River: a functional reversible language

Jen-Shin Lin

December 12, 2016

From our previous researches and studies, we did get some idea of how can we start we developing on functional reversible language. On the one hand, we confirm that indeed linear logic and linear types cannot be directly applied here. However the way of how it explicitly presents and manages information, variables and their flow is a very helpful guide for us. On the other hand the syntax created from reverible logic lacks some essential mechanism or operator such that it can be use pratically and easily. Therefore we combine several techniques and notions so far we know from (i) linear logic and types and its term-assignment [1, 9, 2, 6, 4, 8, 5] (ii) previous works on reversibility [11, 7, 3, 12] , and come up with a new functional language with reversibility, named *river*, which features:

pattern calculus ...

matchable ...

linear context ...

combinator ...

restricted function declaration ...

global function register ...

operational semantics ...

syntactic foundation for dup-eq ...

well-designed syntax ...

The useful techs to go

[1] *view-pattern*$^?$: [2] *de bruijn index*: [3] *locally nameless*: [4] *hybrid context*: [5] *explicit variables management*: [6] *dependent type*: (i) *a*: ... (ii) *b*: ...

Linear logic [4], firstly developed in classical sequent calculus, features that: *any information (truth) cannot be created or discarded arbitrarily.* It also provides an explicit way to formally describe temporal and spatial resources. To see that, as mentioned in Novitzká's work [5], linear logic can realize both of positive and negative data types due to its lack of two important structural rules: *contraction* and *weakening.*

From CurryHoward correspondence, we know that there must be a type system for linear logic that can format the usage of resources and even the information-flow. This give us an inspiration: it may be possible to use linear type for modeling information-flow, and with that one can determine a trace of computation which can show us how to compute backwards. The rest of this article will show how to develop a type system from linear logic and what those type systems can do and is that possible to develop a reversible langauage from it.

Reversible communication

Starting by directly creating typing rules from linear logic, immediately we hit a wall: we don't know how to assign term(s) to those rules. For example, what a λ-term will be for the following rule?

$$\frac{\vdash \Gamma, A \qquad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\ (\otimes\text{-I})$$

The thing is, in classical sequent calculus, there may be more than one consequent on the right-hand-side of a judgment. This is not a scenario that λ-calculus can fit or apply, but, π*-calculus* do. The type system developed directly from classical linear logic is known as *session type* [10]. It have the capability to describe the flow of information or resources involved within the communication between two or even more processes. Obviously this

can be used to model distributed system or parallelism.

In distributed system, reversibility leads to a very practical topic named *reversible distributed system*. Reversible distributed system have the ability to abort and unwind a computation with states recovery. Brown and Sabry [3] have developed a reversible process language with a high-level semantics. This topic and works look very valuable however it is more like a language for reversible communication rather reversible computation. This is not exactly what we were looking for.

Reversible language

As discussed above, the idea of creatnig a type system directly from linear logic turns out not be able to solve our problem. So let's try the conventional method. As plenty works have done [1, 9, 2, 6, 8], it actually not hard to rewrite *classical* linear logic rules into *intuitionistic* one, and the type system we want will reveal itself instantly. This type system is called as *linear type* and it is not too hard to give it a term assigment which will be an extension of λ-calculus.

With the understanding of linear type, lots of researchers put efforts into finding a way to define a well-form reversible programming language. Yokoyama, Axelsen and Glück have developed a reversible functional language based on the technique of pattern maching [11]. Recently they have also developed a reversible imparetive programming language with reversible flowcharts [12]. Spark and Sabry [7], on the other hand, have not used the conventional way. They have developed another logic system from linear logic and discarded some properties those are not helpful for reversible programming. This logic is called reversible logic and they have also defined a simple reversible language, Λ^R , for it.

Conclusion Our goal is to build a reversible functional language. Our original intuition was, linear type could somehow useful and we might be able to discover a reversible type by directly manipulating linear type. In the end, it looks not that simple as we expected. Linear logic give too much power to describe data structures, like the capability to seperate positive and negative types, which is not necessary for reversible computation. The point will be, firstly find some necessary operators and properties from linear type or logic; and secondly, how to use those operators and properties to record or generate a computation path that can directly show us how to compute backwards. Besides finding some new ways to provide reversibility of functional language, there are still several problem waiting us to overcome. For example, recent works were more focusing on providing reversibility via syntax and semantics. which is great. But less works put efforts on applying type system to verify the correctness for reversibility.

References

[1] Benton, N., Bierman, G., and Paiva, V. Term assignment for intuitionistic linear logic. Tech. rep., 1992.

[2] Benton, N., Bierman, G. M., Hyland, J. M. E., and de Paiva, V. A term calculus for intuitionistic linear logic. In *Proceedings of the International Conference on Typed Lambda Calculi and Applications* (1993), M. Bezem and J. F. Groote, Eds., Springer-Verlag LNCS 664, pp. 75–90.

[3] Brown, G., and Sabry, A. Reversible communicating processes. In *Proceedings Eighth International Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software, PLACES 2015, London, UK, 18th April 2015.* (2015), pp. 45–59.

[4] Girard, J.-Y. Linear logic: Its syntax and semantics, 1995.

[5] Novitzká, V., and Mihályi, D. Resource-oriented programming based on linear logic. In *Acta Polytechnica Hungarica* (2007).

[6] Ronchi della Rocca, S., and Roversi, L. Lambda calculus and intuitionistic linear logic. In *Logic Colloquium '94* (Clermont Ferrand, France, 1994).

[7] Sparks, Z., and Sabry, A. Superstructural reversible logic. In *3rd International Workshop on Linearity* (2014).

[8] Turner, D. N., and Wadler, P. Operational interpretations of linear logic, 1998.

[9] Wadler, P. A taste of linear logic. In *Mathematical Foundations of Computer Science* (Gdańsk, Poland, 1993), A. M. Borzyszkowski and S. Sokolowski, Eds., Springer-Verlag LNCS 781, pp. 185–210.

[10] Wadler, P. Propositions as sessions. In *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming* (New York, NY, USA, 2012), ICFP '12, ACM, pp. 273–286.

[11] Yokoyama, T., Axelsen, H. B., and Glück, R. Towards a reversible functional language. In *Proceedings of the Third International Conference on Reversible Computation* (Berlin, Heidelberg, 2012), RC'11, Springer-Verlag, pp. 14–29.

[12] Yokoyama, T., Axelsen, H. B., and Glück, R. Fundamentals of reversible flowchart languages. *Theor. Comput. Sci. 611*, C (Jan. 2016), 87–115.