Teams

Featured Receptiveness

featured (weakly) receptive

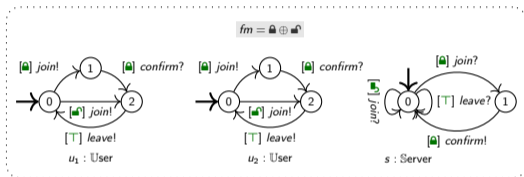
(weakly) receptive

Receptiveness

Featured Team Automata

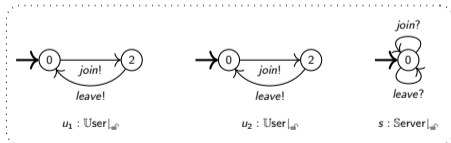
Featured Systems

FM'21:



Systems

ICTAC'20:



Featured Synchronisation Types

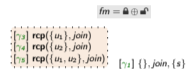
$fst(\{a\}, join) = ([1, *], [1, 1])$
 $fst(\{a\}, leave) = ([1, *], [1, 1])$

$st(join) = ([1, *], [1, 1])$
 $st(leave) = ([1, *], [1, 1])$

$\rightarrow = \rightarrow$
 (diagram commutes)

Synchronisation Types

Featured Teams



Teams

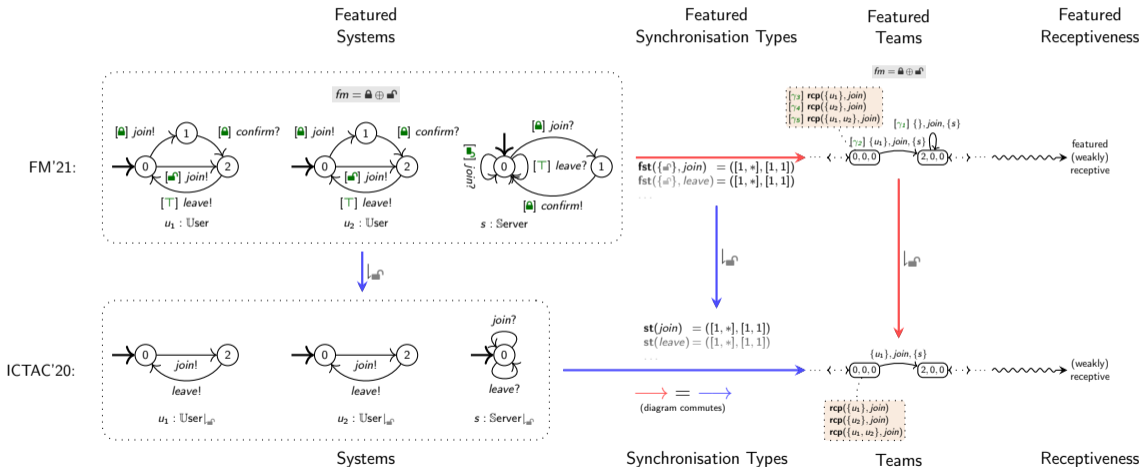
Featured Receptiveness

featured (weakly) receptive

(weakly) receptive

Receptiveness

Featured Team Automata

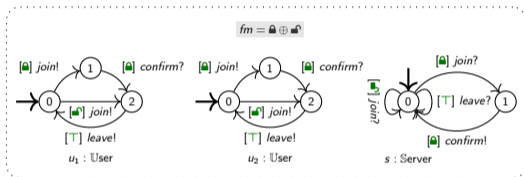


Online prototype: <http://arcatools.org/feta>

Featured Team Automata

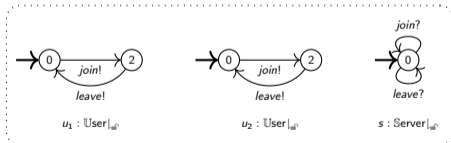
Featured Systems

FM'21:



Systems

ICTAC'20:



Featured Synchronisation Types

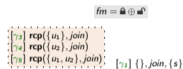
fst($\{a\}, join$) = $([1, *], [1, 1])$
 fst($\{a\}, leave$) = $([1, *], [1, 1])$

st(join) = $([1, *], [1, 1])$
 st(leave) = $([1, *], [1, 1])$

$\rightarrow = \rightarrow$
 (diagram commutes)

Synchronisation Types

Featured Teams



Teams

Featured Receptiveness

featured (weakly) receptive

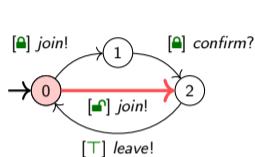
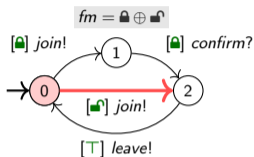
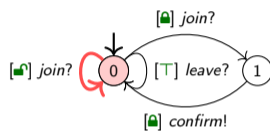
(weakly) receptive

Main Theorem

Receptiveness

Online prototype: <http://arcatools.org/feta>

Featured Team Automata Transitions

 $u_1 : \text{User}$  $u_2 : \text{User}$  $s : \text{Server}$

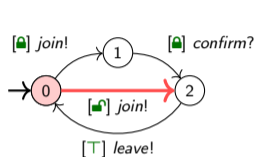
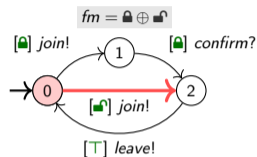
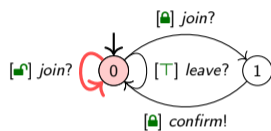
Transitions are **constrained** with feature expressions by:

- **local feature expressions**: characterise the products with all local transitions present
- **fst**: characterise the products that satisfy the corresponding synchronisation type

$$\text{fst}(\{\text{lock}\}, \text{join}) = ([1, 1], [1, 1]) \quad \text{fst}(\{\text{unlock}\}, \text{join}) = ([1, *], [1, 1])$$

$$(0, 0, 0) \xrightarrow{[\quad](\{u_1, u_2\}, \text{join}, \{s\})} \text{fst}[S] (2, 2, 0)$$

Featured Team Automata Transitions

 $u_1 : \text{User}$  $u_2 : \text{User}$  $s : \text{Server}$

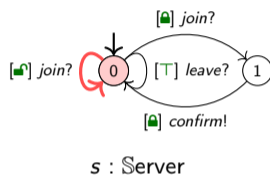
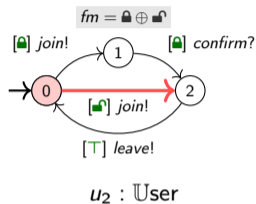
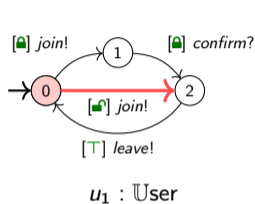
Transitions are **constrained** with feature expressions by:

- **local feature expressions**: characterise the products with all local transitions present
- **fst**: characterise the products that satisfy the corresponding synchronisation type

$$\text{fst}(\{\text{lock}\}, \text{join}) = ([1, 1], [1, 1]) \quad \text{fst}(\{\text{unlock}\}, \text{join}) = ([1, *], [1, 1])$$

$$(0, 0, 0) \xrightarrow{[\text{lock} \wedge \text{lock} \wedge \text{lock}]} [(\{u_1, u_2\}, \text{join}, \{s\})]_{\text{fst}[S]} (2, 2, 0)$$

Featured Team Automata Transitions



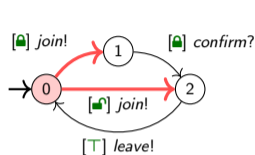
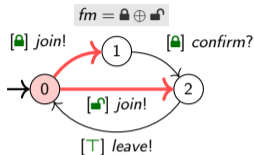
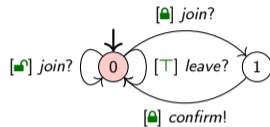
Transitions are **constrained** with feature expressions by:

- **local feature expressions**: characterise the products with all local transitions present
- **fst**: characterise the products that satisfy the corresponding synchronisation type

$$\text{fst}(\{\text{lock}\}, \text{join}) = ([1, 1], [1, 1]) \quad \text{fst}(\{\text{unlock}\}, \text{join}) = ([1, *], [1, 1])$$

$$(0, 0, 0) \xrightarrow{[\text{lock} \wedge \text{lock} \wedge \text{lock} \wedge \text{lock} \wedge \text{lock} \wedge \neg \text{lock}]}_{\text{fst}[S]} (\{u_1, u_2\}, \text{join}, \{s\}) (2, 2, 0)$$

Featured Receptiveness Requirements

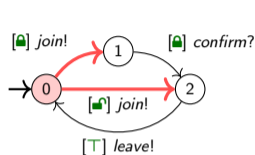
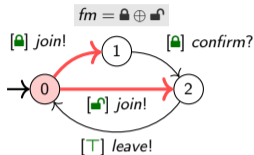
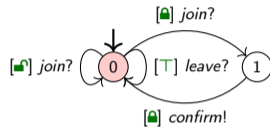
 $u_1 : \text{User}$  $u_2 : \text{User}$  $s : \text{Server}$

$$\text{fst}(\{\text{lock}\}, \text{join}) = ([1, 1], [1, 1]) \quad \text{fst}(\{\text{lock}\}, \text{join}) = ([1, *], [1, 1])$$

At state $(0, 0, 0)$:

$$[\quad] \text{rcp}(\{u_1\}, \text{join}) \wedge [\quad] \text{rcp}(\{u_2\}, \text{join}) \wedge [\quad] \text{rcp}(\{u_1, u_2\}, \text{join})$$

Featured Receptiveness Requirements

 $u_1 : \text{User}$  $u_2 : \text{User}$  $s : \text{Server}$

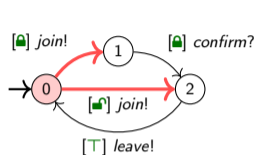
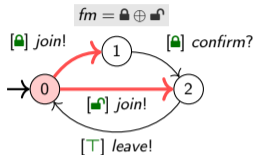
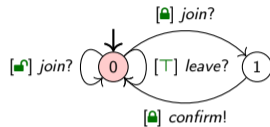
$$\text{fst}(\{\text{lock}\}, \text{join}) = ([1, 1], [1, 1]) \quad \text{fst}(\{\text{lock}\}, \text{join}) = ([1, *], [1, 1])$$

At state $(0, 0, 0)$:

$$[\quad] \text{rcp}(\{u_1\}, \text{join}) \wedge [\quad] \text{rcp}(\{u_2\}, \text{join}) \wedge [\quad] \text{rcp}(\{u_1, u_2\}, \text{join})$$

Featured receptiveness requirements are **constrained** with feature expression by:

Featured Receptiveness Requirements

 $u_1 : \text{User}$  $u_2 : \text{User}$  $s : \text{Server}$

$$\text{fst}(\{\text{lock}\}, \text{join}) = ([1, 1], [1, 1]) \quad \text{fst}(\{\text{lock}\}, \text{join}) = ([1, *], [1, 1])$$

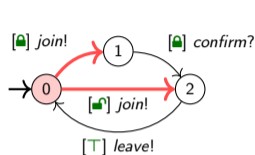
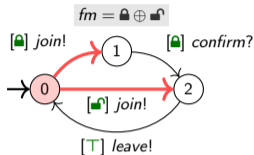
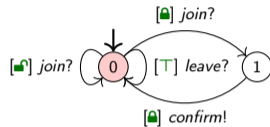
At state $(0, 0, 0)$:

$$[\quad] \text{rcp}(\{u_1\}, \text{join}) \wedge [\quad] \text{rcp}(\{u_2\}, \text{join}) \wedge [\quad] \text{rcp}(\{u_1, u_2\}, \text{join})$$

Featured receptiveness requirements are **constrained** with feature expression by:

- **local feature expressions**: characterise products with enabled local transitions

Featured Receptiveness Requirements

 $u_1 : \text{User}$  $u_2 : \text{User}$  $s : \text{Server}$

$$\text{fst}(\{\text{lock}\}, \text{join}) = ([1, 1], [1, 1]) \quad \text{fst}(\{\text{lock}\}, \text{join}) = ([1, *], [1, 1])$$

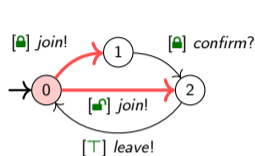
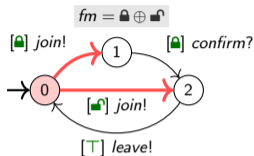
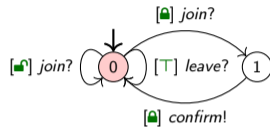
At state $(0, 0, 0)$:

$$[\text{lock} \vee \text{lock}] \text{rcp}(\{u_1\}, \text{join}) \wedge [] \text{rcp}(\{u_2\}, \text{join}) \wedge [] \text{rcp}(\{u_1, u_2\}, \text{join})$$

Featured receptiveness requirements are **constrained** with feature expression by:

- **local feature expressions**: characterise products with enabled local transitions

Featured Receptiveness Requirements

 $u_1 : \text{User}$  $u_2 : \text{User}$  $s : \text{Server}$

$$\text{fst}(\{\text{lock}\}, \text{join}) = ([1, 1], [1, 1]) \quad \text{fst}(\{\text{lock} \oplus \text{lock}\}, \text{join}) = ([1, *], [1, 1])$$

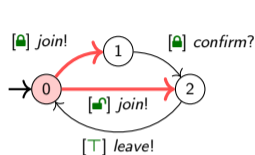
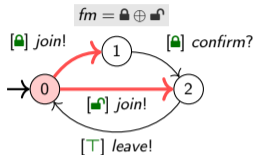
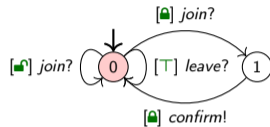
At state $(0, 0, 0)$:

$$[\text{lock} \vee \text{lock}] \text{rcp}(\{u_1\}, \text{join}) \wedge [] \text{rcp}(\{u_2\}, \text{join}) \wedge [] \text{rcp}(\{u_1, u_2\}, \text{join})$$

Featured receptiveness requirements are **constrained** with feature expression by:

- **local feature expressions**: characterise products with enabled local transitions
- **fst**: characterise products with the correct number of senders

Featured Receptiveness Requirements

 $u_1 : \text{User}$  $u_2 : \text{User}$  $s : \text{Server}$

$$\text{fst}(\{\text{lock}\}, \text{join}) = ([1, 1], [1, 1]) \quad \text{fst}(\{\text{lock} \oplus \text{lock}\}, \text{join}) = ([1, *], [1, 1])$$

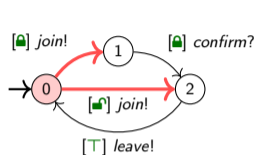
At state $(0, 0, 0)$:

$$[\text{lock} \vee \text{lock} \wedge fm] \text{rcp}(\{u_1\}, \text{join}) \wedge [] \text{rcp}(\{u_2\}, \text{join}) \wedge [] \text{rcp}(\{u_1, u_2\}, \text{join})$$

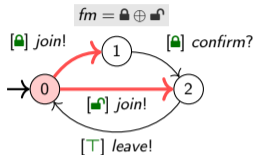
Featured receptiveness requirements are **constrained** with feature expression by:

- **local feature expressions**: characterise products with enabled local transitions
- **fst**: characterise products with the correct number of senders

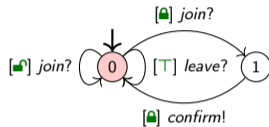
Featured Receptiveness Requirements



$u_1 : \text{User}$



$u_2 : \text{User}$



$s : \text{Server}$

$$\text{fst}(\{\text{lock}\}, \text{join}) = ([1, 1], [1, 1]) \quad \text{fst}(\{\text{server}\}, \text{join}) = ([1, *], [1, 1])$$

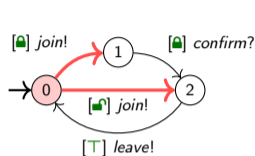
At state $(0, 0, 0)$:

$$[\text{lock} \vee \text{server} \wedge fm] \text{rcp}(\{u_1\}, \text{join}) \wedge [] \text{rcp}(\{u_2\}, \text{join}) \wedge [] \text{rcp}(\{u_1, u_2\}, \text{join})$$

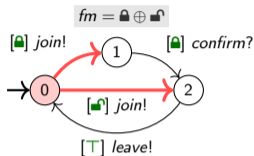
Featured receptiveness requirements are **constrained** with feature expression by:

- **local feature expressions**: characterise products with enabled local transitions
- **fst**: characterise products with the correct number of senders
- **reachable states**: characterise products where the state is reachable

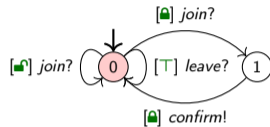
Featured Receptiveness Requirements



$u_1 : \text{User}$



$u_2 : \text{User}$



$s : \text{Server}$

$$\text{fst}(\{u_1\}, \text{join}) = ([1, 1], [1, 1]) \quad \text{fst}(\{u_2\}, \text{join}) = ([1, *], [1, 1])$$

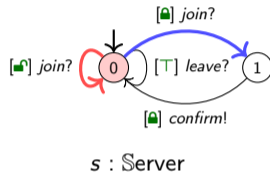
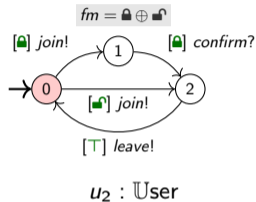
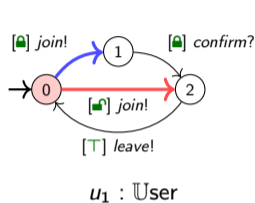
At state $(0, 0, 0)$:

$$[u_1 \vee u_2 \wedge fm] \text{rcp}(\{u_1\}, \text{join}) \wedge [fm] \text{rcp}(\{u_2\}, \text{join}) \wedge [u_1 \vee u_2 \wedge u_1 \wedge \neg u_2 \wedge fm] \text{rcp}(\{u_1, u_2\}, \text{join})$$

Featured receptiveness requirements are **constrained** with feature expression by:

- **local feature expressions**: characterise products with enabled local transitions
- **fst**: characterise products with the correct number of senders
- **reachable states**: characterise products where the state is reachable

Compliance with requirements



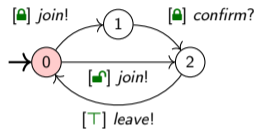
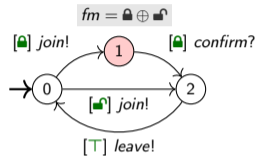
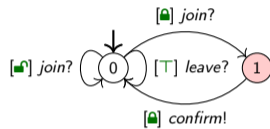
At state $(0, 0, 0)$:

$$\underline{[fm] \text{rcp}(\{u_1\}, \text{join})} \wedge [fm] \text{rcp}(\{u_2\}, \text{join}) \wedge [\text{lock} \wedge \neg \text{lock}] \text{rcp}(\{u_1, u_2\}, \text{join})$$

$$\{\text{lock}\} : (0, 0, 0) \xrightarrow{[\text{lock} \wedge fm] (\{u_1\}, \text{join}, \{s\})} \text{fst}[S](1, 0, 1)$$

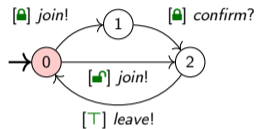
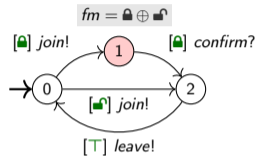
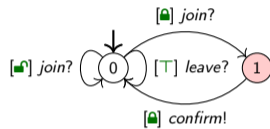
$$\{\text{lock}\} : (0, 0, 0) \xrightarrow{[\text{lock} \wedge fm] (\{u_1\}, \text{join}, \{s\})} \text{fst}[S](2, 0, 0)$$

Compliance with requirements

 $u_1 : \text{User}$  $u_2 : \text{User}$  $s : \text{Server}$ At state $(0, 1, 1)$:

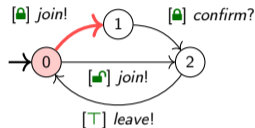
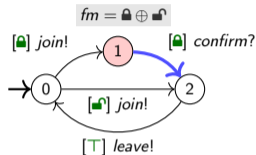
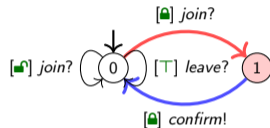
$$[\text{lock} \wedge \neg \text{unlock}] \text{rcp}(\{u_1\}, \text{join}) \wedge \dots$$

Compliance with requirements

 $u_1 : \text{User}$  $u_2 : \text{User}$  $s : \text{Server}$ At state $(0, 1, 1)$:

$$\neg [\text{lock} \wedge \neg \text{lock}] \text{rcp}(\{u_1\}, \text{join}) \wedge \dots$$

Weak compliance with requirements

 $u_1 : \text{User}$  $u_2 : \text{User}$  $s : \text{Server}$ At state $(0, 1, 1)$:

$$[\text{lock} \wedge \neg \text{lock}] \text{rcp}(\{u_1\}, \text{join}) \wedge \dots$$

$$\{\text{lock}\} : (0, 1, 1) \xrightarrow{[\text{lock} \wedge \text{fm}](\{s\}, \text{confirm}, \{u_2\})} \text{fst}[S](0, 2, 0) \xrightarrow{[\text{lock} \wedge \text{fm}](\{u_1\}, \text{join}, \{s\})} \text{fst}[S](1, 2, 1)$$

Online prototype

- Specify
- Generate*
- Visualise
- Statistics

*SAT solver to solve *fm*

Online Tools for Featured Extensions: X

arcatools.org/assets/feta.html#onlinefeta

FETA Development Back to Arcatools

FETA Specification

```

1 FCA user (confirm)(join,leave) = {
2   start 0
3   0 → 1 by join if s
4   1 → 2 by confirm if s
5   0 → 2 by join if o
6   2 → 0 by leave
7 }
8
9 FCA server (join,leave)(confirm) = {
10  start 0
11  0 → 1 by join if s
12  1 → 0 by confirm if s
13  0 → 0 by join if o
14  0 → 0 by leave
15 }
16
17 FS = (u1 → user, u2 → user, s → server)
18
19 FH = s xor o
20
21 FST = {
22   default = one to one // or 1..1 to 1..1
23   (o):join,leave = many to one // or 1..* to 1..1
24 }

```

FETA Examples

Auth Chat

FETA Information

Products: 2
 • {s}, {o}

Featured System:
 • # transitions: 142
 • # states: 18

FETA:
 • # transitions: 18
 • # states: 8

FSTs:
 • fst(o)|(confirm) = 1->1
 fst(s)|(confirm) = 1->1
 fst(s)|(join) = 1->1
 fst(o)|(join) = 1..*->1
 fst(o)|(leave) = 1->1
 fst(o)|(leave) = 1..*->1

FETA

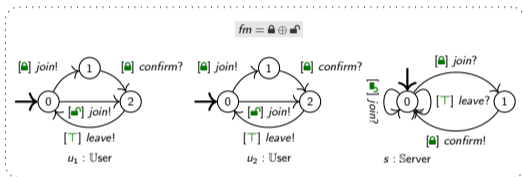
FCA

Copyright 2017-2021 - ARCA, U. Lameiro, pt

Wrapping up

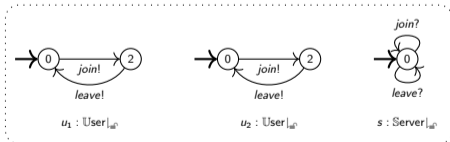
Featured Systems

FM'21:



Systems

ICTAC'20:



Featured Synchronisation Types

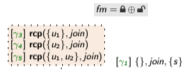
fst($\{a\}, join$) = $([1, *], [1, 1])$
 fst($\{a\}, leave$) = $([1, *], [1, 1])$

st(join) = $([1, *], [1, 1])$
 st(leave) = $([1, *], [1, 1])$

$\rightarrow = \rightarrow$
 (diagram commutes)

Synchronisation Types

Featured Teams



$\{u_1\}, join, \{s\}$

rcp($\{u_1\}, join$)
 rcp($\{u_2\}, join$)
 rcp($\{u_1, u_2\}, join$)

Teams

Featured Receptiveness

featured (weakly) receptive

(weakly) receptive

Main Theorem

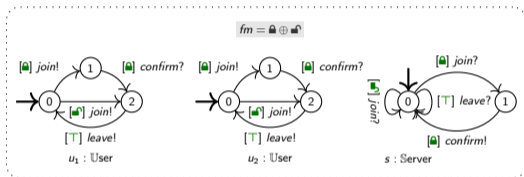
Receptiveness

Online prototype: <http://arcatoools.org/feta>

Future and ongoing work

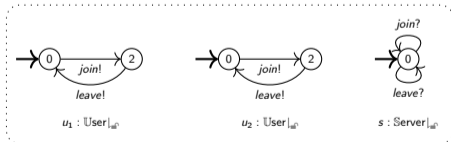
Featured Systems

FM'21:



Systems

ICTAC'20:



Featured Synchronisation Types

Compositionality

$$\text{fst}(\{a\}, \text{join}) = ([1, *], [1, 1])$$

$$\text{fst}(\{a\}, \text{leave}) = ([1, *], [1, 1])$$

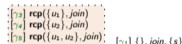
$$\text{st}(\text{join}) = ([1, *], [1, 1])$$

$$\text{st}(\text{leave}) = ([1, *], [1, 1])$$

(diagram commutes)

Featured Teams

fm = {a, T}



[T2] {u1}, join, {s}

Featured Receptiveness

Smarter: which configurations derived compliant teams?

Featured Responsiveness

Main Theorem

featured (weakly) receptive

(weakly) receptive

Synchronisation Types

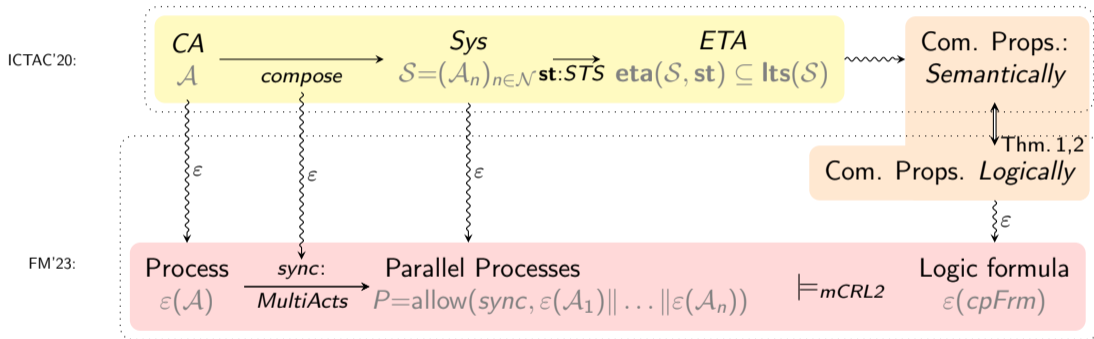
Teams

Receptiveness

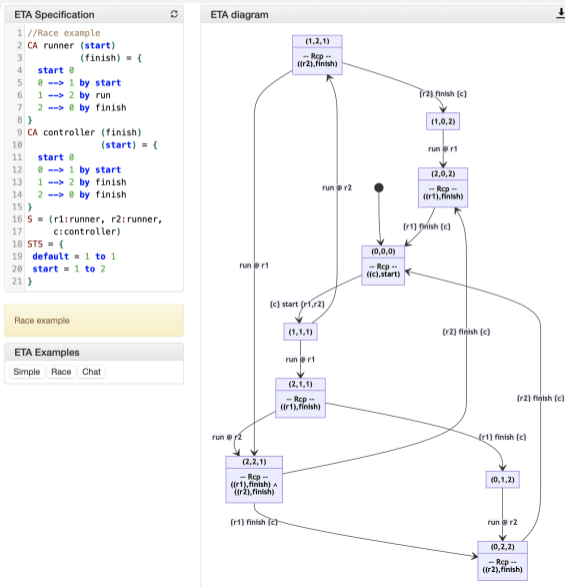
Online prototype: <http://arcatools.org/feta>

Extensions (e.g. FM'23: generate requirements as PDL formulae and check compliance with mCRL2)

Using Dynamic Logic to Verify Team Automata



Online prototype: <https://github.com/arcalab/team-a>



Online prototype: <https://github.com/arcalab/team-a>

```

ETA Specification
1 //Race example
2 CA runner (start)
3   (finish) = {
4     start 0
5     0 --> 1 by start
6     1 --> 2 by run
7     2 --> 0 by finish
8   }
9 CA controller (finish)
10  (start) = {
11    start 0
12    0 --> 1 by start
13    1 --> 2 by finish
14    2 --> 0 by finish
15  }
16 S = (r1:runner, r2:runner,
17      c:controller)
18 STS = {
19   default = 1 to 1
20   start = 1 to 2
21 }

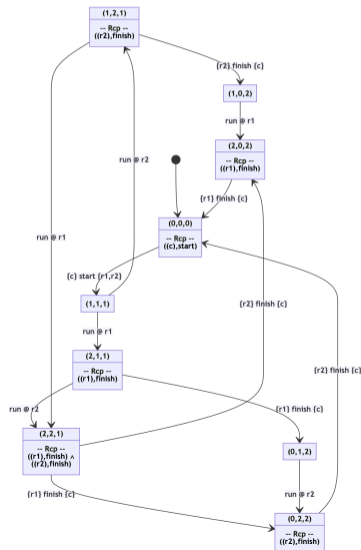
```

Race example

ETA Examples

Simple Race Chat

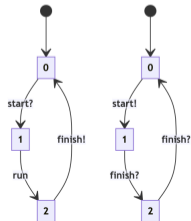
ETA diagram



CA

runner

controller



Online prototype: <https://github.com/arcalab/team-a>

Communication Properties' Characterisation in mCRL2

Receptiveness:

```
[ (r1_finish|c_finish + r2_run + c_start|r1_start|r2_start + r2_finish|c_finish + r1_run)* ](
  ((<c_start> true) => (<c_start|r1_start|r2_start> true)) &&
  ((<r1_finish> true) => (<r1_finish|c_finish> true)) &&
  ((<r2_finish> true) => (<r2_finish|c_finish> true))
)
```

Responsiveness:

```
[ (r1_finish|c_finish + r2_run + c_start|r1_start|r2_start + r2_finish|c_finish + r1_run)* ](
  (<c_finish +
   r1_start|r2_start> true)
  =>
  (<r1_finish|c_finish +
   c_start|r1_start|r2_start +
   r2_finish|c_finish> true)
)
```

Weak Receptiveness:

```
[ (r1_finish|c_finish + r2_run + c_start|r1_start|r2_start + r2_finish|c_finish + r1_run)* ](
  ((<r1_finish> true) => (<(r2_run+r2_finish|c_finish)* . r1_finish|c_finish> true)) &&
  ((<r2_finish> true) => (<(r1_finish|c_finish+r1_run)* . r2_finish|c_finish> true)) &&
  ((<c_start> true) => (<(r2_run+r1_run)* . c_start|r1_start|r2_start> true))
)
```

Weak Responsiveness:

```
[ (r1_finish|c_finish + r2_run + c_start|r1_start|r2_start + r2_finish|c_finish + r1_run)* ](
  (<c_finish +
   r1_start|r2_start> true)
  =>
  (<(r2_run+r1_run)* . r1_finish|c_finish +
   c_start|r1_start|r2_start +
   (r2_run+r1_run)* . r2_finish|c_finish> true)
)
```

View mCRL2 evidence

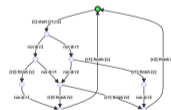
Receptiveness: true



Responsiveness: false



Weak Receptiveness: true



Weak Responsiveness: true



Online prototype: <https://github.com/arcalab/team-a>

mCRL2 full system:

```

act
  r1_start,c_finish,r2_run,c_start,r2_finish,r2_start,r1_finish,r1_run;
proc
  r1(s:Int) =
    (s == 2) -> ( r1_finish.r1(0) ) +
    (s == 1) -> ( r1_run.r1(2) ) +
    (s == 0) -> ( r1_start.r1(1) );
  r2(s:Int) =
    (s == 2) -> ( r2_finish.r2(0) ) +
    (s == 1) -> ( r2_run.r2(2) ) +
    (s == 0) -> ( r2_start.r2(1) );
  c(s:Int) =
    (s == 2) -> ( c_finish.c(0) ) +
    (s == 1) -> ( c_finish.c(2) ) +
    (s == 0) -> ( c_start.c(1) );
init
  allow({
    r1_start,
    c_finish,
    c_start|r2_start,
    c_start,
    c_start|r2_start|r1_start,
    r2_finish|r1_finish,
    c_finish|r2_finish,
    r2_finish,
    r2_start,
    r1_finish,
    c_start|r1_start,
    r1_run,
    c_finish|r2_finish|r1_finish,
    r2_run,
    r2_start|r1_start,
    c_finish|r1_finish},
    r1(0) || r2(0) || c(0));

```

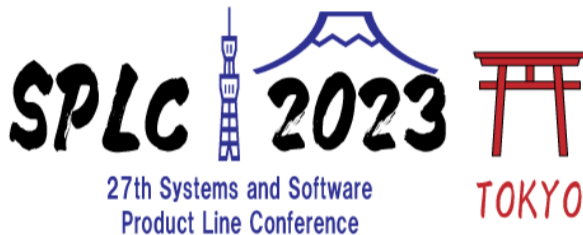
Online prototype: <https://github.com/arcalab/team-a>

System diagram



Thanks for your attention! Questions?

I would be glad to meet some of you at **SPLC 2023** in Tokyo, Japan



Research and Industry track papers deadline: April 13

More tracks (challenges, demonstrations and tools, journal first, doctoral symposium, hall of fame) plus workshops and tutorials

General chairs: Paolo Arcaini, National Institute of Informatics, Japan
and Maurice ter Beek, ISTI-CNR, Pisa, Italy