Introducing a New Project on The Combination of Substructural Logics and Dependent Type Theory

Harley Eades III

Computer Science, Augusta University

Introduction. Over the course of the last thirty years interactive proof assistants based on the correspondence between logic and programming languages have been used to formalize the correctness of several substantial projects, for example, the CompCert project [4] built a fully verified C compiler that has withstood grand attempts at finding bugs within it [8], and seL4 [2] is the first fully verified secure microkernel, and it withstood its own trials when skilled hackers tried to comprise a system running seL4 [7]. These large scale projects show that the use of these types of platforms is important and leads to more correct implementations, but there are major limitations when it comes to the underlying technology of interactive proof assistants. In particular, it is very difficult to formalize the correctness of *resource dependent systems* which require the precise tracking of the use of resources. Substructural logics are resource conscious logics that do precisely track the use of resources, but modern day proof assistants do not provide much support for their use in formalizations, and formalizations that do require the use of substructural logics are forced to use complex encodings [6], and thus, putting their use outside the reach of domain experts.

This talk introduces a new project that proposes a new platform, called Tenli, that integrates substructural logics with dependent types to combine the power of both worlds. In this talk I introduce the initial ideas that will lead to Tenli's development, and give several interesting theoretical and practical directions for future research.

Substructural logics are characterized by which structural rules they allow. In this talk I will consider substructural logics that are determined by combinations of the structural rules of associativity, exchange (symmetry or commutativity), weakening, and contraction. In total, by combining these rules in different ways one obtains sixteen different substructural logics including the substructural logic without any of them. The most popular substructural logics in computer science like Commutative/Non-commutative Linear Logic, Commutative/Non-commutative Affine Logic, Commutative/Non-commutative Relevant Logic (or contraction logic), Associative/Non-associative Lambek Calculus are among the sixteen.

Benton [1] showed that by taking a symmetric monoidal adjunction $\mathcal{I} : F \dashv G : \mathcal{L}$ between a symmetric monoidal closed category (a model of intuitionistic linear logic), $\mathcal{L}$, and a cartesian closed category (a model of intuitionistic logic), $\mathcal{I}$, one obtains a mixed linear/non-linear model capable of producing a logic where both intuitionistic logic and linear logic can be used simultaneously in a very natural way called LNL logic. In fact, contexts indexing sequents of LNL logic contain hypotheses from both logics without any annotations.

Suppose $\mathcal{M}_1$ and $\mathcal{M}_2$ are two categories with a bifunctor $\odot_i : \mathcal{M}_i \times \mathcal{M}_i \longrightarrow \mathcal{M}_i$, a distinguished object $I_i \in \mathsf{Obj}(\mathcal{M}_i)$, and the two natural isomorphisms $\lambda_A : A \odot I_i \longrightarrow A$ and $\rho_A : I_i \odot A \longrightarrow A$. We call these categories *magmoidal categories with a unit*. Examples of magmoidal categories with a unit are monoidal categories, symmetric monoidal categories, and cartesian closed categories. Benton's LNL models can be generalized to magmodial categories. Simply, take an adjunction $\mathcal{M}_1 : F \dashv G : \mathcal{M}_2$ called an *adjoint model* where the functors F and G preserve the magmoidal structure similarly to monoidal functors, but without the coherence diagram for the associator. Now if we add to $\mathcal{M}_1$ some structural rule, for example, by making $\mathcal{M}_1$ a monoidal category, then we obtain an adjoint model that corresponds to a logic where the Lambek calculus ($\mathcal{M}_1$) and the non-associative Lambek calculus ($\mathcal{M}_2$) are mixed similarly to LNL models. We can do this for all the available structural rules, another example would be by taking $\mathcal{M}_2$ to be a symmetric monoidal category, and $\mathcal{M}_1$ to be a symmetric monoidal category with weakening. This adjoint model corresponds to the logic that mixes affine logic ($\mathcal{M}_1$) with linear logic ($\mathcal{M}_2$).

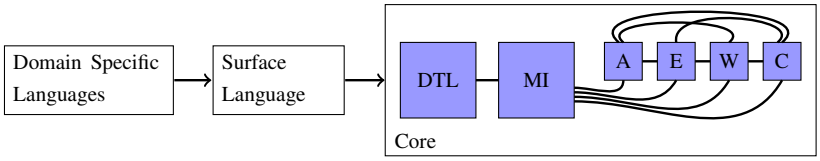
Suppose $\mathcal{M}_1 : F \dashv G : \mathcal{M}_2$ is an adjunction. It is well known that the endofunctor $FG : \mathcal{M}_2 \longrightarrow \mathcal{M}_2$ induces a comonad; here a magmoidal comonad. This comonad pulls all of the structure from $\mathcal{M}_1$ into $\mathcal{M}_2$, but in a controlled way. Perhaps a lesser known result, due to Melliès [5], is that if we have an adjunction $\mathcal{M}_1 : F \dashv G : \mathcal{M}_2 : H \dashv J : \mathcal{M}_3$, then the induced comonad $HFGJ : \mathcal{M}_3 \longrightarrow \mathcal{M}_3$ pulls all of the structure from both $\mathcal{M}_1$ and $\mathcal{M}_2$ into $\mathcal{M}_3$. In fact, we can think of this comonad as the composition of the two comonads $HJ : \mathcal{M}_3 \longrightarrow \mathcal{M}_3$ and $FG : \mathcal{M}_2 \longrightarrow \mathcal{M}_2$, but by using adjunctions the distributive laws needed to compose (co)monads are not necessary. From a logical perspective this gives us a way of combining multiple substructural logics to define new ones that combine structural rules.

Now suppose we have four basic substructural logics we call $\mathcal{A}$ for associativity, $\mathcal{E}$ for exchange, $\mathcal{W}$ for weakening, and $\mathcal{C}$ for contraction; all are essentially magmoidal categories with a single structural rule. Then the following adjoint model corresponds to a logic that can modularly define all sixteen substructural logics discussed above:

$$\begin{array}{llll} \mathrm{C} : F_{\mathrm{CW}} \dashv G_{\mathrm{CW}} : \mathrm{W} & \mathrm{W} : F_{\mathrm{WE}} \dashv G_{\mathrm{WE}} : \mathrm{E} & \mathrm{E} : F_{\mathrm{EA}} \dashv G_{\mathrm{EA}} : \mathrm{A} & \mathrm{A} : F_{\mathrm{AM}} \dashv G_{\mathrm{AM}} : \mathrm{MI} \\ & \mathrm{C} : F_{\mathrm{CE}} \dashv G_{\mathrm{CE}} : \mathrm{E} & \mathrm{W} : F_{\mathrm{WA}} \dashv G_{\mathrm{WA}} : \mathrm{A} & \mathrm{E} : F_{\mathrm{EM}} \dashv G_{\mathrm{EM}} : \mathrm{MI} \\ & & \mathrm{C} : F_{\mathrm{CA}} \dashv G_{\mathrm{CA}} : \mathrm{A} & \mathrm{W} : F_{\mathrm{WM}} \dashv G_{\mathrm{WM}} : \mathrm{MI} \\ & & & \mathrm{C} : F_{\mathrm{CM}} \dashv G_{\mathrm{CM}} : \mathrm{MI} \\ & & & \mathrm{DTL} : F_{\mathrm{DM}} \dashv G_{\mathrm{DM}} : \mathrm{MI} \end{array}$$

The category $\mathcal{M}$ is a magmoidal category with no structural rules, and its job is to bring all the others together so that they can either be used independently, or mixed.

Using the previous model we can now summarize the Tenli design. It consists of the four basic substructural logics, the corresponding logic to magmoidal categories we call MI, and a dependently typed language called DTL in adjunction with MI. We summarize the Tenli design as follows:



In the previous diagram each line corresponds to a syntactic adjunction similarly to Benton's LNL logic. As one might expect the core language is rather complex with lots of syntax for each adjunction. The language DTL will be setup similarly to Krishnaswami et al.'s [3] integration of dependent types with linear logic. The job of the surface language is to make the syntax more programmer friendly, but it will also supply several more features that will make designing domain specific languages easier.

The Tenli design results in a very general language where a number of substructural logics can be used, mixed, and combined with dependent types to formalize both software and systems. Its impact could be substantial. Because of Tenli's generality it is a great candidate to be a system that can be used to develop (embedded) domain specific languages. The project plans to investigate meta-programming with dependent and substructural types and generic programming to make the development of domain specific languages easier.

Conclusion. The Tenli project seeks to develop a proof assistant that integrates substructural logics with dependent types. It is the hope of the project to develop Tenli so that it can be used to develop domain specific languages that make use of dependent types and/or substructural logics to formalize software and systems. After the creation of Tenli we plan to use it to develop a domain specific language for conducting threat analysis using attack trees.

Acknowledgments. The author is supported by the National Science Foundation CRII CISE Research Initiation grant, "CRII:SHF: A New Foundation for Attack Trees Based on Monoidal Categories", under Grant No. 1565557.

References

[1] P. N. Benton. A mixed linear and non-linear logic: Proofs, terms and models (preliminary report). Technical Report UCAM-CL-TR-352, University of Cambridge Computer Laboratory, 1994.

[2] Kevin Elphinstone, Gerwin Klein, Philip Derrin, Timothy Roscoe, and Gernot Heiser. Towards a practical, verified kernel. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Operating Systems*, HOTOS'07, pages 20:1–20:6, Berkeley, CA, USA, 2007. USENIX Association. URL: `http://dl.acm.org/citation.cfm?id=1361397.1361417`.

[3] Neelakantan R. Krishnaswami, Pierre Pradic, and Nick Benton. Integrating linear and dependent types. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 17–30, New York, NY, USA, 2015. ACM.

[4] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, July 2009. URL: `http://doi.acm.org/10.1145/1538788.1538814`, `doi:10.1145/1538788.1538814`.

[5] Paul-André Melliés. Comparing hierarchies of types in models of linear logic. *Information and Computation*, 189(2):202 – 234, 2004.

[6] James F. Power and Caroline Webster. Working with linear logic in coq. In *12th International Conference on Theorem Proving in Higher Order Logics*, pages 14–27. University of Nice, France, September 1999.

[7] New Scientist. Unhackable kernel could keep all computers safe from cyberattack. `https://goo.gl/AL58M9`. Accessed: June 12, 2017.

[8] Xuejun Yang, Yang Chen, Eric Eide, and John Regehr. Finding and understanding bugs in c compilers. *SIGPLAN Not.*, 46(6):283–294, June 2011. URL: `http://doi.acm.org/10.1145/1993316.1993532`, `doi:10.1145/1993316.1993532`.