Session-Typed Concurrency

Elizabeth Sidebottom

Executive Summary

Concurrent programming is becoming increasingly popular and with many ways to achieve concurrency it can be difficult to choose which language or model to use as they all come with benefits and flaws. A simple way to achieve concurrency is through shared memory where different threads have access to the same variables. This can result in race conditions as threads may edit shared variables thus impacting the next thread which needs to use it which can cause a program to be non-deterministic. Another way to achieve concurrency is through message passing where different processes use channels to communicate and send data. However, there is often no way to determine how each process should use a channel so two processes may get stuck waiting for the other to send some information.

We propose using session types to type-check a message passing concurrent system. This will prevent endless waiting as sessions dictate how processes should use a channel to communicate. Our solution will use linear logic as a basis for session types to provide a type-safe concurrent programming language. The only concurrent language (that we know of) which has its basis in linear logic and session types is Concurrent C0[1] which is used as a teaching language for students learning concurrent programming. We can use Concurrent C0 as a starting point to create a new language which includes everything one would expect from an industry viable concurrent language. We will collaborate with the creators of Concurrent C0 and a group of PhD students to create a well-typed concurrent language using linear logic and session types.

Evaluation of Potential Opportunity

Concurrency is built for efficiency, but often the complexity of concurrent languages, or the inability to specify certain behaviours makes concurrent programming a long process full of debugging. A Microsoft survey[2] found that 66% of respondents had issues with concurrency and 65% of respondents who use concurrency thought that these issues are going to become more problematic in the future. It was also found that most often it took days to fix a single bug, and problems tended to be very severe. Looking forward, most respondents said that better tools, compilers, and programming languages would be strongly beneficial to them.

Type-checking is a very important part of programming and can increase both efficiency and code quality[3]. Errors can be noticed at compile-time rather than at run-time so any errors present in a type-checked program will be noticed faster and hopefully corrected by the programmer, which improves both productivity (time taken to complete a specific task), and quality of code. A well-typed language will have the potential to be type-checked and therefore provide a better programming experience.

Having a type-safe language is highly important when learning to code,[4,5] as it helps programmers fix mistakes easily. This can both build confidence as errors can be found without external help and aid in faster learning as progress is not hindered by excessive time taken to find mistakes. There is also something to be said for learning from any mistakes that are made as correcting them can help improve understanding of how the language works.

As computers are becomming more powerful, the opportunity for concurrency grows. Concurrent programs have the ability to be more efficient than ever as multi-core processors can handle many tasks at once with growing ease. This makes now the perfect time to create a concurrent language which enables programmers to produce quality concurrent programs in an efficient manner.

Value Proposition

Our proposition is to create a concurrent programming language which uses session types to enhance a message passing model of concurrency. This will give programmers an efficient way to produce quality concurrent programs in a way that is both accessible for beginners and powerful for experienced concurrent programmers.

We will use session types to dictate how processes should communicate over channels. This will allow for bidirectional communication which is currently very difficult in other concurrent languages. Further, it will also ensure session fidelity which guarantees that processes both send and recieve the correct data in the correct order as dictated by the session type.

A session is a communication between two processes so a session type describes what each process should be doing. So, if we have two processes A and B and a channel x we can use session types to ensure that first A sends some information along x which B will then receive then B can send some information to A, or any other sequence of interactions which may be requied. Furthermore, session-types are linear so if a program has access to a channel, it cannot delete the channel until the session is finished. So we do not have the problem of a program waiting for communication forever.

Our programming language will be invaluble for both beginners and experienced programmers as we will use session types to give channels types and therefore prescribe exactly how they should be used by processes. This along with compile-time type-checking will make our language easy to use and debug which improves efficiency and helps beginners learn faster.

As previously mentioned, the only other programming language we know of which uses linear logic and session types is Concurrent C0. This is used at Carnegie Mellon University to teach beginner programmers how to program concurrently. While this is a great start as it is a working language capable of producing concurrent programs, there are a few downfalls. As Concurrent C0 is a teaching language, it is designed to be simple. For example, there is only one numerical type, namely integers, which while easy to understand for beginners will be frustrating for more experienced programmers. It is highly unlikely that only integers will be used in a program and not having other options like decimals and floats makes this language useless for industry applications.

We will take into account the needs of both industry and individuals to create a language with the power to create programs on a large scale with multiple programmers and the simplicity to be a reasonable first concurrent language a student can learn.

Impact Plan

This project will be in collaboration with the creators of Concurrent C0[6] and we aim to recruit 4 PhD students and one postdoctoral researcher to assist with this research. Microsoft Research

have agreed to put £300,000 towards this project so we will need an additional grant of £200,000 to cover salaries and PhD stipends. This will also cover any equipment costs such as personal computers, standard maintenance, and travel to relevant summer schools and conferences.

Our plan is that this project will take 3 years, but depending on development we will allow for 4 years if needed. We will also be recording our research throughtout the process in hopes to submit a paper to some relevant conferences such as Principles of Programming Languages[7] (POPL) and Programming Language Design and Implementation[8] (PLDI). We may also present our language to participants of Oregon Programming Languages Summer School[9] (OPLSS).

The initial research behind our proposed programming language is currently being undertaken by a masters student at the University of Bristol. This covers the logic behind the language and will be vital in making our language work smoothly. Two of our PhD students will continue this research for 6 months at the beginning of our project to ensure we have everything we need to create our programming language. We will also be carrying out surveys to find out what experienced programmers desire from a concurrent programming language to guide us in our efforts. This will go alongside discussions with Microsoft about what they believe will be the most important requirements for an industrially viable language. We will also be working with the creators of Concurrent C0 to find out what the students they have taught think of their language as it stands to find out how we can best cater for both beginner and experienced programmers. This initial stage of research will determine how we create our programming language and what features to include.

Within this initial stage, our postdoctoral researcher will be looking into compilers so our type-checking system can be efficient and easy to use. We may decide that the best way forwards is to create our own compiler so this research must be done at the start of our project. We will also be considering whether to implement a type inference system so users do not have to specify types whilst programming. This would be a very useful tool as it would make the language less complex and improve efficiency.

We will then begin building our language whilst taking into consideration all the information we gained in our initial research stage. This will involve implementing our session-typed linear logic system to create a message passing concurrent language. We will continually test new additions to ensure our language is type safe at every stage of the process so if there are problems we only have to debug a relatively small section of code. If we decide to make our own compiler, this will be happening in tandem with our language building as we can check how the compiler handles each new aspect of our language and edit accordingly.

As we are building our language, we will be continually testing to ensure it is easy to use and does what we expect it to. This is of utmost importance as we are looking to create a language which is accessible to beginners and advanced programmers alike. We must make certain that our language is type-safe at every stage of development and that our type-checking system works exactly as intended.

Our aim is to have our language be useable after one year of development, then we can start writing programs with it to see where it needs improvement. We will conduct more surveys to determine what users think of the language and what needs work. We plan to use this time to start work on a type inference system so we can improve our language whilst gaining feedback.

After acquiring feedback from users we can begin the next stage of development and improve our language. We will use feedback to implement any changes required to make the language

more user-friendly and efficient. Any additional problems detected by users will also be fixed at this stage of development. After an additional 6 months of development we hope to have a language which is ready for presentation at conferences and summer schools. This will also give us the opportunity to find more industry partners and potentially gain additional investors as well as find more users willing to test our language and give feedback. Further, teaching at summer schools will give us an indication as to whether our language will be a suitable language for students to learn as thier first dive into concurrency.

The remainder of the time allocated for this project will be used for more in depth user-testing to ensure our language is user-friendly and efficient. We plan to promote our language at conferences and summer schools to attract a user-base who are willing to help test our language in this final stage of development. At this stage we will also be working closely with Microsoft and any other industrial partners to make sure their needs for a concurrent language are met since they will determine most heavily what is required for an industry-viable programming language.

References

[1]R. Arnold, "C0, an imperative programming language for novice computer scientists", PhD thesis (Master's thesis, Department of Computer Science, Carnegie Mellon University, 2010).

[2]P. Godefroid and N. Nagappan, "Concurrency at microsoft: an exploratory survey", in Cav workshop on exploiting concurrency efficiently and correctly (Princeton, USA, 2008).

[3]L. Prechelt and W. F. Tichy, "A controlled experiment to assess the benefits of procedure argument type checking", IEEE Transactions on Software Engineering **24**, 302–312 (1998).

[4]M. Kölling, "The problem of teaching object-oriented programming, part 1: languages", Journal of Object-oriented programming **11**, 8–15 (1999).

[5]K. Howell, "First computer languages", Journal of Computing Sciences in Colleges **18**, 317–331 (2003).

[6]M. Willsey, R. Prabhu, and F. Pfenning, "Design and implementation of concurrent c0", arXiv preprint arXiv:1701.04929 (2017).

[7]*Principles of programming languages (popl)*, `https://www.sigplan.org/Conferences/POPL/`.

[8]*Programming language design and implementation (pldi)*, `https://www.sigplan.org/Conferences/PLDI/`.

[9]*Oregon programming languages summer school*, `https://www.cs.uoregon.edu/research/summerschool/archives.html`.