

A resource modality for RAI

Guillaume Combette (CNRS, ENS Lyon)
Guillaume.Combette@ens-lyon.fr

Guillaume Munch-Maccagnoni (Inria, LS2N CNRS)
Guillaume.Munch-Maccagnoni@inria.fr

31st May 2018

1 RAI and move semantics

Stroustrup’s “*Resource acquisition is initialisation*” idiom (RAI, [Stroustrup 1994](#)) attaches *destructors* to types in C++, called whenever the lifetime of a variable ends, either by the end of its scope being reached, by an exception being raised, or by a control operator (*return*, *break*) being called. It is used in C++ to ensure the *basic exception-safety guarantee* ([Stroustrup, 2001](#)). Unlike finalizers called by a tracing garbage collector, destructors are called at fixed and predictable times.

RAI allows a form of resource management, for ensuring for instance that dynamically-allocated memory is always freed by the end of a scope. It is also used to ensure that locks are always freed, connections are always closed, etc. Doing so amounts to treat locks and connections as resources. Thus, a main point of RAI is that destructors may perform effects.

The extension of C++ with *move semantics* ([Hinnant, Dimov, and Abrahams, 2002](#)) allowed to express the moving of non-copiable resources. Moving a resource alters its lifetime: it can change the order in which destructors are called, or transfer the duty of calling the destructor to a different scope. Notably it allowed the definition of a non-copiable *smart pointer* for automatic resource management (*unique_ptr*) expressing ownership.

[Baker \(1994a,b, 1995\)](#) has proposed a synthesis of the notion of resource from systems programming with that of resources from linear logic ([Girard, 1987](#)). Arguably, it contained an early description of move semantics (it mentioned in particular the compatibility of moving with C++-style destructors). Although these articles described many of the ideas behind the resource management model of the C++11 ([Stroustrup, Sutter, and Dos Reis, 2015](#)) and Rust ([Anderson, Bergstrom, Goregaokar, Matthews, McAllister, Moffitt, and Sapin, 2016](#)) languages, they appear in advance of their time and rarely mentioned. In this presentation, we substantiate a link between C++-style destructors and linear logic.

2 A resource modality for RAI

We consider \mathcal{L} any distributive symmetric monoidal closed category (such as in particular any standard model of linear logic). For any $E \in \mathcal{L}$, there is a monad $-\oplus E$. It has been noticed in [Hasegawa](#)

(2004) that this monad lacks in general a strength, and therefore cannot be used to model exceptions like in a cartesian setting (Moggi, 1991). Intuitively, the operations *bind* and *raise* (or *throw*) need to dispose of variables in their context.

The main idea to model exceptions in \mathcal{L} is to consider the slice category \mathcal{L}/I where I is the monoidal unit. We recall that the *slice category* \mathcal{C}/X of a category \mathcal{C} for $X \in \mathcal{C}$ has objects (A, δ) for $A \in \mathcal{C}$ and $\delta \in \mathcal{C}(A, X)$, and morphisms those in \mathcal{C} that preserve the second component. In particular, (X, id_X) is terminal, and so \mathcal{L}/I is affine. When an object $A \in \mathcal{L}$ interprets a type and $\delta \in \mathcal{L}(A, I)$ interprets a derivation, we think of (A, δ) as another type obtained by attaching the destructor δ to the type A .

We are more generally interested in the case where we are given a strong monad (T, η, μ, σ) on \mathcal{L} . We consider the slice category \mathcal{L}/TI and think of objects $(A, \delta : A \rightarrow TI)$ as destructors that may perform an effect. We recall the following result attributed to Street¹:

Proposition 1. *For any monoid M in \mathcal{L} with multiplication $m \in \mathcal{L}(M \otimes M, M)$ and unit $e \in \mathcal{L}(I, M)$, the slice category \mathcal{L}/M has a monoidal structure with unit $I_M = (I, e)$ and tensor $(A, \delta) \otimes (B, \delta') = (A \otimes B, m \circ \delta \otimes \delta')$. The forgetful functor $U : \mathcal{L}/M \rightarrow \mathcal{L}$ is strict monoidal.*

Now, the object TI has a monoid structure given by $\mu_I \circ \sigma_{TI, I} : TI \otimes TI \rightarrow TI$ and $\eta_I : I \rightarrow TI$. Thus, \mathcal{L}/TI has a monoidal structure and strict monoidal forgetful functor $U : \mathcal{L}/TI \rightarrow \mathcal{L}$. \mathcal{L}/TI has a terminal object (TI, id_{TI}) . Notice that if \mathcal{L} is symmetric, the symmetry does not necessarily lift to a symmetry on \mathcal{L}/TI . This is the case though whenever T is commutative. Otherwise, there is a definite order in the application of destructors: the destructor of $P \otimes Q$ first calls the destructor of Q and then the destructor of P .

We notice that the functor U has a right adjoint if and only if \mathcal{L} has the products $- \times TI$ (as is the case in any model of multiplicative-additive intuitionistic linear logic). One therefore has a monoidal adjunction $\mathcal{L}/TI \xrightleftharpoons[U_R]{U_L} \mathcal{L}$, giving rise to a resource modality $S = UR$ on \mathcal{L} in the sense of Mellies (2009). When \mathcal{L} has finite products, this adjunction has the structure of a (non-commutative) linear call-by-push-value model (Curien, Fiore, and Munch-Maccagnoni, 2016). In particular, its deductive system given by the oblique morphisms of the adjunction (Munch-Maccagnoni, 2014), still expresses multiplicative-additive intuitionistic linear logic, though with fewer identities between derivations, reflecting the presence of an evaluation order. The deductive system includes a symmetry $A \otimes B \vdash B \otimes A$ found in

$$\mathcal{L}(UA \otimes UB, UB \otimes UA) \cong \mathcal{L}/TI(A \otimes B, RU(B \otimes A)),$$

in words, moving resources is available as an effectful operation.

In this setting, we study the monad $T\mathcal{E} = T(- \oplus E)$ on \mathcal{L} and strength-like maps $\theta_{P,A} : UP \otimes T\mathcal{E}A \rightarrow T\mathcal{E}(UP \otimes A)$ defined for $P \in \mathcal{L}/TI$ and $A \in \mathcal{L}$.

3 Resource management modes as polarities

We will conclude with considerations of programming language design following from the analogy:

$$\text{smart pointer} \sim \text{resource modality}$$

¹<https://mathoverflow.net/a/229371>

which is suggested by Chirimar, Gunter, and Riecke, 1996 (a resource modality for a reference-counted garbage collection) and the previous section (a resource modality for *unique_ptr*).

We propose to extend it into an analogy:

resource management mode \sim polarity

where the notion of polarities (Girard, 1991, 1993) suggests a way of mixing different resource management modes as kinds in a functional programming language, presented recently in a companion article (Munch-Maccagnoni, 2018).

References

- Brian Anderson, Lars Bergstrom, Manish Goregaokar, Josh Matthews, Keegan McAllister, Jack Moffitt, and Simon Sapin. 2016. Engineering the servo web browser engine using Rust. In *ICSE '16*. <https://doi.org/10.1145/2889160.2889229> 1
- Henry G. Baker. 1994a. Linear logic and permutation stacks - the Forth shall be first. *SIGARCH Computer Architecture News* 22, 1 (1994), 34–43. <https://doi.org/10.1145/181993.181999> 1
- Henry G. Baker. 1994b. Minimum Reference Count Updating with Deferred and Anchored Pointers for Functional Data Structures. *SIGPLAN Notices* 29, 9 (1994), 38–43. <https://doi.org/10.1145/185009.185016> 1
- Henry G. Baker. 1995. "Use-Once" Variables and Linear Objects - Storage Management, Reflection and Multi-Threading. *SIGPLAN Notices* 30, 1 (1995), 45–52. <https://doi.org/10.1145/199818.199860> 1
- R.F. Blute, J.R.B. Cockett, and R.A.G. Seely. 1996. ! and ?-Storage as tensorial strength. *Mathematical Structures in Computer Science* 6, 4 (1996), 313–351.
- Jawahar Chirimar, Carl A. Gunter, and Jon G. Riecke. 1996. Reference Counting as a Computational Interpretation of Linear Logic. *J. Funct. Program.* 6, 2 (1996), 195–244. <https://doi.org/10.1017/S095679680001660> 3
- Pierre-Louis Curien, Marcelo Fiore, and Guillaume Munch-Maccagnoni. 2016. A Theory of Effects and Resources: Adjunction Models and Polarised Calculi. In *Proc. POPL*. <https://doi.org/10.1145/2837614.2837652> 2
- Thomas Ehrhard. 2016. Effects in Call-By-Push-Value, from a Linear Logic point of view. (2016).
- Jean-Yves Girard. 1987. Linear Logic. *Theoretical Computer Science* 50 (1987), 1–102. 1
- Jean-Yves Girard. 1991. A new constructive logic: Classical logic. *Math. Struct. Comp. Sci.* 1, 3 (1991), 255–296. 3
- Jean-Yves Girard. 1993. On the Unity of Logic. *Ann. Pure Appl. Logic* 59, 3 (1993), 201–217. 3
- Masahito Hasegawa. 2004. Semantics of linear continuation-passing in call-by-name. In *International Symposium on Functional and Logic Programming*. Springer, 229–243. 1

- Howard E. Hinnant, Peter Dimov, and Dave Abrahams. 2002. A Proposal to Add Move Semantics Support to the C++ Language. (2002). <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2002/n1377.htm> 1
- Anders Kock. 1972. Strong functors and monoidal monads. *Archiv der Mathematik* 23, 1 (1972), 113–120.
- Paul Blain Levy. 2005. Adjunction models for call-by-push-value with stacks. *Theory and Application of Categories* 14, 5 (2005), 75–110.
- Paul-André Melliès. 2009. *Categorical semantics of linear logic*. Panoramas et Synthèses, Vol. 27. Société Mathématique de France, Chapter 1, 15–215. 2
- Paul-André Melliès. 2012. Parametric monads and enriched adjunctions. *Unpublished manuscript* 28 (2012).
- Eugenio Moggi. 1991. Notions of computation and monads. *Inf. Comput.* 93, 1 (July 1991), 55–92. [https://doi.org/10.1016/0890-5401\(91\)90052-4](https://doi.org/10.1016/0890-5401(91)90052-4) 2
- Guillaume Munch-Maccagnoni. 2014. Models of a Non-Associative Composition. In *Proc. FoSSaCS (LNCS)*, A. Muscholl (Ed.), Vol. 8412. Springer, 397–412. 2
- Guillaume Munch-Maccagnoni. 2018. Resource Polymorphism. (2018). <https://arxiv.org/abs/1803.02796> 3
- Bjarne Stroustrup. 1994. *The design and evolution of C++*. Pearson Education India. 1
- Bjarne Stroustrup. 2001. *Exception Safety: Concepts and Techniques*. Springer Berlin Heidelberg, Berlin, Heidelberg, 60–76. https://doi.org/10.1007/3-540-45407-1_4 1
- Bjarne Stroustrup, Herb Sutter, and Gabriel Dos Reis. 2015. A brief introduction to C++’s model for type- and resource-safety. (2015). <http://www.stroustrup.com/resource-model.pdf> 1