Formal Methods for Communication Protocols

Harnessing distributed software design with behavioural contracts

Emilio Tuosto @ GSSI

– Lecture 4 –

3 - 12 March, 2026 - Novi Sad

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA | DESK Developing Shared Knowledge

– Programming Actors –

Erlang's actor model

Erlang's actor model

- Processes execute function (and run on a pre-emptive VM, not OS)

Erlang's actor model

- Processes execute function (and run on a pre-emptive VM, not OS)
- Processes are veeeeery light!

Erlang's actor model

- Processes execute function (and run on a pre-emptive VM, not OS)
- Processes are veeeeery light!
- Context commutation is veeeeery fast!

Erlang's actor model

- Processes execute function (and run on a pre-emptive VM, not OS)
- Processes are veeeeery light!
- Context commutation is veeeeery fast!
- Hence VM can handle maaaaany processes!

Erlang's actor model

- Processes execute function (and run on a pre-emptive VM, not OS)
- Processes are veeeeery light!
- Context commutation is veeeeery fast!
- Hence VM can handle maaaaany processes!
- Program code is shared: no assignment!

Erlang's basics

Processes

- have unique IDs: e.g., Pid = self()
- can be spawned: e.g., Pid = spawn(module, function, arguments)
- Pids to send messages:

  Output: $Pid\,!\{1,2,3\}$

  Input:

```
receive
  {X} when X = 0 -> X+X;
  {X,Y} when X = Y -> Y;
  {X,Y,Z} when X < Z andalso X = Y * Z -> ...
end
```

  clauses process in order, the first enable is executed. If none enable, then wait.

An example

```
-module(TempConverter).
-export([convert/1]).
convert(Helper) ->
  receive
    {Pid, cs, C} ->
      if
        overloaded() -> Helper ! {Pid, cs, C} ;
        true ->  Pid ! {self(), ft, (1.8 * C) + 32};
      end,
      convert()
    ;
    {Pid, ft, F} ->
      if
        overloaded() -> Helper ! {Pid, ft, F} ;
        true -> Pid ! {self(), cs, (F - 32) / 1.8} ;
      end,
      convert();
      stop -> true
    ;
      _ -> convert()
  end.
```

Erlang programming

Write a client and the helper function.

"The reflexive chemistry" [1, § 3]

$$
\begin{array}{rcl}
P & \triangleq & x\langle \tilde{v} \rangle \\
& \mid & \textbf{def } D \textbf{ in } P \\
& \mid & P \mid P
\end{array}
\qquad
\begin{array}{rcl}
J & \triangleq & x\langle \tilde{v} \rangle \\
& \mid & J \mid J
\end{array}
\qquad
\begin{array}{rcl}
D & \triangleq & J \triangleright P \\
& \mid & D \wedge D
\end{array}
$$

$\mathcal{R}$eactions $\vdash$ $\mathcal{M}$olecules

$$
\begin{array}{rcl}
\mathcal{R} \vdash P \mid Q & \rightleftharpoons & \mathcal{R} \vdash P, Q \\
D \wedge E \vdash P & \rightleftharpoons & D, E \vdash P \\
D \vdash \textbf{def } E \textbf{ in } P & \rightleftharpoons & D, E\sigma_E \vdash P\sigma_E \\
J \triangleright P \vdash J\sigma_J & \rightarrow & J \triangleright P \vdash P\sigma_J
\end{array}
$$

- σ_E renames vars defined in E with fresh names
- σ_J substitutes the receive vars in J with the values in the corresponding molecules

An "simple" example [1, § 3]

Let's program a memory cells to store some values:

$$
\begin{array}{l}
\textbf{def}\ \mathsf{mkcell}\langle v_0, k\rangle\triangleright \\
\quad \textbf{def} \\
\qquad s\langle v\rangle \mid \mathsf{get}\langle k\rangle \quad \triangleright \quad s\langle v\rangle \mid k\langle v\rangle \\
\qquad s\langle v\rangle \mid \mathsf{set}\langle u, k\rangle \quad \triangleright \quad s\langle u\rangle \mid k\langle\rangle \\
\quad \textbf{in} \\
\qquad s\langle v_0\rangle \mid k\langle \mathsf{get}, \mathsf{set}\rangle \\
\textbf{in} \\
\quad \mathsf{mkcell}\langle 3, c\rangle
\end{array}
$$

An "simple" example [1, § 3]

Let's program a memory cells to store some values:

$$
\begin{array}{l}
\textbf{def}\ \mathrm{mkcell}\langle v_0, k\rangle\triangleright \\
\quad \textbf{def} \\
\qquad s\langle v\rangle \mid \mathrm{get}\langle k\rangle \quad \triangleright \quad s\langle v\rangle \mid k\langle v\rangle \\
\qquad s\langle v\rangle \mid \mathrm{set}\langle u, k\rangle \quad \triangleright \quad s\langle u\rangle \mid k\langle\rangle \\
\quad \textbf{in} \\
\qquad s\langle 3\rangle \mid c\langle \mathrm{get}, \mathrm{set}\rangle \\
\textbf{in} \\
\quad \mathrm{mkcell}\langle 3, c\rangle
\end{array}
$$

An "simple" example [1, § 3]

Let's program a memory cells to store some values:

def mkcell$\langle v_0, k\rangle\triangleright$
  def
    $s\langle v\rangle \mid \text{get}\langle k\rangle \quad \triangleright \quad s\langle v\rangle \mid k\langle v\rangle$
    $s\langle v\rangle \mid \text{set}\langle u, k\rangle \quad \triangleright \quad s\langle u\rangle \mid k\langle\rangle$
  in
    $s\langle v_0\rangle \mid k\langle\text{get}, \text{set}\rangle$
in
  mkcell$\langle 3, c\rangle$

Exercise

Extend the program above so that after the invocation to mkcell the cell is set to 0.

“I have this terrible feeling of déjà vu...”

Doesn’t

$$\textbf{def}\ x_1\langle \tilde{v}_1 \rangle \mid x_2\langle \tilde{v} \rangle_2 \mid \cdots\ \textbf{in}\ P$$

remind you something?

"I have this terrible feeling of déjà vu..."

Doesn't

$$\textbf{def}\ x_1\langle \tilde{v}_1 \rangle \mid x_2\langle \tilde{v} \rangle_2 \mid \cdots\ \textbf{in}\ P$$

remind you something? e.g.,

$$\textbf{let}\ x_1 = E_1\ \textbf{in let}\ x_2 = E_2 \cdots\ \textbf{in}\ E$$

$\lambda \to$ Join?

Recall: $M \triangleq x \mid \lambda x.M \mid MM$

Call-by-name

Leftmost-order reduction strategy & no reduction under λ:

$$
\begin{array}{rcl}
[\![x]\!]_v & \triangleq & v\langle x\rangle \\
[\![\lambda x.M]\!]_v & \triangleq & \textbf{def } k\langle x, m\rangle \triangleright [\![M]\!]_m \textbf{ in } v\langle k\rangle \\
[\![MN]\!]_v & \triangleq & \textbf{def } x\langle n\rangle \triangleright [\![N]\!]_n \textbf{ in} \\
 & & \quad \textbf{def } m\langle k\rangle \triangleright k\langle x, v\rangle \textbf{ in } [\![M]\!]_m
\end{array}
$$

Intuition: a λ-expression is a join process that sends the result of its computation

$\lambda \to$ Join?

Recall: $M \triangleq x \mid \lambda x.M \mid MM$

Call-by-name

Leftmost-order reduction strategy & no reduction under λ:

$$
\begin{array}{rcl}
[\![x]\!]_v & \triangleq & v\langle x\rangle \\
[\![\lambda x.M]\!]_v & \triangleq & \textbf{def } k\langle x, m\rangle \triangleright [\![M]\!]_m \textbf{ in } v\langle k\rangle \\
[\![MN]\!]_v & \triangleq & \textbf{def } x\langle n\rangle \triangleright [\![N]\!]_n \textbf{ in} \\
 & & \quad \textbf{def } m\langle k\rangle \triangleright k\langle x, v\rangle \textbf{ in } [\![M]\!]_m
\end{array}
$$

Intuition: a λ-expression is a join process that sends the result of its computation
Some magic ☺

$$
[\![MN]\!]_v \triangleq \textbf{def } m\langle k\rangle \mid n\langle k'\rangle \triangleright k\langle k', v\rangle \textbf{ in } [\![M]\!]_m \mid [\![N]\!]_n
$$

A problem in concurrency [2]

Problem Definition

Santa Claus sleeps in his shop up at the North Pole, and can only be wakened by either all nine reindeer being back from their year long vacation on the beaches of some tropical island in the South Pacific, or by some elves who are having some difficulties making the toys. One elf's problem is never serious enough to wake up Santa (otherwise, he may **never** get any sleep), so, the elves visit Santa in a group of three. When three elves are having their problems solved, any other elves wishing to visit Santa must wait for those elves to return. If Santa wakes up to find three elves waiting at his shop's door, along with the last reindeer having come back from the tropics, Santa has decided that the elves can wait until after Christmas, because it is more important to get his sleigh ready as soon as possible. (It is assumed that the reindeer don't want to leave the tropics, and therefore they stay there until the last possible moment. They might not even come back, but since Santa is footing the bill for their year in paradise ... This could also explain the quickness in their delivering of presents, since the reindeer can't wait to get back to where it is warm.) The penalty for the last reindeer to arrive is that it must get Santa while the others wait in a warming hut before being harnessed to the sleigh.

A Solution

The solution that has worked best over the years, and also appears to be the simplest, is written using C statements and pseudo-code. (Constants are also used in case the number of reindeer were to change, or if the group size of "solution-seeking" elves is modified.) Basically, the reindeer arrive, update the count of how many have arrived, and the last one wakes up Santa. An elf, upon discovering a problem, attempts to modify the count for the number of elves with a problem and either: waits outside Santa's shop if he/she is the first or second such elf; knocks on the door and wakes up Santa if that elf is the third one; or waits in the elves' shop until the elves currently with Santa start coming back. (The code for this solution can be found in the Appendix.)

```
1  receive
2      {reindeer, Pid1} and {reindeer, Pid2} and {reindeer, Pid3}
3        and {reindeer, Pid4} and {reindeer, Pid5} and {reindeer, Pid6}
4        and {reindeer, Pid7} and {reindeer, Pid8} and {reindeer, Pid9} ->
5          io:format("Ho, ho, ho! Let's deliver presents!~n"),
6          [Pid1, Pid2, Pid3, Pid4, Pid5, Pid6, Pid7, Pid8, Pid9];
7      {elf, Pid1} and {elf, Pid2} and {elf, Pid3} ->
8          io:format("Ho, ho, ho! Let's discuss R&D possibilities!~n"),
9          [Pid1, Pid2, Pid3]
10 end
```

The solution with semaphores takes about 2 pages of C code [2]!

Where're join patterns? [Dagstuhl 24051, Jan-Feb 2024]

Where're join patterns? [Dagstuhl 24051, Jan-Feb 2024]

Where're join patterns? [Dagstuhl 24051, Jan-Feb 2024]

Where're join patterns? [Dagstuhl 24051, Jan-Feb 2024]

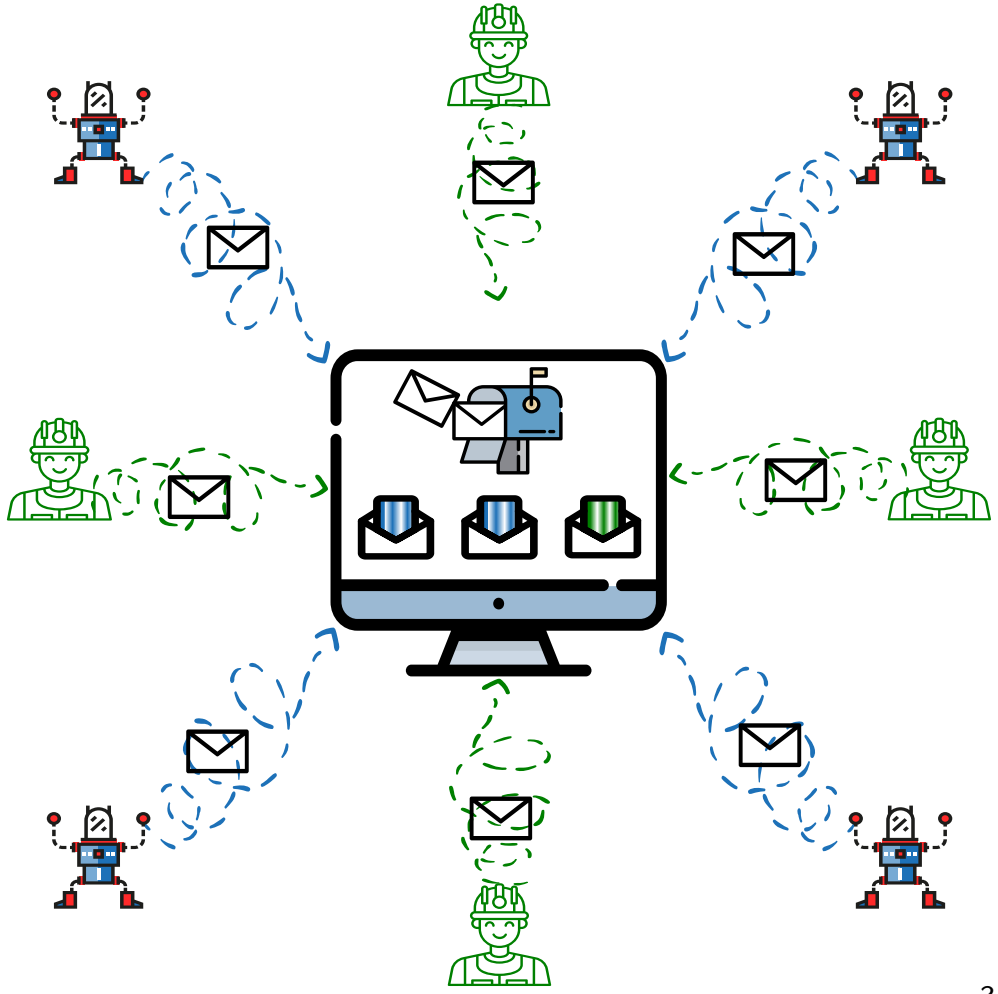
Where're join patterns? [Dagstuhl 24051, Jan-Feb 2024]

JoinActors

The remaining slides of this lecture are courtesy of Ayman Hussein.

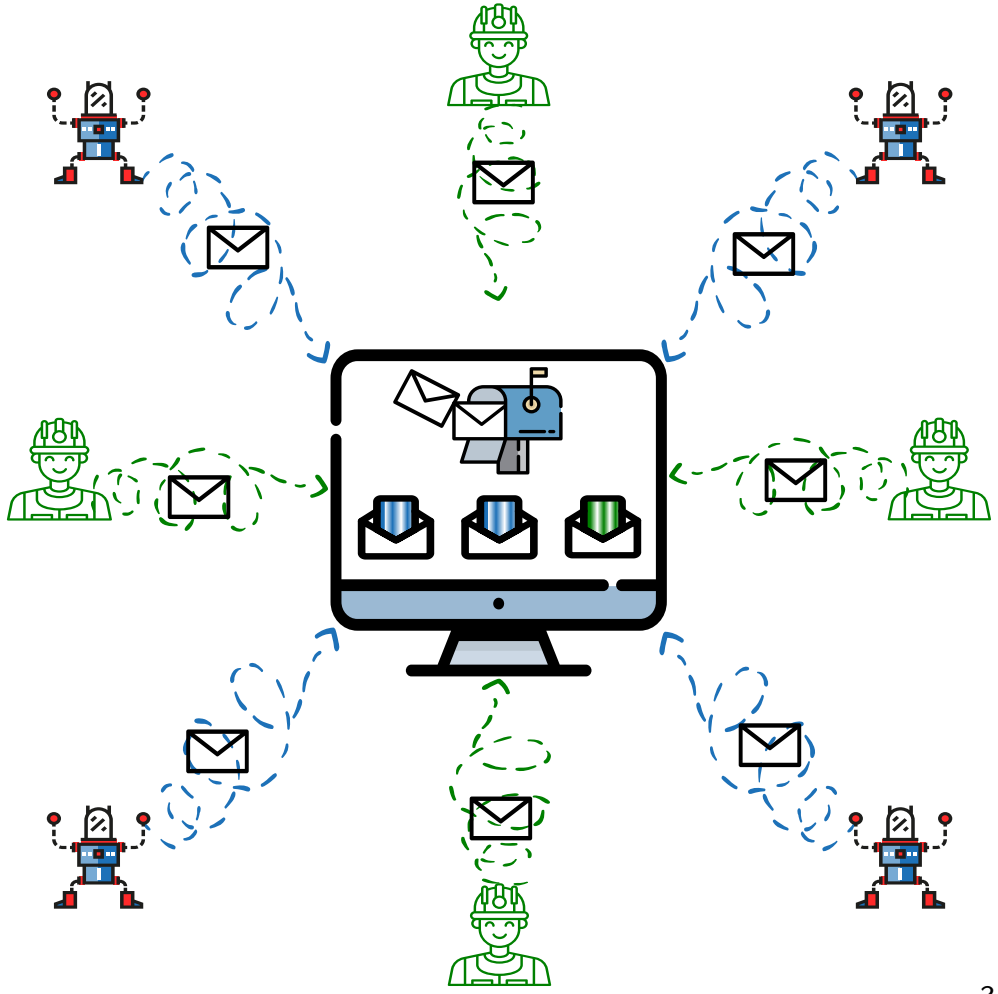
Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers



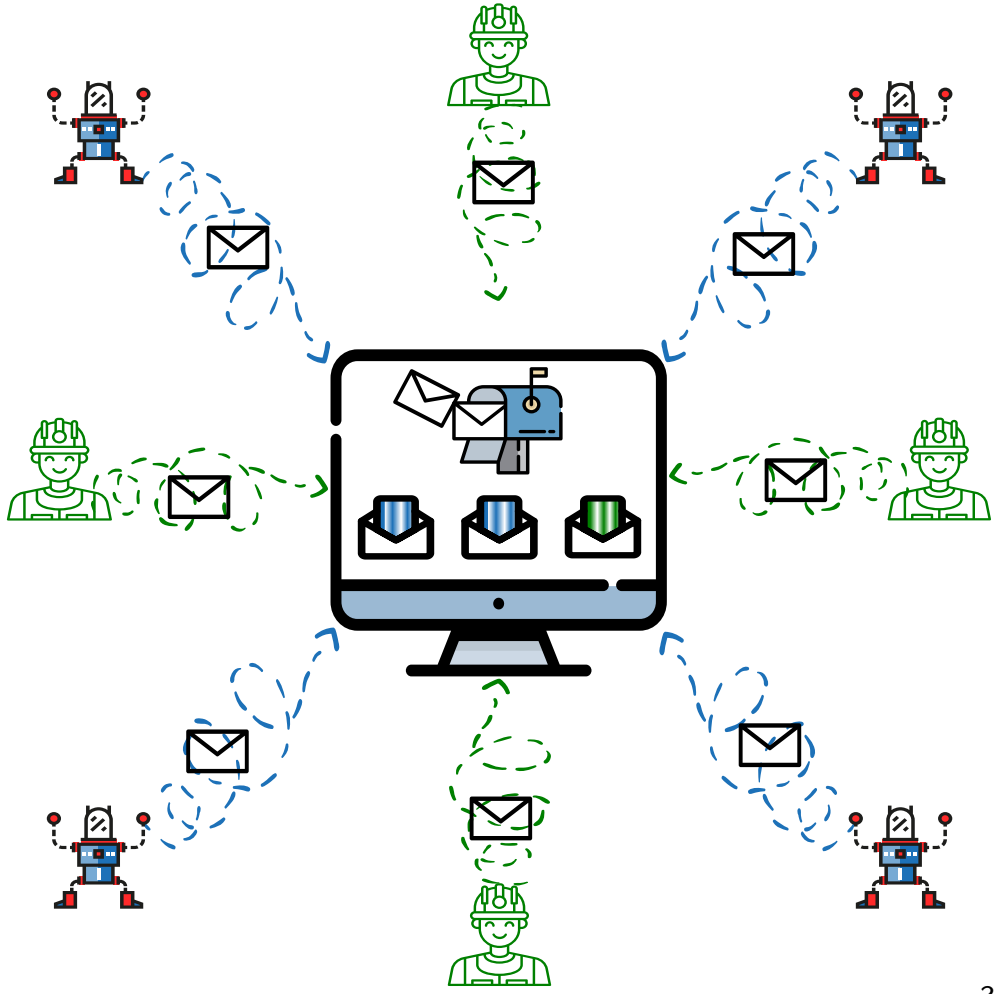
Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers
- Messages arrive **asynchronously** and **out-of-order**



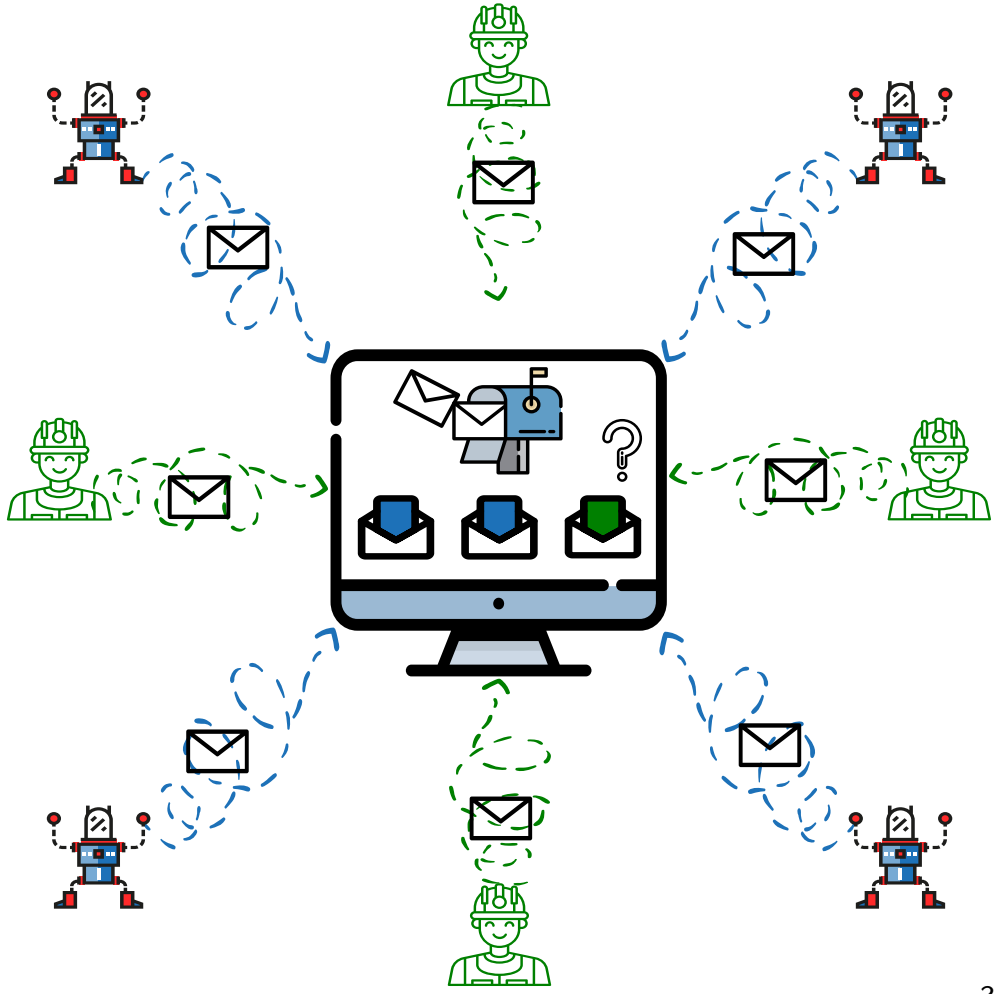
Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers
- Messages arrive **asynchronously** and **out-of-order**
- Monitor reacts to a combination of messages in the mailbox



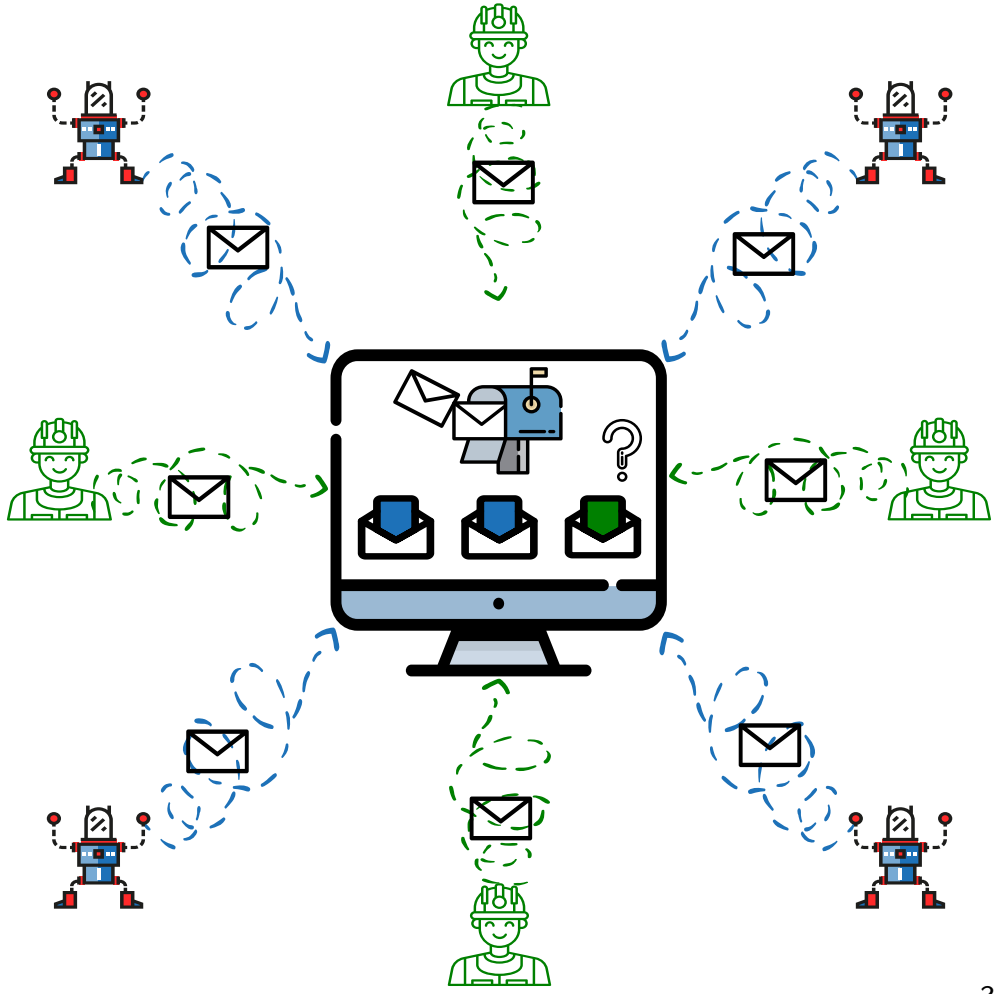
Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers
- Messages arrive **asynchronously** and **out-of-order**
- Monitor reacts to a combination of messages in the mailbox
- Traditionally, programmers write **custom code** for message association

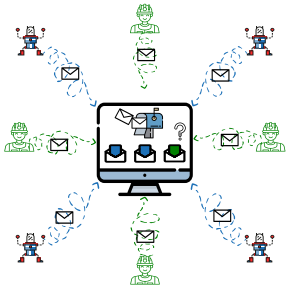


Monitoring a Factory Shop Floor

- The monitoring program must associate machine **Fault** notifications to **Fix** notifications from workers
- Messages arrive **asynchronously** and **out-of-order**
- Monitor reacts to a combination of messages in the mailbox
- Traditionally, programmers write **custom code** for message association (e.g., Akka/Pekko actors, Socket programming)



Factory Shop Monitor Using JoinActors

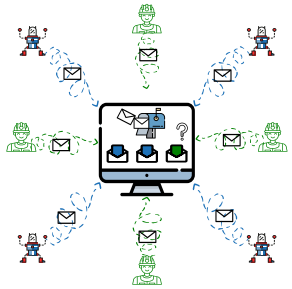


Using our `JoinActors` library we can **declaratively** specify **order-independent message associations**

```
1  def monitor() = Actor[...] {
2    receive { (...) => {
3      case (Fault(id1, _), Fix(id2, _)) if id1 == id2 => ...
4      case (Fault(_, ts1), Fault(id2, ts2), Fix(id3, _))
5                     if id2 == id3 && ts2 - ts1 > TEN_MIN => ...
6    }}
7  }
```

- Uses Scala 3 macros

Join Patterns More Formally



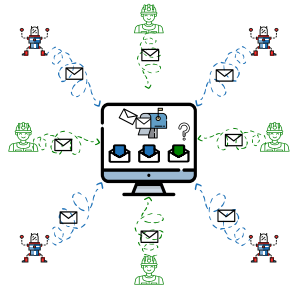
Let D = $\Pi_1 + \Pi_2$ where

Π_1 = **Fault**$(id_1, \_)$ $\wedge$ **Fix**$(id_2, \_)$ if $id_1 = id_2$

Π_2 = **Fault**$(\_, t_1)$ $\wedge$ **Fault**(id_2, t_2) $\wedge$ **Fix**$(id_3, \_)$ if $id_2 = id_3$ && $t_2 - t_1 > 10min$

Refer to the paper for more details

Join Patterns Matching



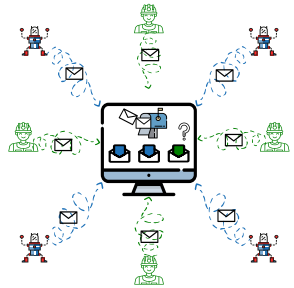
The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

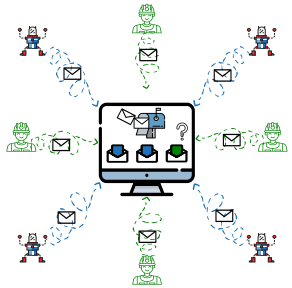
$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$\mathcal{M}$ =

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

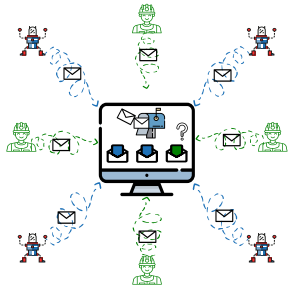
$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1\,(1, 10{:}35)\cdot$$

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

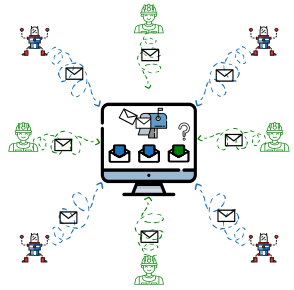
$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1\,(1, 10{:}35) \cdot \texttt{Fault}_2\,(2, 10{:}40) \cdot$$

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

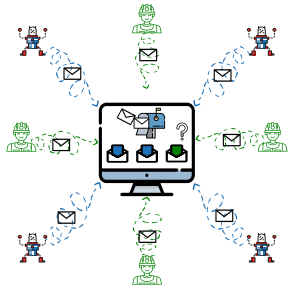
$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1\,(1, 10{:}35) \cdot \texttt{Fault}_2\,(2, 10{:}40) \cdot \texttt{Fault}_3\,(3, 10{:}55) \cdot$$

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

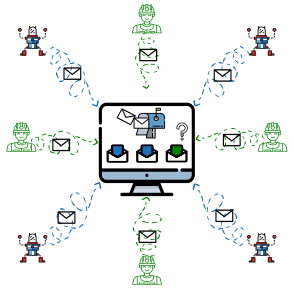
$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

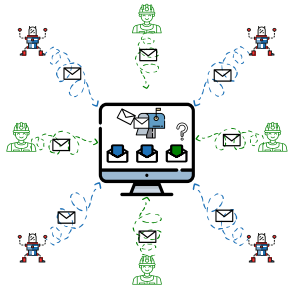
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$.

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

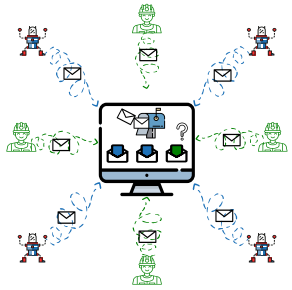
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$. **How and which one do we pick?**

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

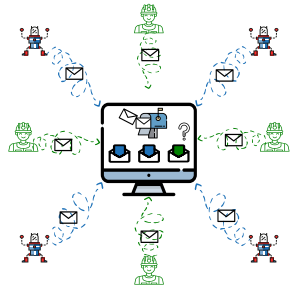
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$. **How and which one do we pick?**
  - $\Pi_1 : \langle\{\texttt{Fault}_3, \texttt{Fix}_4\}\rangle$

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

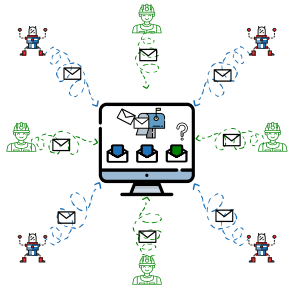
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$. **How and which one do we pick?**
  - $\Pi_1 : \langle \{\texttt{Fault}_3, \texttt{Fix}_4\} \rangle$
  - $\Pi_2 : \langle \{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}, \{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\} \rangle$

Join Patterns Matching



The join definition for the factory shop floor monitor is $\mathsf{D} = \Pi_1 + \Pi_2$ where

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

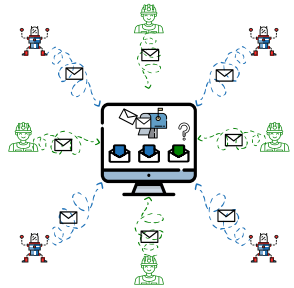
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

Now consider the following mailbox $\mathcal{M}$:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- We have many options to match from $\mathcal{M}$. **How and which one do we pick?**
  - $\Pi_1 : \langle \{\texttt{Fault}_3, \texttt{Fix}_4\} \rangle$
  - $\Pi_2 : \langle \{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}, \{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\} \rangle$
- In existing literature, the selection is either
  - **Non-deterministic** choice. This is usually undesirable
  - Pick **longest-matching sequence**

Our Proposal: "Fair Match"



Recall that we have the following $\mathsf{D} = \Pi_1 + \Pi_2$ where:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

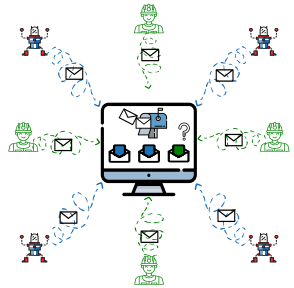
$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

And the following final mailbox configuration:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- A "fair" match is the one that consumes the **oldest** messages in $\mathcal{M}$
- No message that can be matched is left in the mailbox **indefinitely**

Our Proposal: "Fair Match"



Recall that we have the following $\mathsf{D} = \Pi_1 + \Pi_2$ where:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

And the following final mailbox configuration:

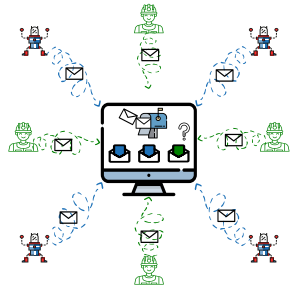
$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- A "fair" match is the one that consumes the **oldest** messages in $\mathcal{M}$
- No message that can be matched is left in the mailbox **indefinitely**
- Now we can pick the **fairest** match from $\mathcal{M}$:

$$\Pi_1 : \langle \{\texttt{Fault}_3, \texttt{Fix}_4\} \rangle$$

$$\Pi_2 : \langle \{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}, \{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\} \rangle$$

Our Proposal: "Fair Match"



Recall that we have the following $\mathsf{D} = \Pi_1 + \Pi_2$ where:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

And the following final mailbox configuration:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

- A "fair" match is the one that consumes the **oldest** messages in $\mathcal{M}$
- No message that can be matched is left in the mailbox **indefinitely**
- Now we can pick the **fairest** match from $\mathcal{M}$:

$$\Pi_1 : \langle \{\texttt{Fault}_3, \texttt{Fix}_4\} \rangle$$

$$\Pi_2 : \langle \{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\}, \{\texttt{Fault}_2, \texttt{Fault}_3, \texttt{Fix}_4\} \rangle$$

$$\mathsf{D} : \langle \{\texttt{Fault}_3, \texttt{Fix}_4\}, \{\texttt{Fault}_1, \texttt{Fault}_3, \texttt{Fix}_4\} \rangle$$

"Fair" Match Formalisation

We have formalised this notion of "fair" join pattern matching declaratively using inference rules:

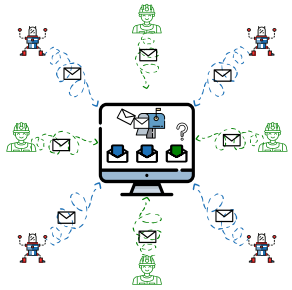
$$\frac{\forall i \in \{1, \ldots, n\} : \mu_i \sigma = m_i \qquad \gamma\sigma}{m_1 \cdot \ldots \cdot m_n \vDash_\sigma \mu_1 \wedge \ldots \wedge \mu_n \text{ if } \gamma}$$ **Match Messages Against Pattern**

$$\frac{\mathcal{M}[\mathcal{I}] \vDash_\sigma \Pi \text{ for some } \sigma}{\mathcal{M} \vDash_\mathcal{I} \Pi}$$ **Pick Messages From $\mathcal{M}$**

$$\frac{\mathcal{M} \vDash_\mathcal{I} \Pi \qquad \forall \mathcal{I}'.\ (\mathcal{M} \vDash_{\mathcal{I}'} \Pi \implies \mathcal{I} \leqslant_{\text{lex}} \mathcal{I}')}{\mathcal{M} \vDash \Pi \rightsquigarrow \mathcal{I}}$$ **Select Fairest Match**

- Translate inference rules into a "fair" message matching **brute-force algorithm**
- Current implementations use matching without fairness e.g. (Haller et al. COORDINATION 2008, Plociniczak and Eisenbach COORDINATION 2010, Avila et al. 2020)
- Refer to the paper for more details

Brute-force Algorithm for "Fair" Message Matching



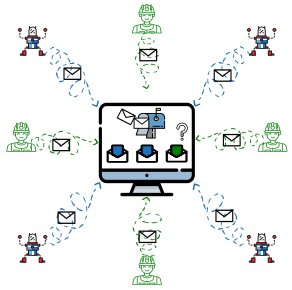
Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} =$$

Brute-force Algorithm for "Fair" Message Matching



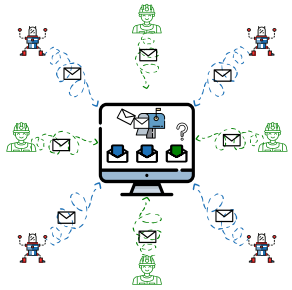
Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_)\,\cdot$$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

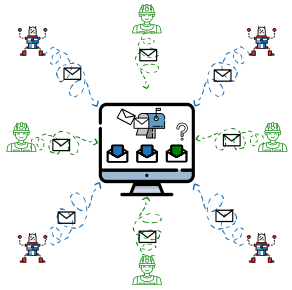
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_)\,\cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_)\rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

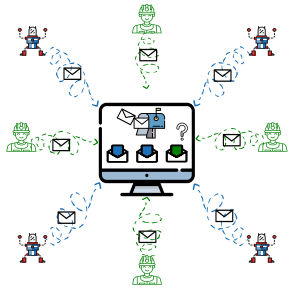
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1\,(3, \_)\,\cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1\,(3, \_)\rangle$ – Not enough messages ×

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

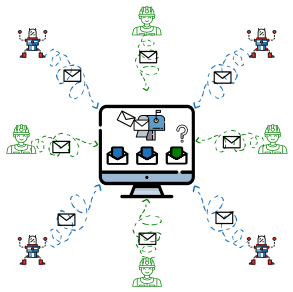
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1(3, \_) \rangle$ – Not enough messages ×

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

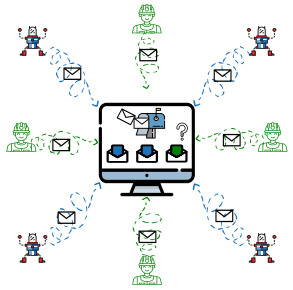
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1(3, \_) \rangle$ – Not enough messages ×
$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

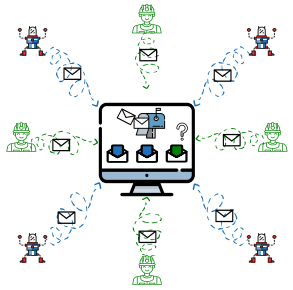
$$\mathcal{M} = \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \cdot \texttt{Fault}_3(2, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1(3, \_) \rangle$ – Not enough messages ×

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

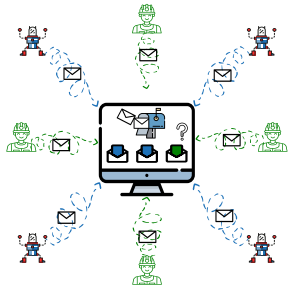
and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \cdot \texttt{Fault}_3(2, \_) \cdot$$

- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1(3, \_) \rangle$ – Not enough messages ×
$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$
$\mathcal{M}[1 \cdot 2 \cdot 3]$:

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \cdot \texttt{Fault}_3(2, \_) \cdot$$

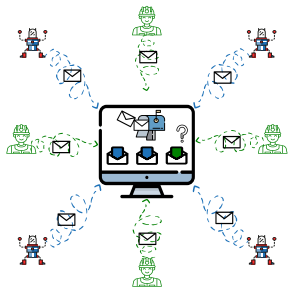
- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1(3, \_) \rangle$ – Not enough messages ×

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \cdot \texttt{Fault}_3(2, \_) \cdot$$

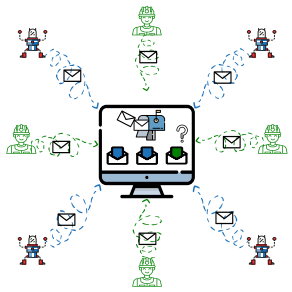
- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1(3, \_) \rangle$ – Not enough messages ×

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$, $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_3(2, \_) \rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \cdot \texttt{Fault}_3(2, \_) \cdot \texttt{Fault}_4(3, \_)$$

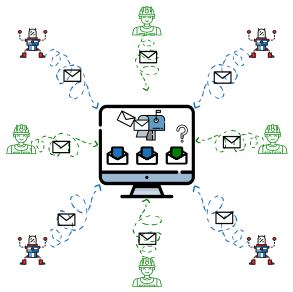
- Find a match for Π_1 from $\mathcal{M}$

$\mathcal{M}[1]$: $\langle \texttt{Fix}_1(3, \_) \rangle$ – Not enough messages ×

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$, $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_3(2, \_) \rangle$

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \cdot \texttt{Fault}_3(2, \_) \cdot \texttt{Fault}_4(3, \_)$$

- Find a match for Π_1 from $\mathcal{M}$

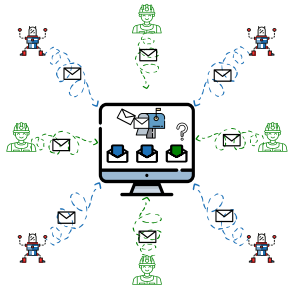
$\mathcal{M}[1]$: $\langle \texttt{Fix}_1(3, \_) \rangle$ – Not enough messages ×

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$, $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_3(2, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3 \cdot 4]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$, $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_3(2, \_) \rangle$,

Brute-force Algorithm for "Fair" Message Matching



Naive algorithm that performs **redundant** matching attempts
We have that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \cdot \texttt{Fault}_3(2, \_) \cdot \texttt{Fault}_4(3, \_)$$

- Find a match for Π_1 from $\mathcal{M}$

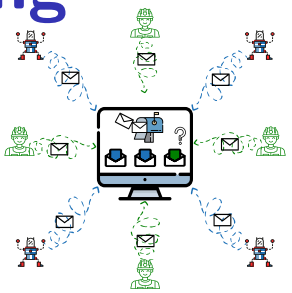
$\mathcal{M}[1]$: $\langle \texttt{Fix}_1(3, \_) \rangle$ – Not enough messages ×

$\mathcal{M}[1 \cdot 2]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$, $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_3(2, \_) \rangle$

$\mathcal{M}[1 \cdot 2 \cdot 3 \cdot 4]$: $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_2(1, \_) \rangle$, $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_3(2, \_) \rangle$, $\langle \texttt{Fix}_1(3, \_) \cdot \texttt{Fault}_4(3, \_) \rangle$

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

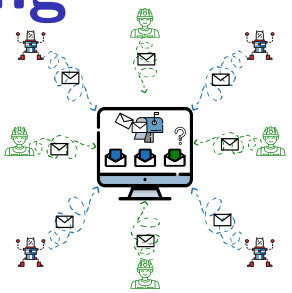
and the following mailbox:

$\mathcal{M}$ =

$\emptyset$

Check if $id_1 = id_2$

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

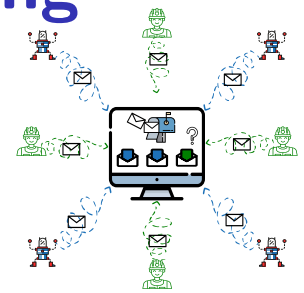
and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot$$

$\emptyset$

Check if $id_1 = id_2$

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

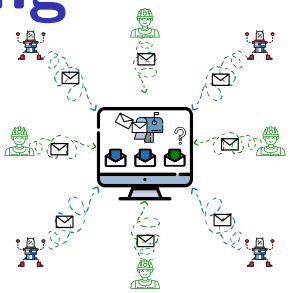
and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1\,(1, \_)\,\cdot$$

Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for "Fair" Message Matching



Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

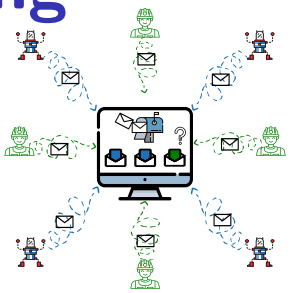
and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot$$

Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for "Fair" Message Matching



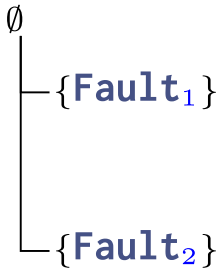
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

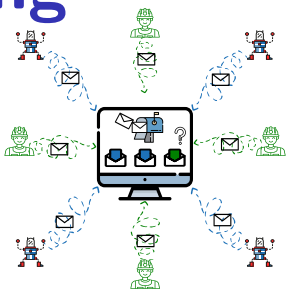
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot$$



Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for "Fair" Message Matching



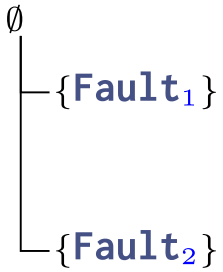
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

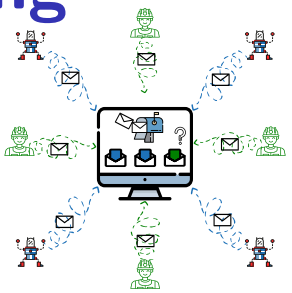
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot$$



Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for "Fair" Message Matching



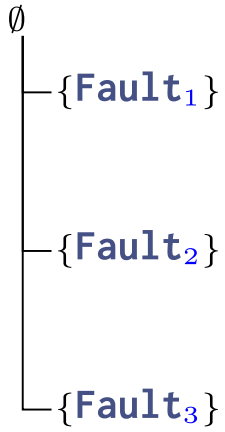
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

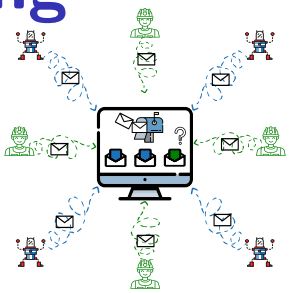
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot$$



Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for "Fair" Message Matching



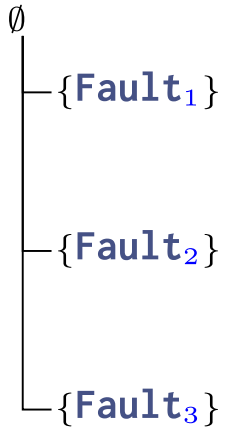
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

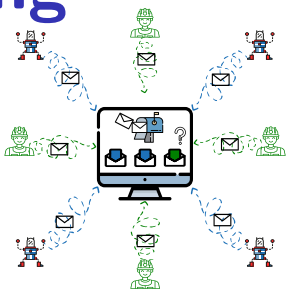
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot \texttt{Fix}_4(3, \_)$$



Check if $id_1 = id_2$

- Not enough messages to match

Stateful Tree-based Algorithm for "Fair" Message Matching



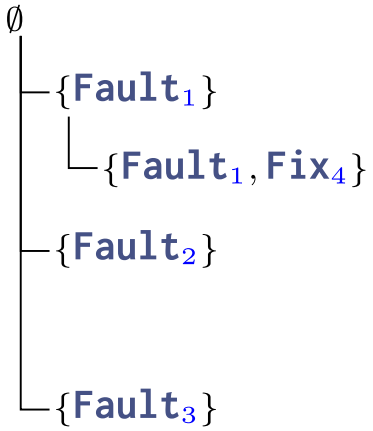
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

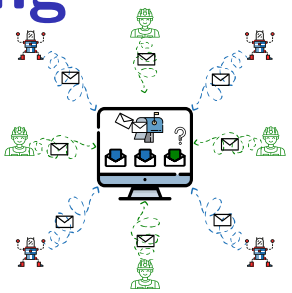
and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot \texttt{Fix}_4(3, \_)$$



Check if $id_1 = id_2$

Stateful Tree-based Algorithm for "Fair" Message Matching



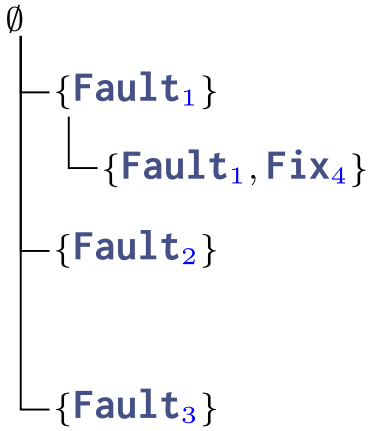
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

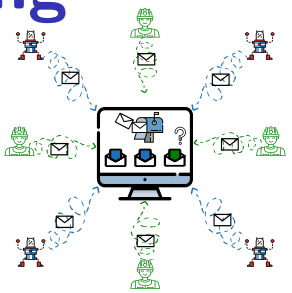
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot \texttt{Fix}_4(3, \_)$$



Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$

Stateful Tree-based Algorithm for "Fair" Message Matching



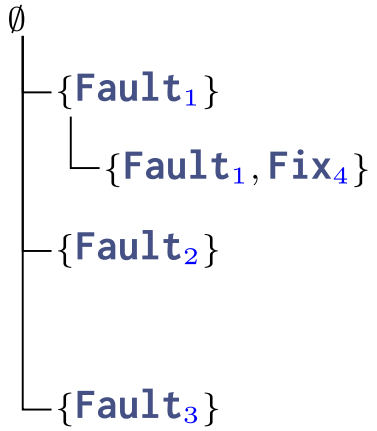
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

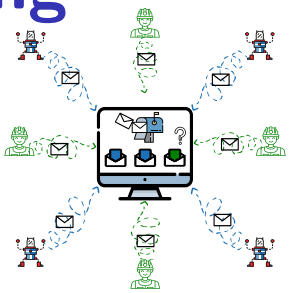
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot \texttt{Fix}_4(3, \_)$$



Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×

Stateful Tree-based Algorithm for "Fair" Message Matching



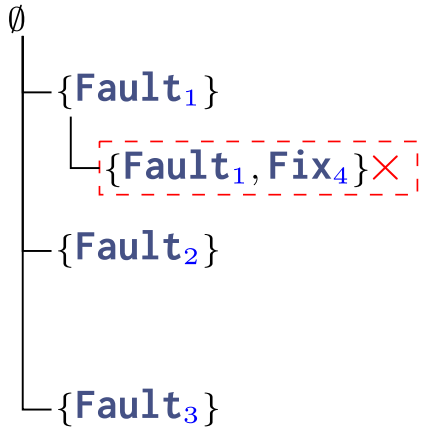
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

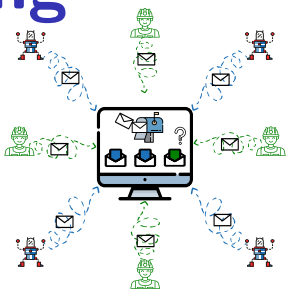
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot \texttt{Fix}_4(3, \_)$$



Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×

Stateful Tree-based Algorithm for "Fair" Message Matching



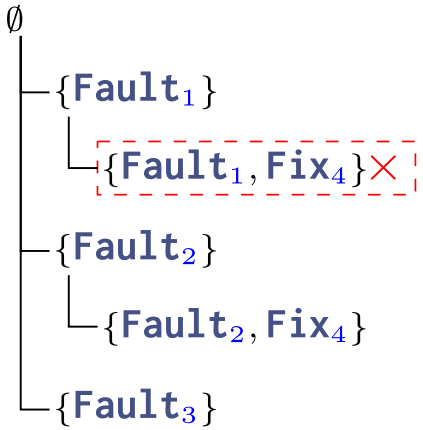
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

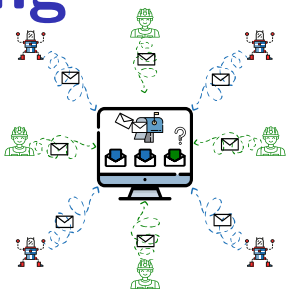
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot \texttt{Fix}_4(3, \_)$$



Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×

Stateful Tree-based Algorithm for "Fair" Message Matching



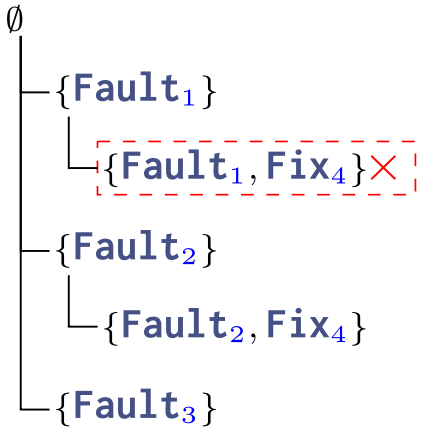
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

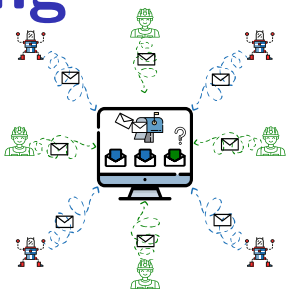
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot \texttt{Fix}_4(3, \_)$$



Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$

Stateful Tree-based Algorithm for "Fair" Message Matching



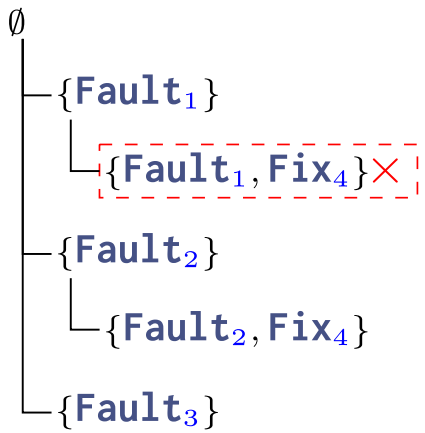
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

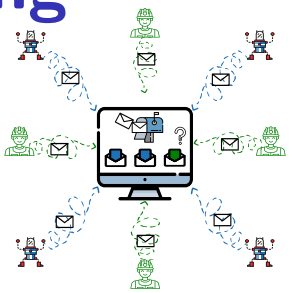
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot \texttt{Fix}_4(3, \_)$$



Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×

Stateful Tree-based Algorithm for "Fair" Message Matching



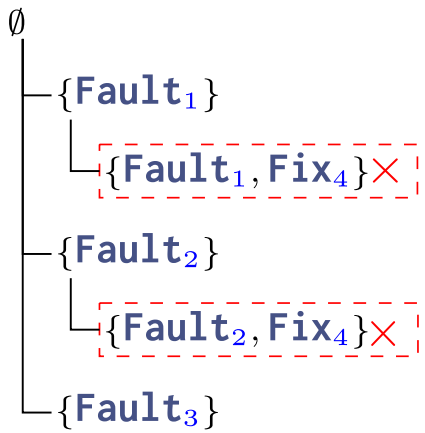
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

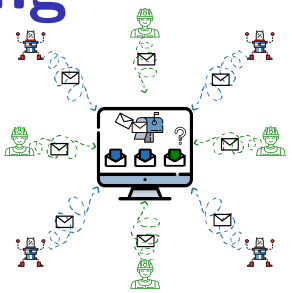
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot \texttt{Fix}_4(3, \_)$$



Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×

Stateful Tree-based Algorithm for "Fair" Message Matching



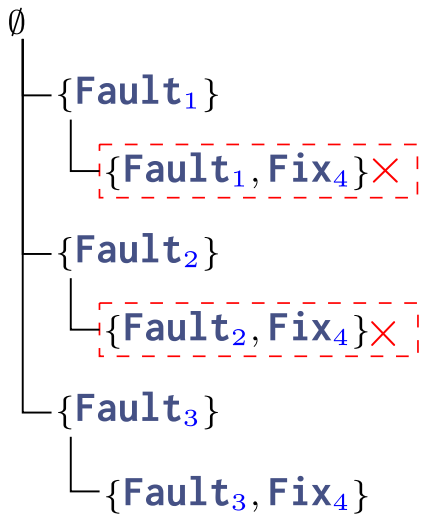
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

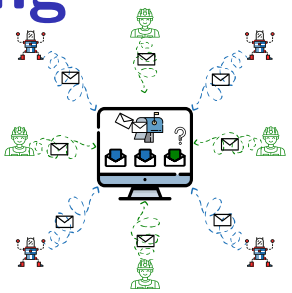
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot \texttt{Fix}_4(3, \_)$$



Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×

Stateful Tree-based Algorithm for "Fair" Message Matching



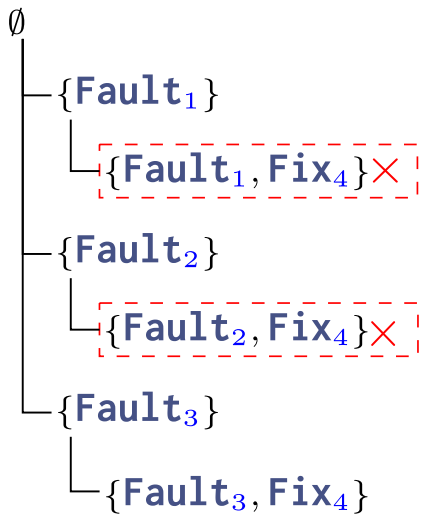
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

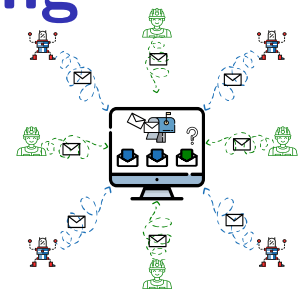
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot \texttt{Fix}_4(3, \_)$$



Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×
- **Attempt 3**: $3 = 3$

Stateful Tree-based Algorithm for "Fair" Message Matching



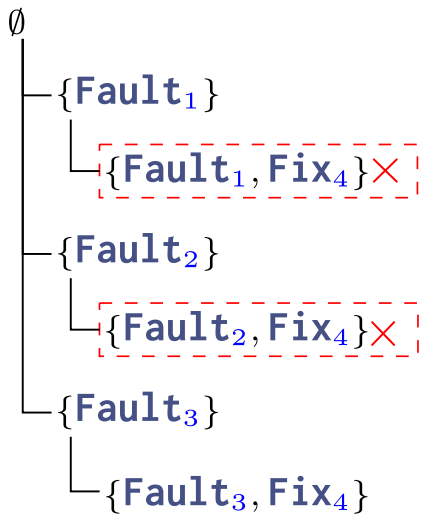
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

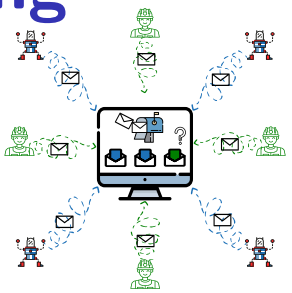
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot \texttt{Fix}_4(3, \_)$$



Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×
- **Attempt 3**: $3 = 3$ ✓

Stateful Tree-based Algorithm for "Fair" Message Matching



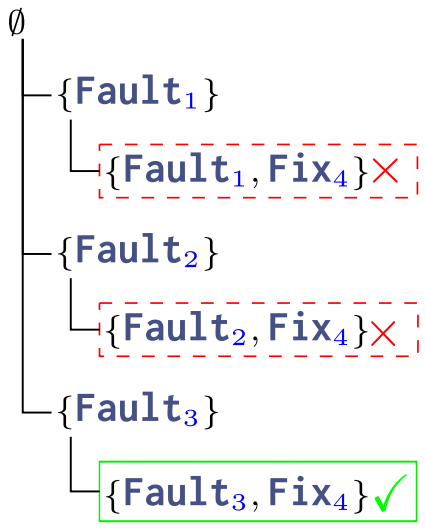
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

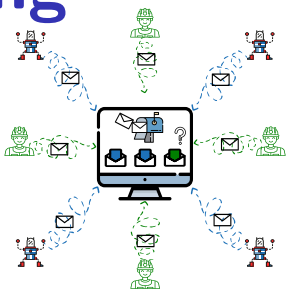
$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot \texttt{Fix}_4(3, \_)$$



Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×
- **Attempt 3**: $3 = 3$ ✓

Stateful Tree-based Algorithm for "Fair" Message Matching



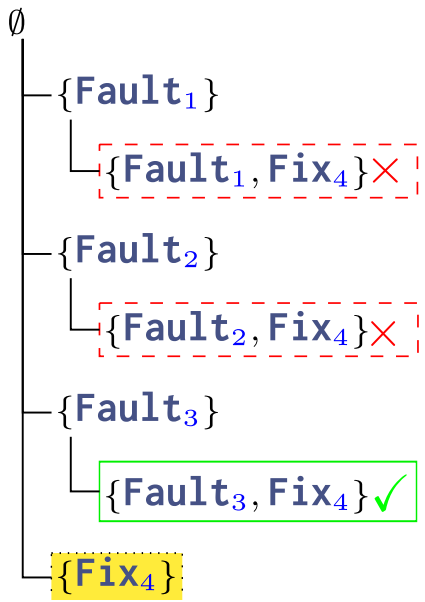
Use state to track **partial matches** and avoid redundant matching attempts

Recall that Π_1:

$$\Pi_1 = \texttt{Fault}(id_1, \_) \wedge \texttt{Fix}(id_2, \_) \text{ if } id_1 = id_2$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1(1, \_) \cdot \texttt{Fault}_2(2, \_) \cdot \texttt{Fault}_3(3, \_) \cdot \texttt{Fix}_4(3, \_)$$

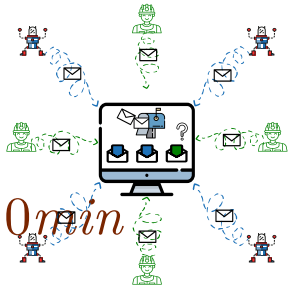


Check if $id_1 = id_2$

- **Attempt 1**: $1 \neq 3$ ×
- **Attempt 2**: $2 \neq 3$ ×
- **Attempt 3**: $3 = 3$ ✓

We don't record a partial match $\texttt{Fix}_4$ because we matched earlier

Tree Construction (continued)



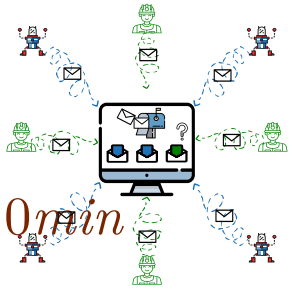
We now consider the second join pattern Π_2:

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$

Tree Construction (continued)

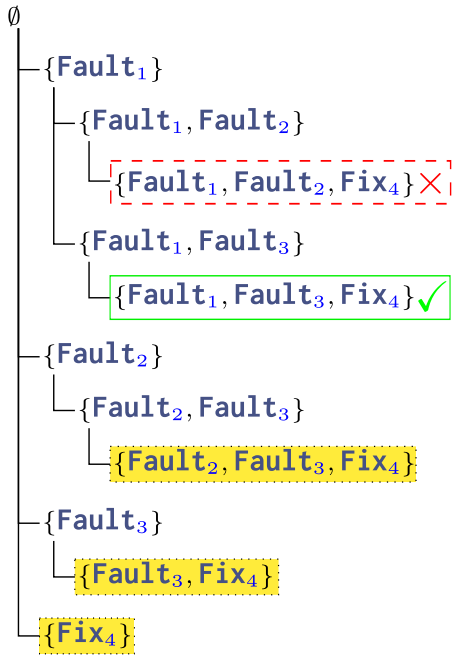


We now consider the second join pattern Π_2:

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$



Check if $id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$:

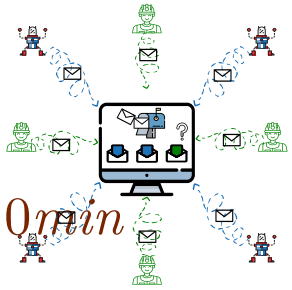
- **Attempt 1**:

  $1 = 3 \;\&\&\; 10:40 - 10:35 > 10min$ ×

- **Attempt 2**:

  $3 = 3 \;\&\&\; 10:55 - 10:35 > 10min$ ✓

Tree Construction (continued)

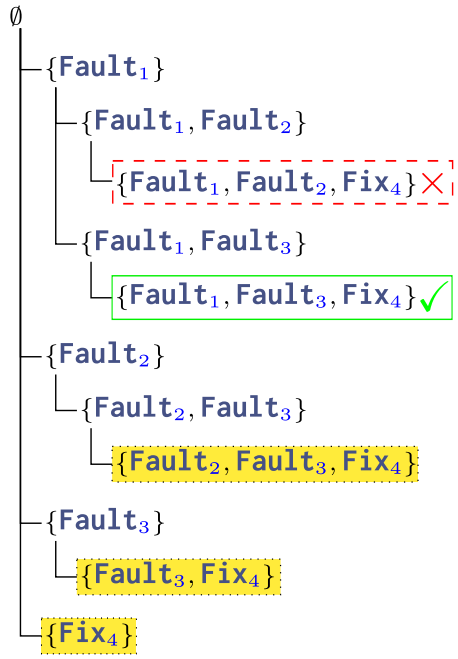


We now consider the second join pattern Π_2:

$$\Pi_2 = \texttt{Fault}(\_, t_1) \wedge \texttt{Fault}(id_2, t_2) \wedge \texttt{Fix}(id_3, \_) \text{ if } id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$$

and the following mailbox:

$$\mathcal{M} = \texttt{Fault}_1(1, 10{:}35) \cdot \texttt{Fault}_2(2, 10{:}40) \cdot \texttt{Fault}_3(3, 10{:}55) \cdot \texttt{Fix}_4(3, 11{:}00)$$



Check if $id_2 = id_3 \;\&\&\; t_2 - t_1 > 10min$:

- **Attempt 1**:

  $1 = 3 \;\&\&\; 10:40 - 10:35 > 10min$ ×

- **Attempt 2**:

  $3 = 3 \;\&\&\; 10:55 - 10:35 > 10min$ ✓

We avoid computing (partial) matches

Performance Evaluation

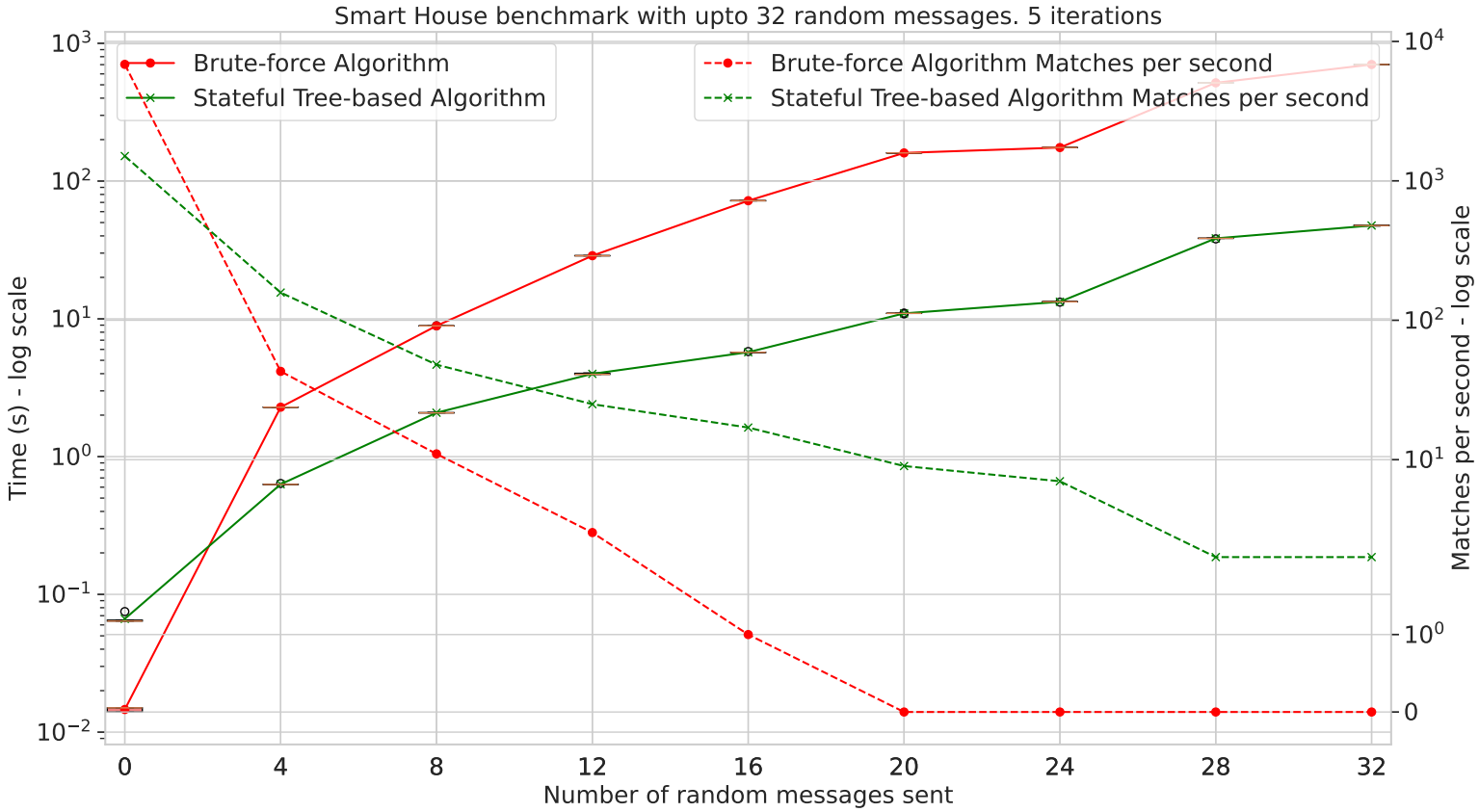


Figure: Smart House benchmark based on (Rodriguez-Avila et al. 2021)

References I

[1] Cedric Fournet and George Gonthier. The reflexive CHAM and the join-calculus.
In *Conference Record of POPL '96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 372–385, St. Petersburg Beach, Florida, January 1996.

[2] John A. Trono. A new exercise in concurrency.
SIGCSE Bull., 26(3):8–10, September 1994.