Intermediate Logic Proofs as Concurrent Programs

Federico Aschieri, Agata Ciabattoni, and Francesco A. Genco

TU Wien
Vienna, Austria

Building on ideas of Haskell Curry, in 1969 William Howard showed that constructing an intuitionistic proof is not at all different from writing a program in λ-calculus [8]. He also showed that the reduction of the proof to its normal form exactly corresponds to the evaluation of the associated program. This relation between intuitionistic natural deduction and simply typed λ-calculus is now called the Curry–Howard correspondence. In 1990 Griffin showed that such a correspondence is not limited to intuitionistic logic but a similar relation holds between classical logic and sequential extensions of simply typed λ-calculus featuring control operators [7]. One year later, in 1991, Avron noticed a connection between concurrent computation and hypersequent calculus – a proof calculus well suited for capturing logics intermediate between intuituionistic and classical logic, see [3]. He envisaged, in particular, the possibility of using the intermediate logics that can be captured by hypersequent calculi "as bases for parallel λ-calculi" [4].

The translation in [5] from hypersequent rules into higher-level natural deduction rules [9] made it possible to define natural deduction calculi matching the parallel structure of hypersequents. Building on this, we establish modular Curry–Howard correspondences for a family of natural deduction calculi and we prove their normalization. These correspondences provide a concurrent computational interpretation for intermediate logics that are naturally formalized as hypersequent calculi. The calculi resulting from this computational interpretation are extensions of the simply typed λ-calculus by a parallelism operator and communication channel variables. We thus confirm Avron's 1991 thesis for a rather general class of intermediate logics and present some specific instances of particular proof-theoretical interest.

In particular, we first introduce the typed concurrent λ-calculi λ_{Cl} [2] and λ_{G} [1]. These calculi are defined extending simply typed λ-calculus by the type assignment rules

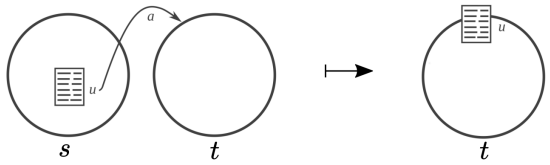
$$\dfrac{\begin{array}{cc}[a:\neg A] & [a:A] \\ \vdots & \vdots \\ s:B & t:B\end{array}}{s \parallel_a t : B} \qquad \text{and} \qquad \dfrac{\begin{array}{cc}[a:A\to B] & [a:B\to A] \\ \vdots & \vdots \\ s:C & t:C\end{array}}{s \parallel_a t : C}$$

respectively. These rules logically correspond to the excluded middle law $\neg A \vee A$ and to the linearity axiom $(A \to B) \vee (B \to A)$, respectively, and hence allow us to provide a concurrent interpretation of classical logic and Gödel–Dummett logic. The computational rôle of these rules is to introduce the parallelism operator $\parallel_a$. The parallelism operator, in turn, acts as a binder for the communication variables a occurring in s and in t. Thus, in the calculus λ_{Cl} we can compose processes in parallel and establish communication channels of the form

between them. The communication reduction rule (*basic cross reduction*) of λ_{Cl} is

$$\mathcal{S}[a^{\neg A}\, v] \parallel_a t \;\mapsto\; t[v/a] \qquad\qquad \text{for } s = \mathcal{S}[a^{\neg A} v] \text{ and } v \text{ closed term}$$

and can be intuitively represented as

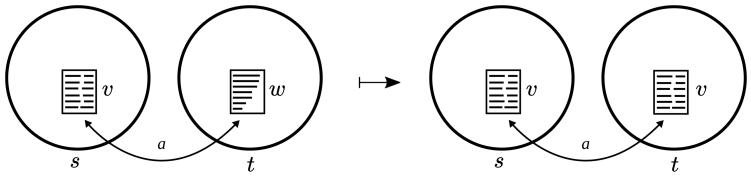


This reduction rule enables us to use channels in one direction only: we can only transmit the argument v of $a^{\neg A}$ from s to t. On the other hand, in λ_{G} we can establish channels of the form

The corresponding basic reduction rules are two, one for transmitting messages from left to right:

$$\mathcal{S}[a\,v] \parallel_a \mathcal{T}[a\,w] \;\mapsto\; \mathcal{S}[a\,v] \parallel_a \mathcal{T}[v] \qquad \text{for } s = \mathcal{S}[a\,v], t = \mathcal{T}[a\,w] \text{ and } v \text{ closed term}$$

which we can represent as



and one for transmitting messages from right to left:

$$\mathcal{S}[a\,v] \parallel_a \mathcal{T}[a\,w] \;\mapsto\; \mathcal{S}[w] \parallel_a \mathcal{T}[a\,w] \qquad \text{for } s = \mathcal{S}[a\,v], t = \mathcal{T}[a\,w] \text{ and } w \text{ closed term}$$

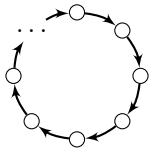
Thus in λ_{G} we can encode dialogues between processes during which messages are exchanged in both directions.

Generalising the ideas used for λ_{Cl} and λ_{G}, we then present a family of concurrent λ-calculi which provide concurrent computational interpretations for all the intermediate logics that can be defined extending intuitionistic logic by axioms of the form

$$(F_1 \to G_1) \vee \ldots \vee (F_n \to G_n)$$

where for $i \in \{1, \ldots, n\}$ no F_i is repeated and if $F_i \neq \top$ then $F_i = G_j$ for some $j \in \{1, \ldots, n\}$.

The corresponding minimal communication topologies generalize those of λ_{Cl} and λ_{G} and include, for instance, cyclic graphs such as



Even though the rather simple communication reductions shown above – the *basic cross reductions* – seem to cover in practice most of the expressiveness needs of concurrent programming, the normalisation of the proof-systems on which the discussed concurrent λ-calculi are based induces

much more general forms of communications. In order to obtain analytic proof-terms, we need also to be able to transmit open processes that have bonds with their original environment. We need thus, more importantly, to be able to restore the required dependencies after the communication. The corresponding computational problem is often called the problem of the *transmission of closures*, see for example [6], and is very well known in the context of *code mobility*, which is the field of study precisely concerned with the issues related to the transmission of functions between programs. Fortunately, our proof systems do not only require very general reductions, but also provides a solution to the problems arising from them. This solution is realized in the presented λ-calculi as the *full cross reduction* rules, which implement the required term communication and establish a new communication channel on the fly in order to handle the dependencies, or *closure*, of the transmitted term.

We prove a general normalization result for the introduced calculi, we show that they are strictly more expressive than simply typed λ-calculus and discuss their computational features.

References

[1] Federico Aschieri, Agata Ciabattoni, and Francesco A. Genco. Gödel logic: from natural deduction to parallel computation. In *LICS 2017*, pages 1–12, 2017.

[2] Federico Aschieri, Agata Ciabattoni, and Francesco A. Genco. Classical proofs as parallel programs. In *GandALF 2018*, 2018.

[3] Arnon Avron. A constructive analysis of RM. *Journal of Symbolic Logic*, 52(4):939–951, 1987.

[4] Arnon Avron. Hypersequents, logical consequence and intermediate logics for concurrency. *Annals of Mathematics and Artificial Intelligence*, 4(3):225–248, 1991.

[5] Agata Ciabattoni and Francesco A. Genco. Hypersequents and systems of rules: Embeddings and applications. *ACM Transactions on Computational Logic (TOCL)*, 19(2):11:1–11:27, 2018.

[6] Jeff Epstein, Andrew P. Black, and Simon L. Peyton Jones. Towards haskell in the cloud. In *ACM Haskell Symposium 2011*, pages 118–129, 2011.

[7] Timothy G. Griffin. A formulae-as-type notion of control. In *POPL 1990*, 1990.

[8] William A. Howard. The formulae-as-types notion of construction. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–491. Academic Press, 1980.

[9] Peter Schroeder-Heister. The calculus of higher-level rules, propositional quantification, and the foundational approach to proof-theoretic harmony. *Studia Logica*, 102(6):1185–1216, 2014.