MAGπ: Types for Failure-Prone Communication

Matthew Alan Le Brun
University of Glasgow
m.le-brun.1@research.gla.ac.uk

Ornela Dardha
University of Glasgow
ornela.dardha@glasgow.ac.uk

This talk proposal is based on work accepted for publication at ESOP'23 [7]. We introduce MAGπ—Multiparty, Asynchronous and Generalised π-calculus—an extension of generalised session type theory [10] into a calculus capable of modelling *non-Byzantine faults*, for various physical topologies and network assumptions. Our contributions are: *(1)* a calculus and type-system enriched with *timeouts* and *message loss semantics*—capable of modelling the widest set of *non-Byzantine faults*; *(2)* a novel and most general definition of *reliability*, allowing MAGπ to model physical topologies of distributed systems; *(3)* a generalised theory capable of specifying assumptions of underlying network protocols; and *(4)* type properties that lift the benefits of generalised MPST into our realm of failure-prone communication.

1 Modelling Failure

It is common for session type theories to assume that communication failures do not occur. The few (and rapidly increasing) works that consider failures tend to make heavy assumptions that impede their viability for realistic complex distributed applications. *E.g.*, *asynchronous* theories [8, 6, 10] model distributed communication under ideal "TCP-like" assumptions; *affine sessions* [9, 5, 3] do not support *arbitrary* failures that may stem from hardware faults, network inconsistencies *etc.*; *coordinator model* approaches [1, 4, 11] assume strict degrees of reliability (resilient processes, reliable broadcasts, *etc.*).

MAGπ is *the first language* to support the widest set of non-Byzantine faults, including *message loss*, *message delays* and *message reordering*; *crash failures* and *link failures*; and *network partitioning*. We achieve this by extending the multiparty session calculus [10] with a *failure-handling timeout branch* $c\,\&_{i\in I}\{[\mathsf{q}_i]\,?\mathsf{m}_i(\mathrm{d}).\mathrm{P}, \textcircled{\scriptsize t}.Q\}$, and operational semantics that directly model *message delay* and *message loss* (figure 1). The combination of timeouts and message loss allows MAGπ to model all the aforementioned faults.

We also equip the type system with optional timeout branches $\&_{i\in I}\{\mathsf{p}_i\,?\mathsf{m}_i(\mathbb{T}_i).\mathbb{S}_i\ \ [,\textcircled{\scriptsize t}.\mathbb{S}']\}$. This allows arbitrary failures to be reasoned about at the *type level*. An example session type of the Ping utility from the Internet Control Message Protocol is given in appendix A.

$$\textbf{[R-}\textcircled{\scriptsize t}\textbf{]}\quad s[\mathsf{q}]\,\&_{i\in I}\{[\mathsf{p}_i]\,?\mathsf{m}_i(x_i).P_i, \textcircled{\scriptsize t}.Q\} \mid s:\sigma \;\longrightarrow\; Q \mid s{:}\sigma \qquad \text{(message delay)}$$
$$\textbf{[R-}\downarrow\textbf{]}\quad s:h\cdot\sigma \;\longrightarrow\; s:\sigma \qquad \text{(message loss)}$$

Figure 1: Reduction rules modelling primitive failures in MAGπ

2 Reliability

Reliability refers to defining *where* failures may (not) occur within the communication system. Previous related works [2, 1, 12] adopt various definitions. Generally, either one central reliable node, or a single

Submitted to:
PLACES 2023

set of globally reliable nodes, are statically configured and assumed to not be affected by any failures. We introduce a novel definition of *reliability* to model physical topologies of distributed systems—by taking into account the *viewpoint* of each participant. We allow reliability to be defined for *each* communicating entity, as every participant has a unique observation of the global system. *E.g.* consider two processes running on the same physical hardware, and two more processes residing on a geographically co-located server. Every entity in this configuration has a different outlook on which communication channels are prone to network failures.

Our theory is generic enough to handle both extremes: *(i) totally unreliable* configurations where all communication is prone to failure (example use-cases are consensus algorithms); or *(ii) totally reliable* configurations where no network failures occur.

3 Generalising Network Assumptions

Distributed algorithms are not all designed to operate over the same network assumptions. Low-level distributed programming generally makes minimal network assumptions by running over the User Datagram Protocol (UDP); application-level programming tends to assume the conveniences provided by the Transmission Control Protocol (TCP). The assumptions made on underlying network protocols heavily impact the design of distributed systems. MAGπ can be configured accordingly, based on the network protocols the target source code should operate on. *E.g.* for UDP-based programs one may configure MAGπ with *total message reordering*. Contrastively, TCP-based programs may see MAGπ configured to assume *partial message reordering* (messages of a single channel are not scrambled).

4 Type Properties

MAGπ is built on *generalised* MPST [10], hence it inherits all of the same benefits. Notably, instead of syntactically guaranteeing specific properties, programs can be checked against a number of desired properties (deadlock-freedom, termination, liveness, *etc.*) *a posteriori*. Additionally, the two main results which follow from our type system are *failure-handling safety*, guaranteeing that *all* failure-prone communication has defined failure-handling branches (timeout branches); and *reliability adherence*, ensuring that no redundant timeouts are defined.

The significance of this work shines when combining the benefits of generalised MPST with our result of *failure-handling safety*. That is to say that if a program is, *e.g.*, deadlock-free, then all of the failure-handling timeout branches preserve this deadlock-freedom. This holds for all verifiable properties.

5 Conclusion and Future Work

To conclude, we presented MAGπ—a Multiparty, Asynchronous and Generalised π-calculus addressing the widest set of non-Byzantine faults by using timeouts and a novel viewpoint-specific definition of reliability. Our language builds on *generalised* and *asynchronous* MPST, and has configurable network assumptions. As future work, we wish to develop a model checking tool to verify a program's adherence to a desired list of properties. We also aim to investigate linear logic for Curry-Howard correspondences to understand the canonical meaning of faults and reliability. Lastly, our ambition is extend this work into the realm of Byzantine faults in combination with the non-Byzantine addressed thus far.

References

[1] Manuel Adameit, Kirstin Peters & Uwe Nestmann (2017): *Session Types for Link Failures*. In Ahmed Bouajjani & Alexandra Silva, editors: *Formal Techniques for Distributed Objects, Components, and Systems - 37th IFIP WG 6.1 International Conference, FORTE 2017, Held as Part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, Neuchâtel, Switzerland, June 19-22, 2017, Proceedings*, *Lecture Notes in Computer Science* 10321, Springer, pp. 1–16, doi:10.1007/978-3-319-60225-7_1.

[2] Adam D. Barwell, Alceste Scalas, Nobuko Yoshida & Fangyi Zhou (2022): *Generalised Multiparty Session Types with Crash-Stop Failures*. In Bartek Klin, Slawomir Lasota & Anca Muscholl, editors: *33rd International Conference on Concurrency Theory, CONCUR 2022, September 12-16, 2022, Warsaw, Poland*, *LIPIcs* 243, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 35:1–35:25, doi:10.4230/LIPIcs.CONCUR.2022.35.

[3] Sara Capecchi, Elena Giachino & Nobuko Yoshida (2016): *Global escape in multiparty sessions*. *Math. Struct. Comput. Sci.* 26(2), pp. 156–205, doi:10.1017/S0960129514000164.

[4] Tzu-Chun Chen, Malte Viering, Andi Bejleri, Lukasz Ziarek & Patrick Eugster (2016): *A Type Theory for Robust Failure Handling in Distributed Systems*. In Elvira Albert & Ivan Lanese, editors: *Formal Techniques for Distributed Objects, Components, and Systems - 36th IFIP WG 6.1 International Conference, FORTE 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*, *Lecture Notes in Computer Science* 9688, Springer, pp. 96–113, doi:10.1007/978-3-319-39570-8_7.

[5] Simon Fowler, Sam Lindley, J. Garrett Morris & Sára Decova (2019): *Exceptional asynchronous session types: session types without tiers*. *Proc. ACM Program. Lang.* 3(POPL), pp. 28:1–28:29, doi:10.1145/3290341.

[6] Kohei Honda, Nobuko Yoshida & Marco Carbone (2016): *Multiparty Asynchronous Session Types*. *J. ACM* 63(1), pp. 9:1–9:67, doi:10.1145/2827695.

[7] Matthew Alan Le Brun & Ornela Dardha (2023): *MAGπ: Types for Failure-Prone Communication*, doi:10.48550/ARXIV.2301.10827. Available at `https://arxiv.org/abs/2301.10827`.

[8] Rupak Majumdar, Madhavan Mukund, Felix Stutz & Damien Zufferey (2021): *Generalising Projection in Asynchronous Multiparty Session Types*. In Serge Haddad & Daniele Varacca, editors: *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, *LIPIcs* 203, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 35:1–35:24, doi:10.4230/LIPIcs.CONCUR.2021.35.

[9] Dimitris Mostrous & Vasco T. Vasconcelos (2018): *Affine Sessions*. *Log. Methods Comput. Sci.* 14(4), doi:10.23638/LMCS-14(4:14)2018.

[10] Alceste Scalas & Nobuko Yoshida (2019): *Less is more: multiparty session types revisited*. *Proc. ACM Program. Lang.* 3(POPL), pp. 30:1–30:29, doi:10.1145/3290343.

[11] Malte Viering, Tzu-Chun Chen, Patrick Eugster, Raymond Hu & Lukasz Ziarek (2018): *A Typing Discipline for Statically Verified Crash Failure Handling in Distributed Systems*. In Amal Ahmed, editor: *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, *Lecture Notes in Computer Science* 10801, Springer, pp. 799–826, doi:10.1007/978-3-319-89884-1_28.

[12] Malte Viering, Raymond Hu, Patrick Eugster & Lukasz Ziarek (2021): *A multiparty session typing discipline for fault-tolerant event-driven distributed programming*. *Proc. ACM Program. Lang.* 5(OOPSLA), pp. 1–30, doi:10.1145/3485501.

A Ping

To demonstrate the expressiveness of MAGπ we present a session-typed encoding of the Ping utility from the Internet Control Message Protocol (ICMP). The ping utility consists of a total of three *roles* (participants) communicating amongst each other: two roles, p and r, communicate *reliably* with each other, and both communicate *unreliably* with a third role q.

Our definition of reliability takes into account the viewpoint of each role, thus allowing roles to have their own (possibly empty) *reliability set*. Following the assumptions above, the reliability set for p is $\{\mathsf{r}\}$, for r is $\{\mathsf{p}\}$, and for q is $\emptyset$.

Below we give the session types, denoted $\mathbb{S}_{\mathsf{r}}$, $\mathbb{S}_{\mathsf{p}}$ and $\mathbb{S}_{\mathsf{q}}$ for roles r, p and q respectively, for the 3-attempt Ping utility.

$$\mathbb{S}_{\mathsf{r}} = \&\{\mathsf{p}\,?\,\mathsf{ok}().\mathbf{end},\ \mathsf{p}\,?\,\mathsf{ko}().\mathbf{end}\}$$

$$\mathbb{S}_{\mathsf{p}} = \mathsf{q}\,!\,\mathsf{ping}().\,\&\begin{cases}\mathsf{q}\,?\,\mathsf{pong}().\mathsf{r}\,!\,\mathsf{ok}().\mathbf{end},\\ \text{⏲}.\,\mathsf{q}\,!\,\mathsf{ping}().\,\&\begin{cases}\mathsf{q}\,?\,\mathsf{pong}().\mathsf{r}\,!\,\mathsf{ok}().\mathbf{end},\\ \text{⏲}.\,\mathsf{q}\,!\,\mathsf{ping}().\,\&\begin{cases}\mathsf{q}\,?\,\mathsf{pong}().\mathsf{r}\,!\,\mathsf{ok}().\mathbf{end},\\ \text{⏲}.\,\mathsf{r}\,!\,\mathsf{ko}().\mathbf{end}\end{cases}\end{cases}\end{cases}$$

$$\mathbb{S}_{\mathsf{q}} = \&\begin{cases}\mathsf{p}\,?\,\mathsf{ping}().\mathsf{p}\,!\,\mathsf{pong}().\mathbf{end},\\ \text{⏲}.\ \&\begin{cases}\mathsf{p}\,?\,\mathsf{ping}().\mathsf{p}\,!\,\mathsf{pong}().\mathbf{end},\\ \text{⏲}.\ \&\begin{cases}\mathsf{p}\,?\,\mathsf{ping}().\mathsf{p}\,!\,\mathsf{pong}().\mathbf{end},\\ \text{⏲}.\mathbf{end}\end{cases}\end{cases}\end{cases}$$

Role r is the *receiver* (&–called *branching*), which waits on two options: it receives from p either the label ok or ko and then it terminates the protocol ($\mathbf{end}$). Role p is the *sender* ($\oplus$[1]–called *selection*), and it tries to obtain information on the status of q. It begins by sending a ping message to q ($\mathsf{q}\,!\,\mathsf{ping}()$), then waits to receive from q. If a pong is received ($\mathsf{q}\,?\,\mathsf{pong}()$) in the top branch, then it concludes that the status of q is *reachable* and sends this information to r ($\mathsf{r}\,!\,\mathsf{ok}()$), after which it terminates. Alternatively, p enters a *timeout branch* (⏲). For simplicity, we assume p will attempt to communicate with q three times (shown in the three-time indentation of the timeout branch) before assuming q is *unreachable*; after which the session will terminate by sending ko to r, followed by $\mathbf{end}$. In the same lines, the protocol for role q is given by the session type $\mathbb{S}_{\mathsf{q}}$, where its timeout branches match the timeouts from $\mathbb{S}_{\mathsf{p}}$.

[1] For readability, we adopt a shorthand notation for sending towards a single role and for payloads of type unit, such that $\oplus\{\mathsf{s}\,!\,\mathtt{m}(\mathtt{unit}).\mathbb{S}\}$ is represented by $\mathsf{s}\,!\,\mathtt{m}().\mathbb{S}$.