The Australian National University
2600 ACT | Canberra | Australia

Australian National University

School of Computing

College of Engineering, Computing and Cybernetics (CECC)

Enriching a Verified Choreographic Language with a Simply Typed Lambda Calculus

— Honours project (S1/S2 2024)

A thesis submitted for the degree
Bachelor of Advanced Computing (Research and Development)

By:
Xin Lu

Supervisor:
Dr. Michael Norrish

October 2024

Declaration:

I declare that this work:

- upholds the principles of academic integrity, as defined in the University Academic Misconduct Rules;
- is original, except where collaboration (for example group work) has been authorised in writing by the course convener in the class summary and/or Wattle site;
- is produced for the purposes of this assessment task and has not been submitted for assessment in any other context, except where authorised in writing by the course convener;
- gives appropriate acknowledgement of the ideas, scholarship and intellectual property of others insofar as these have been used;
- in no part involves copying, cheating, collusion, fabrication, plagiarism or recycling.

October, Xin Lu

Acknowledgements

If you wish to do so, you can include some Acknowledgements here. If you don't want to, just comment out the line where this file is included.

There is absolutely no need to write an Acknowledgement section, so only do so when you *want* to – it's always important to stay sincere. One reason for including an acknowledgement could be to thank your supervisor for extraordinary supervision (or any other reason you deem noteworthy). Some supervisors sacrifice a lot, e.g., are always available, meet on weekends, provide multiple rounds of corrections for theses reports, or the like (keep in mind that writing a thesis is special for you, but not for them, so they do actually not have any reason to sacrifice their private time for this!). Seeing acknowledgements in this report can feel like a nice appreciation of this voluntary effort. For large works that form the end of some studies (like an Honours or Master thesis), it is also not uncommon to read acknowledgements to one's parents or partner. But again, completely optional!

Abstract

- choreography diagram; CC, Kalas - the meaning of deadlock freedom by design - mostly focus on interactions via message passing - contribution 1: richerLang: call by value, functional big step semantics with clock to be implemented in HOL4, richer data types, environment semantics; an environmental language model with type theory; and strong normalisation property - contribution 2: the enriched choreography, with a simple type theory, Kalas have the safety property of deadlock freedom

Table of Contents

Chapter 1

Introduction

Distributed systems consist of multiple endpoints that communicate by exchanging messages, operating with asynchrony and parallelism as these messages are sent and received between the various endpoints. But programming distributed system is notoriously error-prone as programmer has to implement the communication protocol by developing individual endpoint programs. Mismatched message sending and receiving can lead to errors such as *deadlock*, where the system is waiting forever for a message.

Choreography arises as a programming diagram to address this issue by providing a concrete global description of how the messages are exchanged between endpoints in a distributed system. A choreography program is written in a similar style to the "Alice and Bob" notation by Needham and Schroeder (1978):

1. $\texttt{Alice} \rightarrow \texttt{Bob} : \textit{key}$
2. $\texttt{Bob} \rightarrow \texttt{Alice} : \textit{message}$

Thus message mismatches are disallowed from the choreographic perspective. A property we refer to as *deadlock free by design*. The global choreography is then projected into process models for each endpoint via EndPoint Projection (EPP), with properties such as deadlock free by design preserved (Hallal et al., 2018).

While most choreography languages focus on the message exchange behaviours, few pay attention to the local computation happening in the individual endpoint. It is shown by Hirsch and Garg (2022) that as long as the local language exhibits type preservation and progress, the choreography inherits these properties as well. Thus, when analyzing message exchange behaviors in choreography—such as multiparty sessions, asynchrony, and parallelism—one can safely assume good behaviours of local computation (Montesi and Yoshida, 2013; Cruz-Filipe and Montesi, 2017; Carbone and Montesi, 2013). But when it comes to writing the choreography program that implements concrete system behaviours, if local computation is ever required by the system, one must describe the inputs and outputs of local computations. Current work either delegate this part to an assumed well-behaved external implementation, for example, Kalas Pohjola et al. (2022) and Pirouette Hirsch and Garg (2022), or provide a basic framework where only natural numbers are considered, as in Core Choreography (CC) Cruz-Filipe and Montesi (2017). This makes writing choregraphy program with local computation implementation an unpleasant experience. For instance, a choreography program in Kalas, where the client performs local computation using input from the server and then sends the result back, would look like:

$$\begin{array}{ll} \texttt{1.} & \texttt{server}.\mathit{var} \twoheadrightarrow \texttt{client}.x; \\ \texttt{2.} & \textbf{let } v@\texttt{client} = \texttt{fun}(x) \textbf{ in} \\ \texttt{3.} & \texttt{client}.v \twoheadrightarrow \texttt{server}.\mathit{result}; \end{array}$$

In Kalas, processes communicate exclusively via strings. Therefore, in the external implementation of $\texttt{fun}(x)$, the string value of x must first be converted to the desired data type for computation. Afterward, the result is converted back to a string before updating the client process variable v.

Thus this thesis extends Kalas, a verified choregraphy language with machine-checked end-to-end compilation, with a simple language of expressions over types such as integers, strings and booleans. We ensure that the extended Kalas has important properties such as type preservation and progress. Using the extended Kalas, the previous choregraphy where client computes the factorial of input integer can be written as:

$$\begin{array}{ll} \texttt{1.} & \texttt{server}.\mathit{var} \twoheadrightarrow \texttt{client}.x; \\ \texttt{2.} & \textbf{let } v@\texttt{client} = \textbf{case } \texttt{NumOf}x \textbf{ of in} \\ \texttt{3.} & \texttt{client}.v \twoheadrightarrow \texttt{server}.\mathit{result}; \\ \texttt{4.} & \texttt{client}.v \twoheadrightarrow \texttt{server}.\mathit{result}; \\ \texttt{5.} & \texttt{client}.v \twoheadrightarrow \texttt{server}.\mathit{result}; \end{array}$$

This thesis provides three main contributions.

- The first contribution is an environment language with functional big step semantics. We also provide a typing system. The language is implemented using the HOL4 proof assistant (Slind and Norrish, 2008). We give type soundness proof for the proposed semantics and typing rules.
- The second contribution is the strong normalisation proof for our environment language model. It is essentially the strong normalisation proof for a simply-typed lambda calculus, but we define the logic relation based on environment semantics.
- The third contribution is the enriched Kalas. Besides common data types such as integer, string, and boolean, we also add function, pair, and sum types to the choregraphy. Common operators for our data types are included, such as addition, modulo, and negation. Integer-string convertor is implemented as well for the integration with Kalas. Last but not least, we prove the enriched Kalas exhibits type preservation and progress.

Chapter 2

Background

2.1 Choreography as a Programming Diagram

- why choregraphy is desired - how choregraphy can be applied to a real example - deadlock free by design and the idea of EPP - a choregraphy diagram showing the choregraphy and its projections

2.2 Interactive Theorem Proving

- how proofs are done using HOL

2.3 Kalas

- overview including intro to its compiler - semantics - implementation in HOL

Chapter 3

Related Work

some intro ..

3.1 Choreography Models

some intro ..

3.1.1 Typing System for Choreography

Choreography language model can ensure deadlock freedom without using a typing system. The property is typically proven through structural induction on the choreography's semantics, where an applicable rule always enables reduction, or reduction follows by inductive hypothesis (Carbone and Montesi, 2013; Cruz-Filipe and Montesi, 2017; Pohjola et al., 2022). Typing system in this case might still be desired since it can discipline choreography in other ways such as correct protocol implementation.

Channel Choreography (ChC) by Carbone and Montesi (2013) is a rich choregraphy language where a choreography program consists of where a program consists of roles, threads, and sessions that implement communication protocols. Deadlock freedom is guaranteed by its semantics, while the typing system ensures correct protocol implementation by sessions. The typing context includes three components: a service environment Γ, which stores global types for public channels specifying session execution and local expressions (annotated with threads); a thread environment Θ, which tracks the roles of threads in each session; and a session environment Δ, which stores the types of active sessions.

It ensures that a well-typed choreography, with public channels specified by Γ and threads assuming roles in Θ, maintains disciplined sessions governed by Δ. Additionally, runtime typing introduces a delegation environment to handle changes in typing context due to asynchrony or parallelism, ensuring the program adheres to the protocol during execution.

When local computation is involved in the choreography, since we cannot solve the halting problem in semantics by deciding whether the local computation will terminate, a typing system is desired to reassert deadlock freedom. To the best of our knowledge, the closet work to this discussion is Pirouette by Hirsch and Garg (2022). They assume a substitution model with small-step semantics for the local language. Based on a set of admissible typing rules for the sake of providing a reasoning ground, their results show that type preservation and progress in the local language ensure the same for the choreography language. Their progress result aligns with our results, but our local language adopts big-step semantics and thus preservation for choreography requires strong normalisation from local language rather than type preservation.

Pirouette is a higher-order functional choreography language with three value types: local values, local functions (mapping local values to choreographies), and global functions (mapping choreographies to choreographies). This structure enables choreographies to return values, forming the basis of its typing system. On the other hand, since neither Kalas or the enriched Kalas has return values, our typing system mainly checks for the well-formedness of a choreography within the typing environment and if any local computation is involved, we discipline it with its own typing system in a localised typing environment. Pirouette types its local values in a similar projected typing environment, binding variables to types at a specific location.

Another state-of-art functional choregraphy language Chorλ by Cruz-Filipe et al. (2022) uses a different approach than Pirouette, where choreographies are interpreted as terms in λ-calculus. Chorλ assumes local values for communications, without focusing on how they are computed. This results in a distinct typing system: local value types are annotated with roles and are part of the global types, rather than being projected from a global environment. Type preservation and progress follow from Chorλ's typing and semantic rules.

3.1.2 Implementation and Handling Exceptions

- Choral: an object-oriented choreography implemented as a java library (Giallorenzo et al., 2024). It treats choregraphy as class and EPP as generating role classes from the choregraphy implementation. It also offloads the local computation to Java. Higher-order functional choreography models such as Pirouette and Chorλ can be viewed as formal model candidates for it. Its type checker is similar to the typing system in ChC where it has types for public channels where roles for sender and receiver are specified as well as the type of messages being communicated an it also has local types annotated with roles.

In terms of communication failure which may raise an exception when a role is trying to perform operation based on reading from the place where a received message has not arrived yet, Choral implements the failure model in RC by Montesi and Peressotti

(2017), using recover strategies such as capped attempts or timer within java try-catch block.

- RC, a model where communication failure is considered, recover strategy for senders and receivers, either while loop until received (exactly once delivery for setting 1), or with a timer or capped tries (best efforts); typing system to ensure almost once delivery (and exactly one delivery where message won't be lost); they use configuration where sender and receiver have stacks and it is initially false, with payload value, or ticked meaning no longer in the stack (sent/received) to implement send/received and the failure rules for semantics and typings; we do not consider message sending failure, but we do have exception caused by local computation failure (e.g. division by zero) or bad message value type (not a string), and we record the exception using transition labels. But we do not consider any recover strategy and always transits the choregraphy into a termination state (nil)

3.2 Some Title

- functional big-step semantics

- SN for STLC; strong normalisation for languages of environmental semantics

Chapter 4

The Environment Model

4.1 Syntax

- binary operators - unary operators (StrOf, NumOf) - value script?

4.2 Semantics

- functional big step semantics: intepreter style with clock, total function. for being implemented in HOL4 - properties: clock increment (cases on result) (and clock decrement) - closure: issues with dynamic environment, so we use lexical environment; do we explain restrciting to fv(e) and how? - exceptions

4.3 Typing

4.3.1 Syntax

- rich data types: int, string, boolean, sum type, pair type, closure, ... - typecheck - uoptype, boptype - valuetype - envtype

4.3.2 Typing Rules

- typecheck - uoptype, boptype - valuetype: closure - no type uniqueness

4.3.3 Main Properties

- value invertability; envtype used for fnT case of it - operators: uoptype soundness and boptype soundness (use the invertability) - typecheck: reducing typing environment (for soundness fn case)

4.3.4 Type Soundness

Chapter 5

Strong Normalisation

Chapter 6

The Enriched Choreography

6.1 Syntax

6.2 Semantics

- exceptions

6.3 Typing

6.3.1 Typing Rules

6.3.2 Main Properties

6.3.3 Type Soundness

Chapter 7

Concluding Remarks

If you wish, you may also name that section *"Conclusion and Future Work"*, though it might not be a perfect choice to have a section named "A & B" if it has subsections "A" and "B". Also note that you don't necessarily have to use these subsections; that also depends on how much content you have in each. (E.g., having a section header might be odd if it contains just three lines.)

7.1 Conclusion

This section usually summarizes the entire paper including the conclusions drawn, e.g., did the developed techniques work? Maybe add why or why not. Also don't hold back on limitations of your work; it shows that you understood what you have done. And science isn't about claiming how great something is, but about objectively testing hypotheses. Also note that every single scientific paper has such a section, so you can check out many examples, preferably at top-tier venues, e.g., by your supervisor(s).

7.2 Future Work

- asynchronus messages; confluence property - Progress for EPP

Chapter 8

Test

We define what it is for a choreograph to be well-formed with the $G\,,\mathit{Th} \vdash c\checkmark$ relation.

This is a theorem:

$$\vdash \emptyset, \Theta \vdash c\checkmark \;\Rightarrow\; \exists\, \tau\ l\ s'\ c'.\ \emptyset \triangleright c \ \underset{l}{\overset{\tau}{\rightarrow}}\ s' \triangleright c' \ \vee\ \neg \texttt{not\_finish}\ c$$

The transition relation looks like `eval_exp` $\mathit{clk}\ E\ \mathit{exp}$

Theorem 8.0.1. *some text here*

1. *(Operational completeness) If* $G\,,\mathit{Th} \vdash c\checkmark$ *then there exist . . .*
2. *(Operational soundness) If* `eval_exp` $\mathit{clk}\ E\ \mathit{exp}$ *then there exist . . .*

Definition 1 (Kalas syntax)**.** *Choreographies* in Kalas, ranged over by C, are inductively defined by the grammar

Appendix A

Appendix: Explanation on Appendices

You may use appendices to provide additional information that is in principle relevant to your work, though you don't want *every reader* to look at the entire material, but only those interested.

There are many cases where an appendix may make sense. For example:

- You developed various variants of some algorithm, but you only describe one of them in the main body, since the different variants are not that different.
- You may have conducted an extensive empirical analysis, yet you don't want to provide *all* results. So you focus on the most relevant results in the main body of your work to get the message across. Yet you present the remaining and complete results here for the more interested reader.
- You developed a model of some sort. In your work, you explained an excerpt of the model. You also used mathematical syntax for this. Here, you can (if you wish) provide the actual model as you provided it in probably some textfile. Note that you don't have to do this, as artifacts can be submitted separately. Consult your supervisor in such a case.
- You could also provide a list of figures and/or list of tables in here (via the commands `\listoffigures` and `\listoftables`, respectively). Do this only if you think that this is beneficial for your work. If you want to include it, you can of course also provide it right after the table of contents. You might want to make this dependent on how many people you think are interested in this.

Appendix B

Appendix: Explanation on Page Borders

What you find here is an explanation of why the border width keeps flipping from left to right – which you might have spotted and wondered why that's the case.

Firstly, that is *intended* and thus correct, so there is no reason to worry about this. The reason is that this document is configured as a two-sided book, which means:

- We assume the document will be printed out,
- that this will be done in a two-sided mode (i.e., the document will be printed on both sides of each page), and
- that the bookbinding will be in the middle, just like in every book.

When you open the book, there are three borders of equal size n. This however requires that even pages have a border of n on their left and $\frac{n}{2}$ on their right, and odd pages have a border of $\frac{n}{2}$ on their left and n on their right. This is illustrated in Figure B.1.

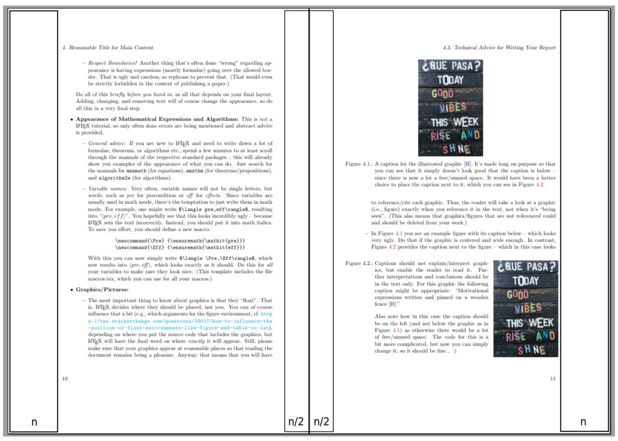

Figure B.1: Illustration showing why page borders flip.

Bibliography

Carbone, M. and Montesi, F., 2013. Deadlock-freedom-by-design: multiparty asynchronous global programming. 48, 1 (2013), 263–274. doi:10.1145/2480359.2429101. https://dl.acm.org/doi/10.1145/2480359.2429101. [Cited on pages 1 and 5.]

Cruz-Filipe, L.; Graversen, E.; Lugović, L.; Montesi, F.; and Peressotti, M., 2022. Functional choreographic programming. In *International Colloquium on Theoretical Aspects of Computing*, 212–237. Springer. [Cited on page 6.]

Cruz-Filipe, L. and Montesi, F., 2017. A core model for choreographic programming. In *Formal Aspects of Component Software: 13th International Conference, FACS 2016, Besançon, France, October 19-21, 2016, Revised Selected Papers 13*, 17–35. Springer. [Cited on pages 1, 2, and 5.]

Giallorenzo, S.; Montesi, F.; and Peressotti, M., 2024. Choral: Object-oriented choreographic programming. *ACM Transactions on Programming Languages and Systems*, 46, 1 (2024), 1–59. [Cited on page 6.]

Hallal, R.; Jaber, M.; and Abdallah, R., 2018. From global choreography to efficient distributed implementation. In *2018 International Conference on High Performance Computing & Simulation (HPCS)*, 756–763. doi:10.1109/HPCS.2018.00122. https://ieeexplore.ieee.org/abstract/document/8514427?casa_token=B9uMnW0mxFEAAAAA:DMmhwgQZJnHAX6o6p-EHBs4K9rct4pEKen9fdt2CXHC6NOWZxpHT5FSZZFAchGBDjiqmPeviH1U. [Cited on page 1.]

Hirsch, A. K. and Garg, D., 2022. Pirouette: higher-order typed functional choreographies. 6 (2022), 23:1–23:27. doi:10.1145/3498684. https://dl.acm.org/doi/10.1145/3498684. [Cited on pages 1, 2, and 6.]

Montesi, F. and Peressotti, M., 2017. Choreographies meet communication failures. *arXiv preprint arXiv:1712.05465*, (2017). [Cited on page 6.]

Montesi, F. and Yoshida, N., 2013. Compositional choreographies. In *CONCUR 2013 – Concurrency Theory* (Berlin, Heidelberg, 2013), 425–439. Springer. doi:10.1007/978-3-642-40184-8_30. [Cited on page 1.]

Needham, R. M. and Schroeder, M. D., 1978. Using encryption for authentication in large networks of computers. 21, 12 (1978), 993–999. doi:10.1145/359657.359659. https://dl.acm.org/doi/10.1145/359657.359659. [Cited on page 1.]

Pohjola, J. Å.; Gómez-Londoño, A.; Shaker, J.; and Norrish, M., 2022. Kalas: A verified, end-to-end compiler for a choreographic language. In *13th International Conference on Interactive Theorem Proving (ITP 2022)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik. [Cited on pages 2 and 5.]

Slind, K. and Norrish, M., 2008. A brief overview of hol4. In *International Conference on Theorem Proving in Higher Order Logics*, 28–32. Springer. [Cited on page 2.]