Specification and analysis of communication protocols

Emilio Tuosto @ GSSI

20-24 April
and
4-7 May, 2026
Dipartimento di Informatica, Pisa

Initial plan

Motivations

Topics distributed coordination

- Choreographic models
- An unusual choreographic model
- Adding join patterns to actors
- A surprising (at least for me) application in economics

Tooling

Summary & re-planning

- ✓ Motivations
- ✓ Topics distributed coordination
  - ✓ Choreographic models
    - **ChorGram**: a toolkit for g-choreographies
  - ✓ An unusual choreographic model
    - Tools for swarms
  - Programming actors
    - A surprising (at least for me) application in economics
    - Adding join patterns to actors

– Tooling –

A Modelling Exercise

Given a bank's API B s.t.

- GET authReq :: authenticate; return authFail or granted
- GET authWithdrawal :: request cash; return allow or deny
- GET getBalance :: get balance; return balance

A Modelling Exercise

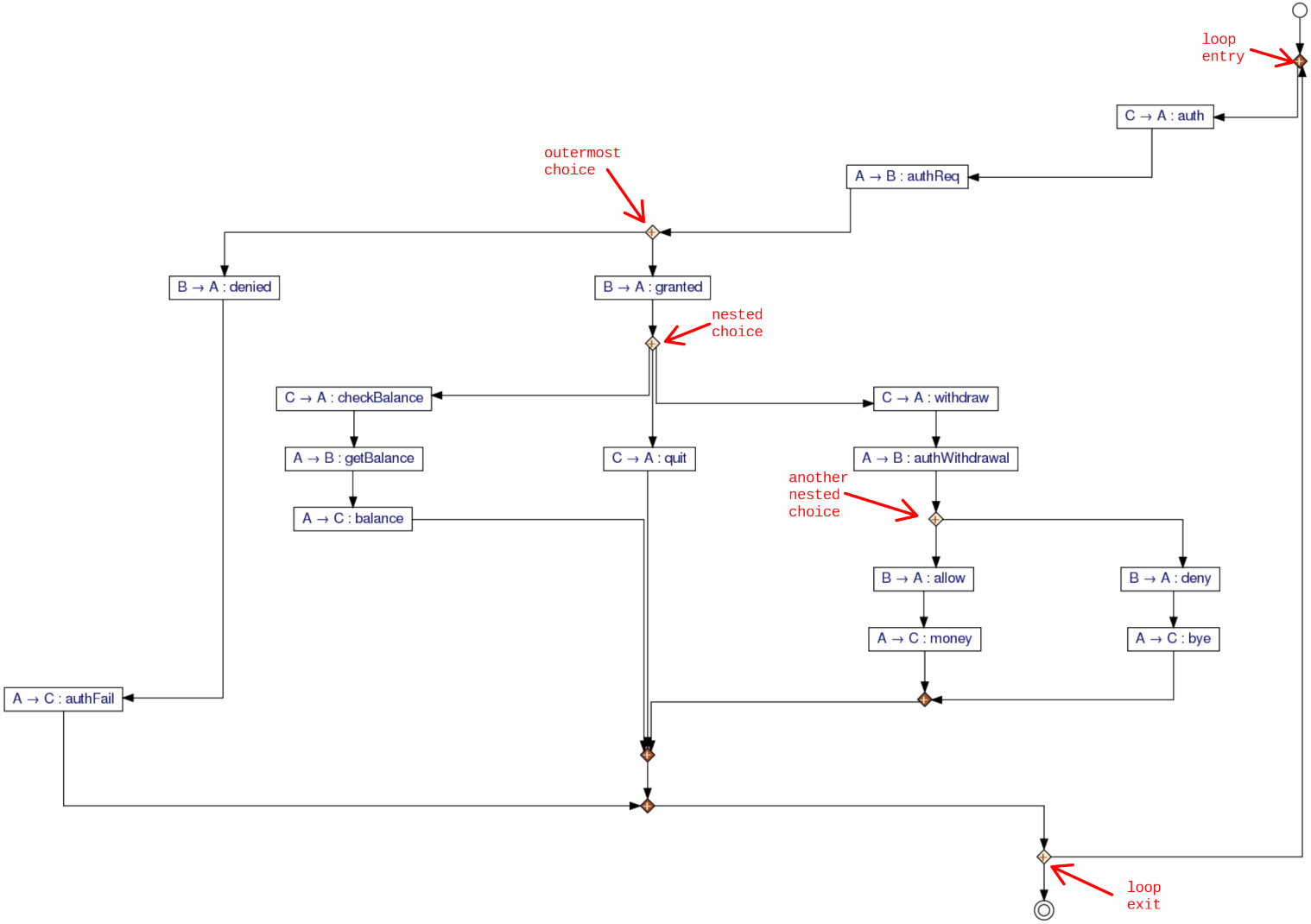
Given a bank's API B s.t.

- GET authReq :: authenticate; return authFail or granted
- GET authWithdrawal :: request cash; return allow or deny
- GET getBalance :: get balance; return balance

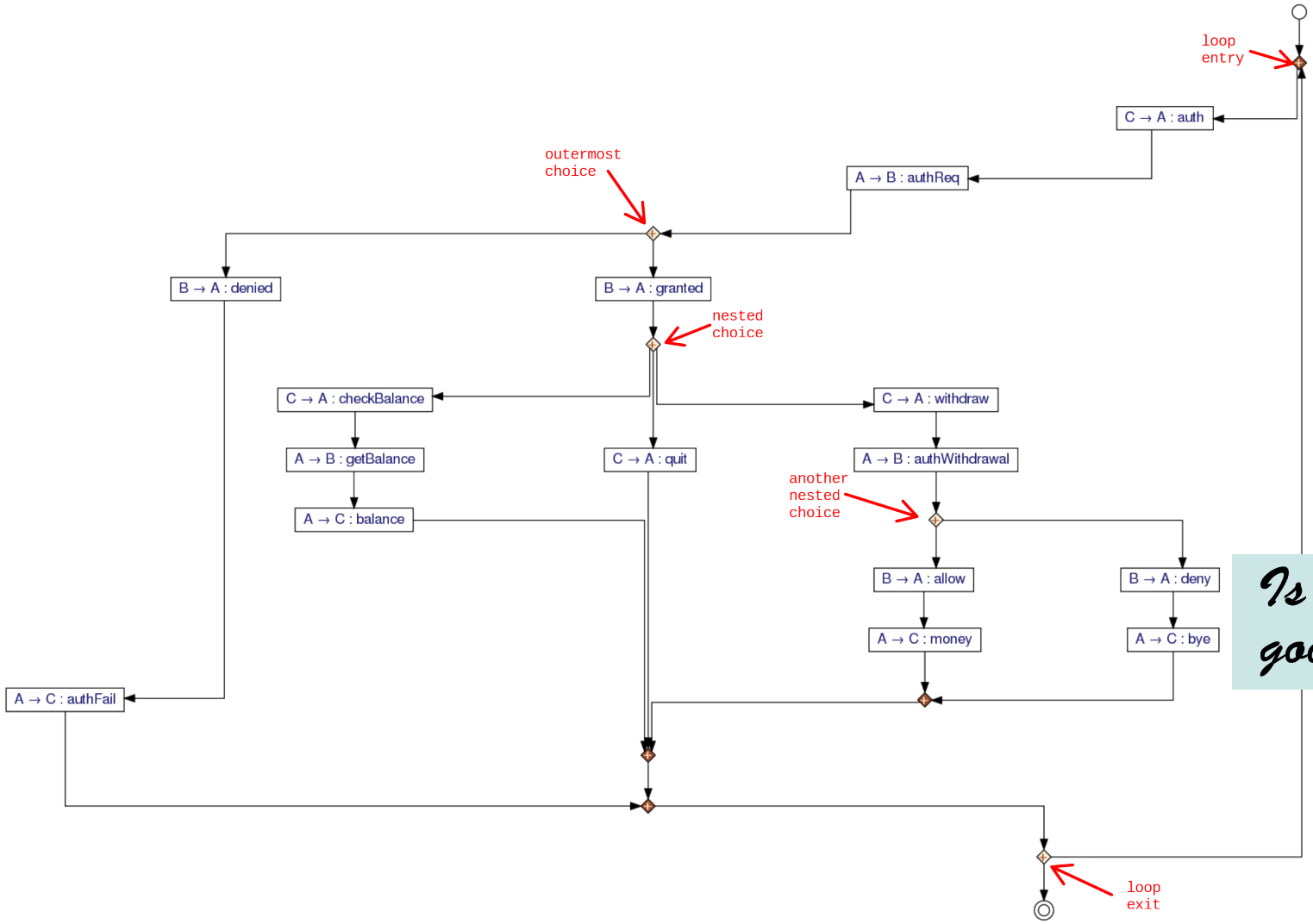
Design an ATM machine A

- GET auth :: authentication request; return authFail or granted
- GET withdraw :: request cash; return money or bye
- GET checkBalance :: check balance request; return balance
- ...

Define the global view

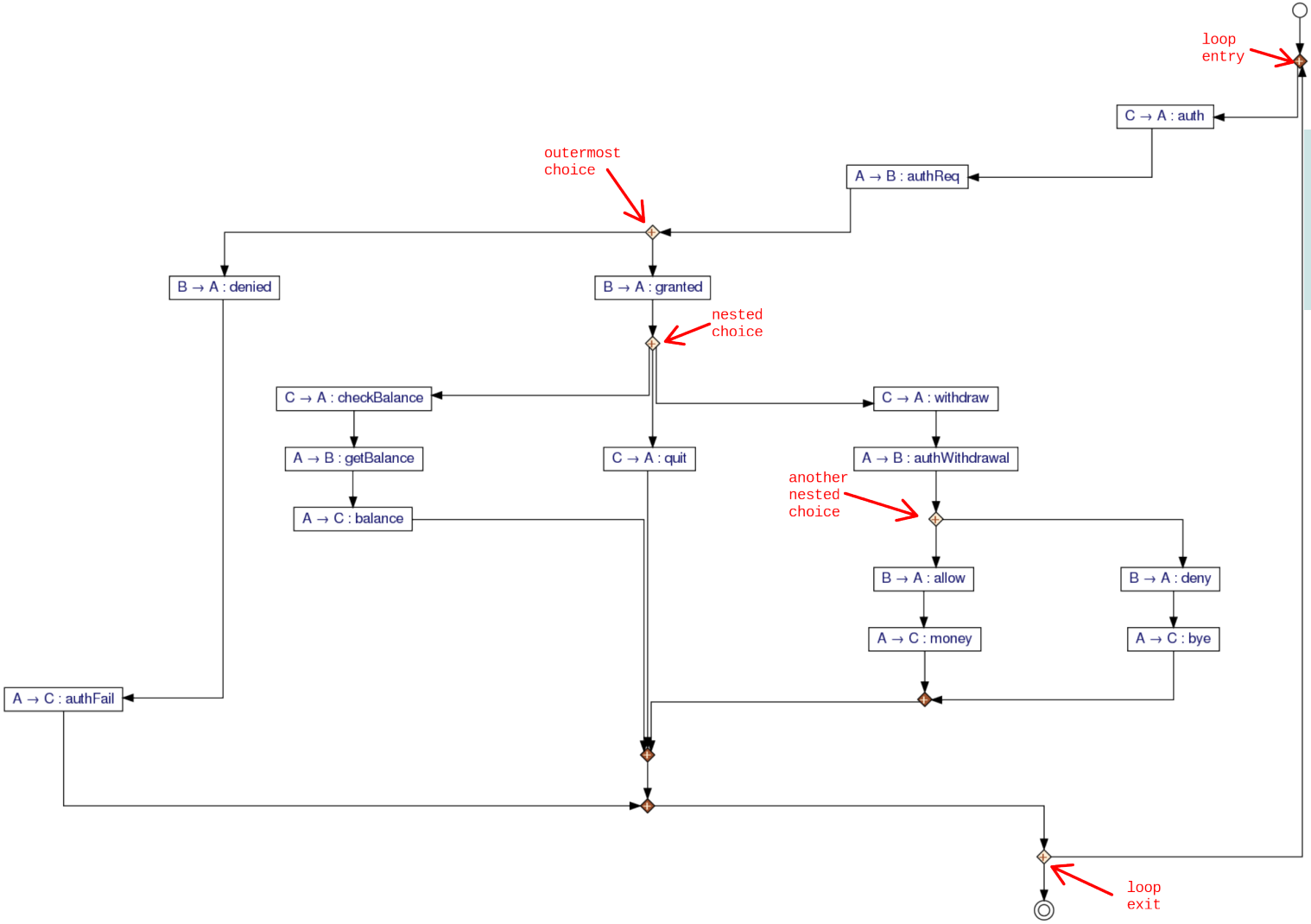


Define the global view



Is this any good?

Define the global view



on the highway to shell!

Other features of ChorGram

Chorgram toolkit enables

Other features of ChorGram

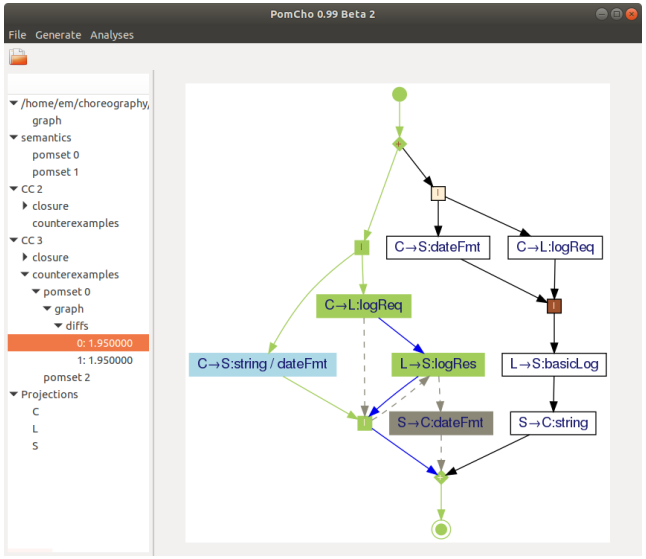
Chorgram toolkit enables

- Bottom-up analysis [19]

Other features of ChorGram

Chorgram toolkit enables

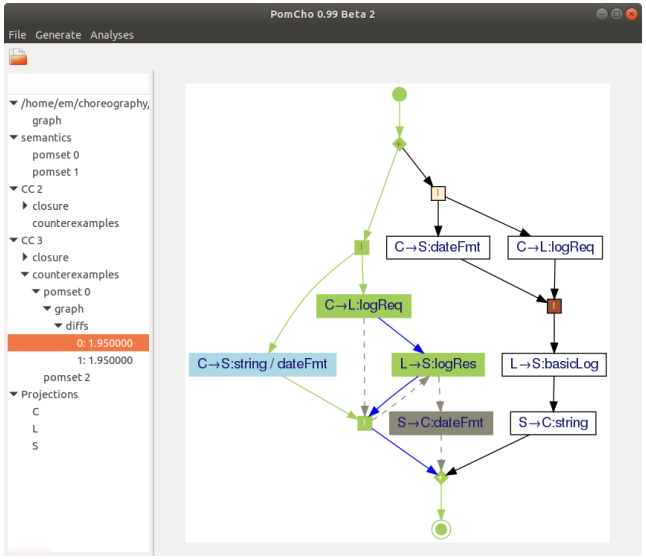
- Bottom-up analysis [19]
- Other Bug fixing facilities [10]



Other features of ChorGram

Chorgram toolkit enables

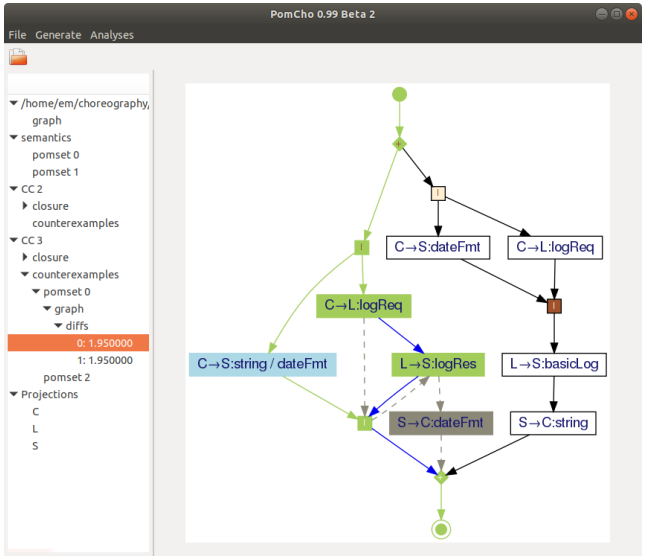
- Bottom-up analysis [19]
- Other Bug fixing facilities [10]
- Model-driven testing [3, 4, 5]



Other features of ChorGram

Chorgram toolkit enables

- Bottom-up analysis [19]
- Other Bug fixing facilities [10]



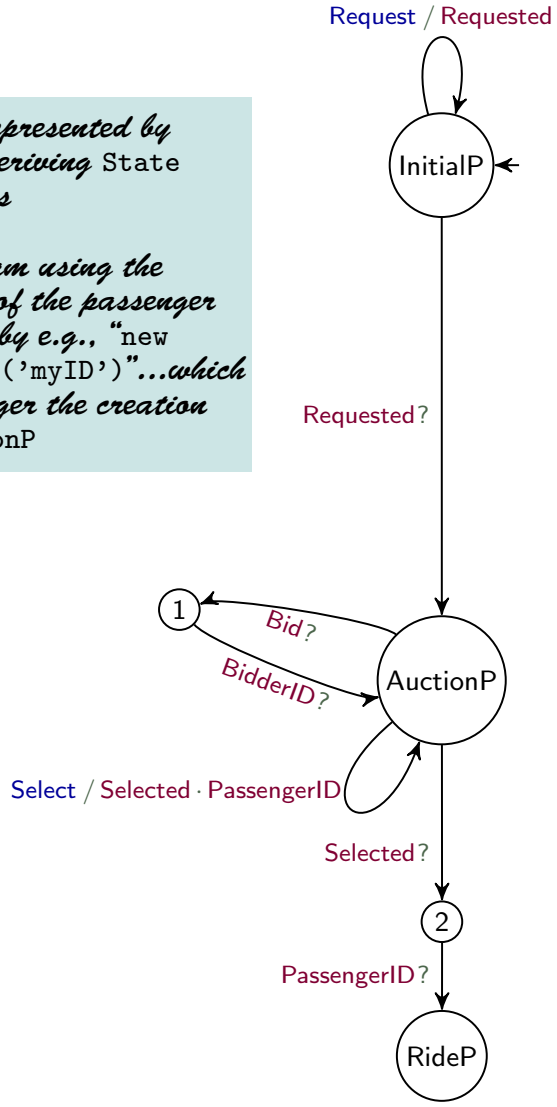
- Model-driven testing [3, 4, 5]
- Static analysis of SLAs [21, 22]

Coding swarms

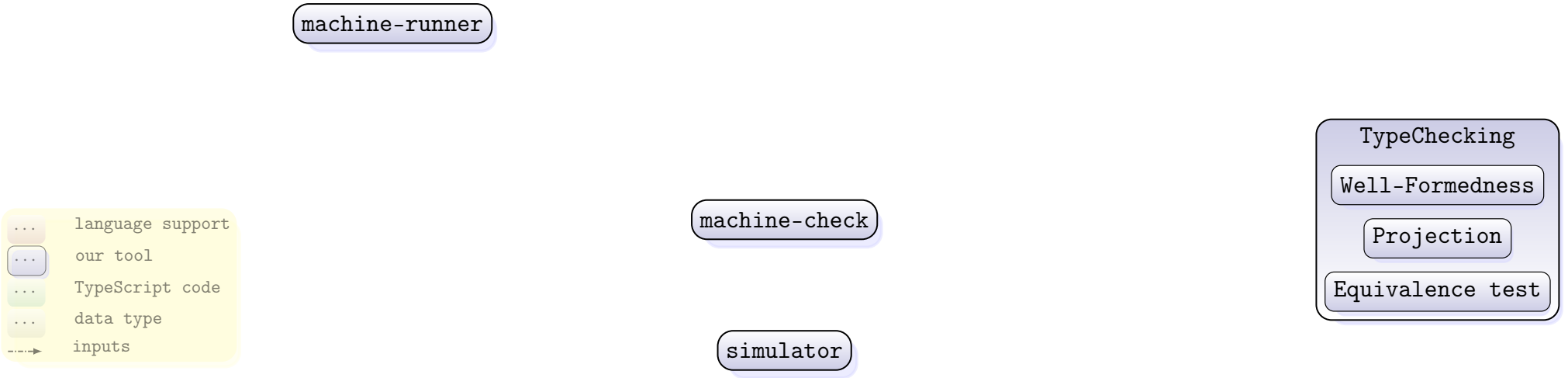
```
type Requested = { type: 'Requested'; pickup: string; dest: string }
type Events = Requested | Bid | BidderID | Selected | ...
/** Initial state for role P */
@proto('taxiRide') // decorator injects inferred protocol into runtime
export class InitialP extends State<Events> {
  constructor(public id: string) { super() }
  execRequest(pickup: string, dest: string) {
    return this.events({ type: 'Requested', pickup, dest })
  }
  onRequested(ev: Requested) {
    return new AuctionP(this.id, ev.pickup, ev.dest, [])
  }
}
@proto('taxiRide')
export class AuctionP extends State<Events> {
  constructor(public id: string, public pickup: string, public dest: string,
    public bids: BidData[]) { super() }
  onBid(ev1: Bid, ev2: BidderID) {
    const [ price, time ] = ev1
    this.bids.push({ price, time, bidderID: ev2.id })
    return this
  }
  execSelect(taxiId: string) {
    return this.events({ type: 'Selected', taxiID },
                       { type: 'PassengerID', id: this.id })
  }
  onSelected(ev: Selected, id: PassengerID) {
    return new RideP(this.id, ev.taxiID)
  }
}
@proto('taxiRide')
export class RideP extends State<Events> { ... }
```

States represented by classes deriving `State` ***base class***

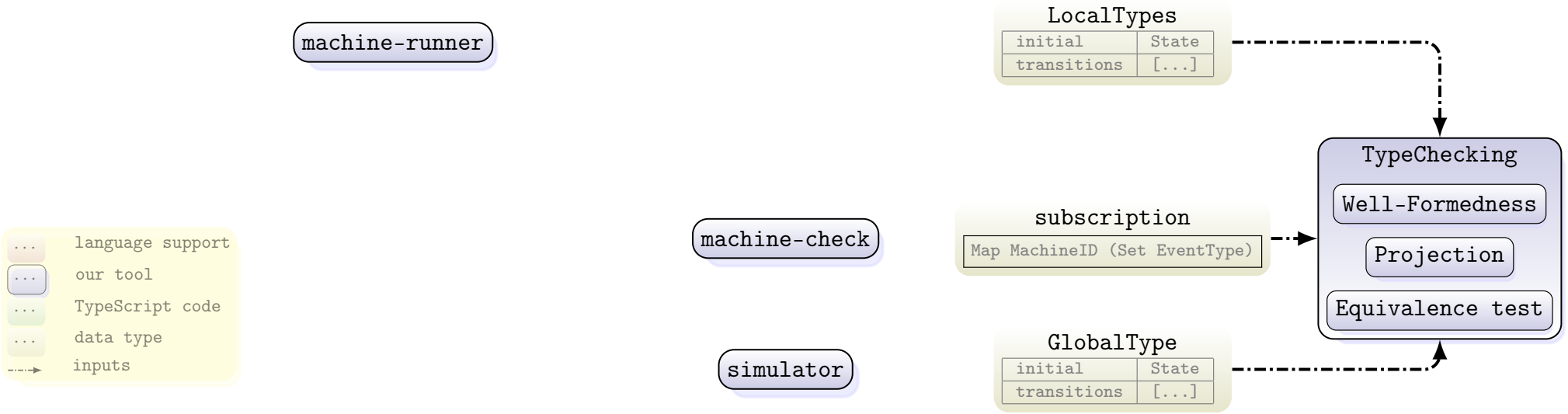
A program using the machine of the passenger P ***starts by e.g.,*** "`new InitialP('myID')`"***...which may trigger the creation of*** `AuctionP`



Swarms in practice [18, 16]

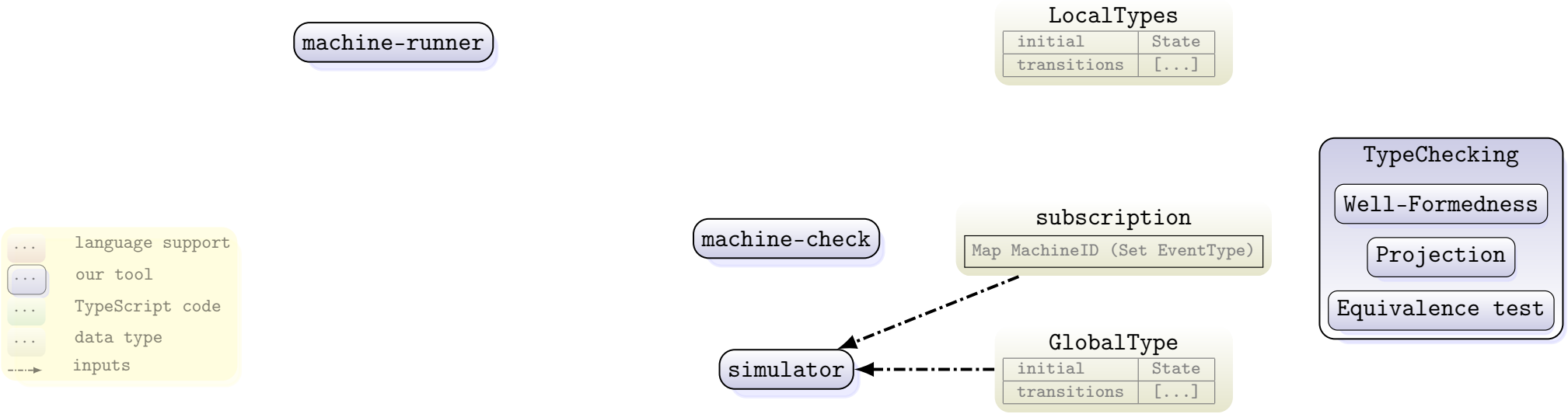


Swarms in practice [18, 16]



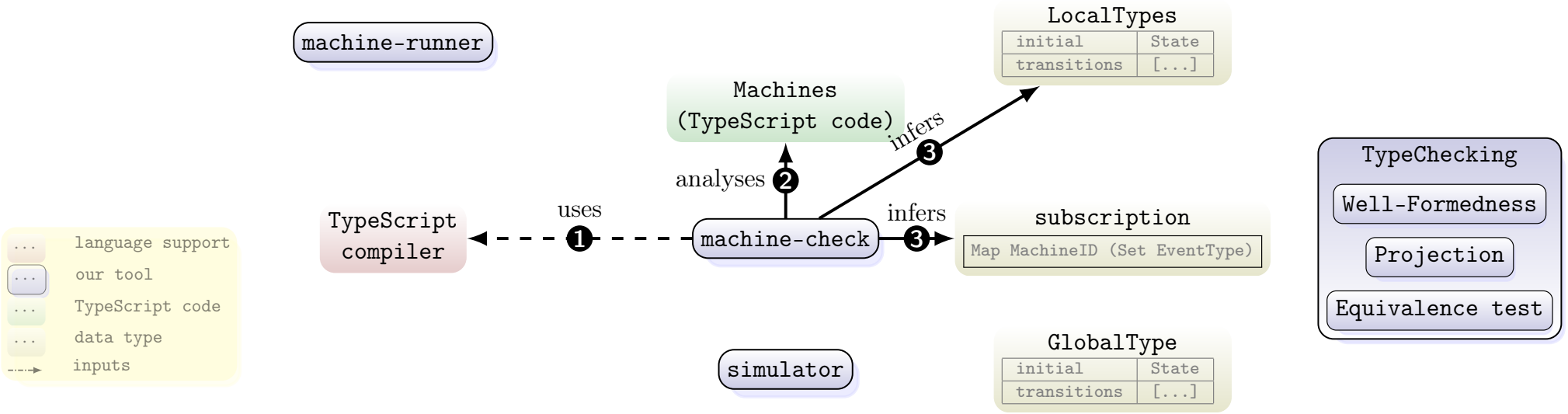
- `TypeChecking` implements the functionalities of our typing discipline

Swarms in practice [18, 16]



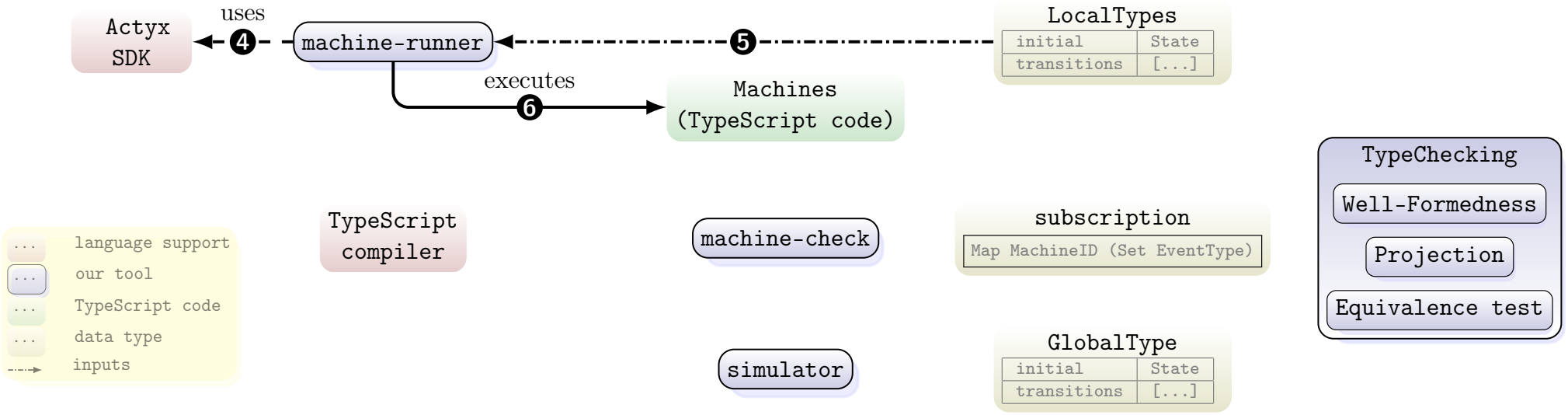
- `TypeChecking` implements the functionalities of our typing discipline
- `simulator` simulates the semantics of swarm realisations

Swarms in practice [18, 16]



- `TypeChecking` implements the functionalities of our typing discipline
- `simulator` simulates the semantics of swarm realisations
- `machine-check` integrate our framework in the Actyx platform

Swarms in practice [18, 16]



- `TypeChecking` implements the functionalities of our typing discipline
- `simulator` simulates the semantics of swarm realisations
- `machine-check` integrate our framework in the Actyx platform
- `machine-runner` interprets local types subscribing to event types, processes the past events, and waits for commands to be given

– 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)
- Mailbox-based communication:

  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 processed 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.
```

“The reflexive chemistry” [8, § 3]

$$
\begin{array}{rcl}
P & \triangleq & x\langle\tilde{v}\rangle \\
 & \mid & \textbf{def } D \textbf{ in } P \\
 & \mid & P \mid P \\
 & \mid & \mathbf{0}
\end{array}
\qquad
\begin{array}{rcl}
J & \triangleq & x\langle\tilde{v}\rangle \\
 & \mid & J \mid J \\
 & & \text{assume all } \tilde{v} \text{ fresh}
\end{array}
\qquad
\begin{array}{rcl}
D & \triangleq & J \triangleright P \\
 & \mid & D \wedge D
\end{array}
$$

"The reflexive chemistry" [8, § 3]

$$
\begin{array}{rcl}
P & \triangleq & x\langle\tilde{v}\rangle \\
 & | & \textbf{def } D \textbf{ in } P \\
 & | & P \mid P \\
 & | & \mathbf{0}
\end{array}
\qquad
\begin{array}{rcl}
J & \triangleq & x\langle\tilde{v}\rangle \\
 & | & J \mid J \\
 & & \text{assume all } \tilde{v} \text{ fresh}
\end{array}
\qquad
\begin{array}{rcl}
D & \triangleq & J \triangleright P \\
 & | & D \wedge D
\end{array}
$$

$\mathcal{R}$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 [8, § 3]

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

$$
\begin{array}{l}
\textbf{def}\ \mathrm{mkcell}\langle v_0, k\rangle\triangleright \\
\quad \textbf{def} \\
\qquad s\langle v\rangle \mid \mathrm{get}\langle k\rangle \quad \triangleright \quad s\langle v\rangle \mid k\langle v\rangle \\
\qquad \wedge \\
\qquad s\langle v\rangle \mid \mathrm{set}\langle u, k\rangle \quad \triangleright \quad s\langle u\rangle \mid k\langle\rangle \\
\quad \textbf{in} \\
\qquad s\langle v_0\rangle \mid k\langle \mathrm{get}, \mathrm{set}\rangle \\
\textbf{in} \\
\quad \textbf{def}\ k\langle \mathrm{get}, \mathrm{set}\rangle \triangleright \mathbf{0}\ \textbf{in}\ \mathrm{set}\langle 0, k\rangle \\
\mathrm{mkcell}\langle 3, c\rangle
\end{array}
$$

An "simple" example [8, § 3]

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

$$
\begin{array}{l}
\textbf{def}\ \mathrm{mkcell}\langle v_0, k\rangle \triangleright \\
\quad \textbf{def} \\
\qquad s\langle v\rangle \mid \mathrm{get}\langle k\rangle \quad \triangleright \quad s\langle v\rangle \mid k\langle v\rangle \\
\qquad \wedge \\
\qquad s\langle v\rangle \mid \mathrm{set}\langle u, k\rangle \quad \triangleright \quad s\langle u\rangle \mid k\langle\rangle \\
\quad \textbf{in} \\
\qquad s\langle 3\rangle \mid c\langle \mathrm{get}, \mathrm{set}\rangle \\
\textbf{in} \\
\quad \textbf{def}\ k\langle \mathrm{get}, \mathrm{set}\rangle \triangleright \mathbf{0}\ \textbf{in}\ \mathrm{set}\langle 0, k\rangle \\
\mathrm{mkcell}\langle 3, c\rangle
\end{array}
$$

An "simple" example [8, § 3]

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

$$
\begin{array}{l}
\textbf{def}\ \mathrm{mkcell}\langle v_0, k\rangle\triangleright \\
\quad \textbf{def} \\
\qquad s\langle v\rangle \mid \mathrm{get}\langle k\rangle \quad \triangleright \quad s\langle v\rangle \mid k\langle v\rangle \\
\qquad \wedge \\
\qquad s\langle v\rangle \mid \mathrm{set}\langle u, k\rangle \quad \triangleright \quad s\langle u\rangle \mid k\langle\rangle \\
\quad \textbf{in} \\
\qquad s\langle v_0\rangle \mid k\langle \mathrm{get}, \mathrm{set}\rangle \\
\textbf{in} \\
\quad \textbf{def}\ k\langle \mathrm{get}, \mathrm{set}\rangle \triangleright \mathbf{0}\ \textbf{in}\ \mathrm{set}\langle 0, k\rangle \\
\mathrm{mkcell}\langle 3, c\rangle
\end{array}
$$

References I

[1] Gul Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*.
MIT Press, Cambridge, MA, USA, 1986.

[2] Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines.
JACM, 30(2):323–342, 1983.

[3] Alex Coto, Roberto Guanciale, and Emilio Tuosto. An abstract framework for choreographic testing.
In *ICE*, volume abs/2009.07990, 2020.

[4] Alex Coto, Roberto Guanciale, and Emilio Tuosto. Choreographic development of message-passing applications - A tutorial.
In Simon Bliudze and Laura Bocchi, editors, *Coordination Models and Languages - 22nd IFIP WG 6.1 International Conference, COORDINATION 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15-19, 2020, Proceedings*, volume 12134 of *Lecture Notes in Computer Science*, pages 20–36. Springer, 2020.

[5] Alex Coto, Roberto Guanciale, and Emilio Tuosto. An abstract framework for choreographic testing.
Journal of Logic and Algebraic Methods in Programming, 123:100712, 2021. Extended version of [4].

[6] Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata.
In *ESOP 2012*, pages 194–213, 2012.

[7] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process.
J. ACM, 32(2):374–382, apr 1985.

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

References II

[9] Roberto Guanciale and Emilio Tuosto. An abstract semantics of the global view of choreographies.
In *Proceedings 9th Interaction and Concurrency Experience, ICE 2016, Heraklion, Greece, 8-9 June 2016.*, pages 67–82, 2016.

[10] Roberto Guanciale and Emilio Tuosto. Realisability of pomsets.
Journal of Logic and Algebraic Methods in Programming, 108:69–89, 2019.

[11] Roberto Guanciale and Emilio Tuosto. Pomcho: a tool chain for choreographic design.
Science of Computer Programming, 202:102535, 2021.

[12] Philipp Haller, Ayman Hussein, Hernán C. Melgratti, Alceste Scalas, and Emilio Tuosto. Fair join pattern matching for actors.
In Jonathan Aldrich and Guido Salvaneschi, editors, *38th European Conference on Object-Oriented Programming, ECOOP 2024, Vienna, Austria, September 16-20, 2024*, volume 313 of *LIPIcs*, pages 17:1–17:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[13] Philipp Haller, Ayman Hussein, Hernán C. Melgratti, Alceste Scalas, and Emilio Tuosto. Fair join pattern matching for actors (artifact).
Dagstuhl Artifacts Ser., 10(2):8:1–8:3, 2024.

[14] Carl Hewitt, Peter Boehler Bishop, and Richard Steiger. A Universal Modular ACTOR Formalism for Artificial Intelligence.
In Nils J. Nilsson, editor, *Proceedings of the 3rd International Joint Conference on Artificial Intelligence. Standford, CA, USA, August 20-23, 1973*, pages 235–245. William Kaufmann, 1973.

[15] Ayman Hussein, Philipp Haller, Ioannis Karras, Hernán C. Melgratti, Alceste Scalas, and Emilio Tuosto. Joinactors: A modular library for actors with join patterns.
Art Sci. Eng. Program., 11(1), 2026.

References III

[16] Nickolas Kavantzas, Davide Burdett, Gregory Ritzinger, Tony Fletcher, and Yves Lafon. Web services choreography description language version 1.0.
http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217. Working Draft 17 December 2004.

[17] Roland Kuhn and Alan Darmasaputra. Behaviorally typed state machines in typescript for heterogeneous swarms.
In René Just and Gordon Fraser, editors, *Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2023, Seattle, WA, USA, July 17-21, 2023*, pages 1475–1478. ACM, 2023.

[18] Roland Kuhn, Hernán C. Melgratti, and Emilio Tuosto. Behavioural types for local-first software.
In Karim Ali and Guido Salvaneschi, editors, *37th European Conference on Object-Oriented Programming, ECOOP 2023, July 17-21, 2023, Seattle, Washington, United States*, volume 263 of *LIPIcs*, pages 15:1–15:28. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[19] Roland Kuhn, Hernán C. Melgratti, and Emilio Tuosto. Behavioural types for local-first software (artifact).
Dagstuhl Artifacts Ser., 9(2):14:1–14:5, 2023.

[20] Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From Communicating Machines to Graphical Choreographies.
In *POPL15*, pages 221–232, 2015.

[21] Hubert Plociniczak and Susan Eisenbach. Jerlang: Erlang with joins.
In *Coordination Models and Languages: 12th International Conference, COORDINATION 2010, Amsterdam, The Netherlands, June 7-9, 2010. Proceedings 12*, pages 61–75. Springer, 2010.

[22] Carlos G. Lopez Pombo, Agustín E. Martinez Suñé, and Emilio Tuosto. A dynamic temporal logic for quality of service in choreographic models.
In Erika Ábrahám, Clemens Dubslaff, and Silvia Lizeth Tapia Tarifa, editors, *Theoretical Aspects of Computing – ICTAC 2023*, pages 119–138. Springer, 2023.

References IV

[23] Carlos López Pombo, Agustín E. Martinez Suñé, and Emilio Tuosto. A dynamic temporal logic for quality of service in choreographic models.
TCS, 1043:115247, 2025.

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

[25] Emilio Tuosto and Roberto Guanciale. Semantics of global view of choreographies.
Journal of Logic and Algebraic Methods in Programming, 95:17–40, 2018. Revised and extended version of [?]. available at http://www.cs.le.ac.uk/people/et52/jlamp-with-proofs.pdf.