Rocq CARVe-ing: A Library for Substructural Meta-Theory

Daniel Zackon
McGill University
Montreal, Canada
daniel.zackon@mail.mcgill.ca

Ryan Kavanagh
Université du Québec à Montréal
Montreal, Canada
kavanagh.ryan@uqam.ca

Alberto Momigliano
University of Milan
Milan, Italy
momigliano@di.unimi.it

Brigitte Pientka
McGill University
Montreal, Canada
brigitte.pientka@mcgill.ca

Abstract

We present CARVe ('Contexts as Resource Vectors'), a Rocq library for mechanizing substructural logics and languages. CARVe represents resource usage algebraically, eliminating the need for explicit context splitting and complex re-indexing. This approach integrates smoothly with well-scoped de Bruijn syntax, simplifying formalizations of linear, affine and other substructural systems.

1 Substructural Contexts as Resource Vectors

The structural rules of weakening, contraction, and exchange govern how assumptions in a logical system may be duplicated, discarded, or reordered. Omitting or relaxing these rules gives rise to logics such as linear, affine, or relevant logic, and to corresponding resource-sensitive programming languages. These logics and languages are collectively called *substructural*.

Because they treat assumptions as resources rather than freely reusable hypotheses, mechanizing the meta-theory of substructural languages demands the careful management of contexts. Notably, in linear or affine systems, typing rules often require splitting a context into disjoint parts—each allocated to different subterm—so that no assumption is used more than once. When working with de Bruijn representations of syntax, one typically must carefully re-index variables after each such manipulation, which can substantially complicate formalization.

An alternative approach is offered by CARVe ('Contexts as Resource Vectors') [14], a general framework for representing and manipulating substructural contexts in mechanizations. Building on Schack-Nielsen and Schürmann's approach [9], CARVe annotates each context element with a *multiplicity tag* drawn from a *resource algebra*. These tags encode information about an assumption's availability, such as whether it may be used exactly once, multiple times, or not at all. Instead of physically restructuring a context when assumptions are used or distributed across subterms, CARVe updates their tags to reflect the new usage status. As a consequence, contexts remain essentially intact across proof trees. This design thus integrates naturally with *well-scoped* de Bruijn representations of syntax [1], since the indices and scoping discipline are preserved automatically.

Significantly, the underlying algebraic structure ensures that operations such as context splitting and tag updating are well-behaved by construction: properties of the chosen algebra lift automatically to the level of contexts. This modularity allows CARVe to serve as a general-purpose framework for mechanizing various substructural type systems without requiring modifications to the core context machinery.

CARVe was initially implemented in Beluga [6, 7]. However, the framework is not inherently specific to any one proof assistant. Arguably, CARVe becomes even more valuable in mainstream theorem provers such as Rocq where explicit de Bruijn representations are the norm. Rocq's type system also supports a more modular, reusable CARVe library which can be readily instantiated with different resource algebras. Moreover, Rocq's large and active user and developer community, together with its large ecosystem of libraries that can be combined with CARVe for more complex mechanizations, increasing the practical applicability of the framework.

CARVe is now available as a library in Rocq at

https://github.com/dzackon/rocq-carve.

The implementation is a generalization and abstraction of our previous approach in Beluga [14].

2 Implementation

Appel et al.'s [2] Mechanized Semantic Library (MSL) for separation algebras provides an elegant foundation for implementing CARVe in Rocq. Separation algebras share many commonalities with CARVe's resource algebras. For instance, both have partial operators that are generally associative, commutative, cancellative, functional, unital, and positive (*cf*. [2, § 2] and [14, § 2]). Moreover, just as separation algebraic structure is often lifted to richer type structures like

RocqPL 2026, Rennes, France

lists or function spaces, CARVe also lifts resource algebra structure to contexts modeled as lists or (finite) maps. MSL provides structures for defining separation algebras, reasoning about them, and lifting their structure to richer types. We rely on this infrastructure to implement CARVe in Rocq.

We first define a collection of resource algebras. We model their partial binary operators as ternary relations; specifically, as instances of MSL's `Join` class:

```
Class Join (t: Type) : Type :=
  join: t → t → t → Prop.
```

We then show that each join operation satisfies algebraic properties like associativity or commutativity and instantiate the MSL classes for separation algebras and cancellative algebras. These instances let us apply MSL's extensive libraries of algebraic identities when reasoning about resource algebras.

Next, we define two kinds of contexts for use in case studies: list contexts and function contexts. List contexts are lists of resources tagged with an algebra element, *i.e.*, terms of type `list (R * A)` where `R` is a type of resources and `A` the algebra carrier. List contexts are suitable for modeling contexts of named or unnamed resources, like sequents. Function contexts are total functions `D → R * A` where `D` is usually a finite set. In particular, when working with de Bruijn indices, elements in `D` are finite sets of natural numbers. We lift `A`'s algebraic structure to list and function contexts using MSL's separation algebra generators. For example, a product generator defines the obvious component-wise join operator on products of algebras (we endow `R` with a trivial algebra), and the list generator similarly defines a join operator on lists element-wise. The induced join operator on list contexts corresponds exactly to its inductive characterization in [14], and it instantiates the same MSL classes as `A`. This generator-based approach lets us lift algebraic properties from resource algebras to context merging for free, and avoids the need to prove such properties on a case-by-case basis in proof developments.

3 Case Studies

Alongside our implementation of CARVe, we have used it to mechanize several case studies in Rocq, porting and modifying our previous development in Beluga [14]. Since we cannot use higher-order abstract syntax as a representation technique, we follow the standard well-scoped de Bruijn approach. Importantly, context splits require no index shifting or renaming to preserve well-scopedness, since CARVe contexts remain structurally intact across splits: the datatype of terms is identical to that used for the systems' ordinary intuitionistic counterparts. Linearity, in other words, is not imposed as a syntactic property but as a semantic one, captured separately through resource annotations.

As a result, we can use the Autosubst [11] library essentially out of the box for the main auto-generated definitions.

Weak Normalization. Our main case study includes two logical-relations proofs of weak normalization for an intuitionistic-linear λ-calculus with unit and bang types. One follows the approach of Dreyer et al. [3] as mechanized by Stark [10], who split the logical relation into value and expression components, with only the former defined recursively on the type. The other proof adopts the traditional approach, defining the logical predicate recursively on the structure of types. In both cases, we proceed in the usual way by proving that every well-typed term satisfies its logical predicate, from which weak normalization follows as a corollary.

A key ingredient in both proofs is the notion of reducible simultaneous substitution, *i.e.*, mappings from variables to terms within the logical relation at the appropriate type. In traditional mechanizations of substructural systems with de Bruijn indices, reasoning about such substitutions is a burdensome task: they too must be explicitly split and merged, which requires a careful treatment of indices. With CARVe, however, we obtain as a near-immediate lemma that reducible substitutions are preserved across context splits.

Type safety. We implement proofs of progress and preservation for the above-mentioned intuitionistic-linear λ-calculus as a traditional benchmark. The proof of progress is a straightforward adaptation of the existing Rocq script from *Software Foundations* [8]. Preservation is more subtle, and we present two mechanizations. The first is based on context morphisms and requires a refined type-instantiation lemma that accounts for linear resource splitting. The second proceeds via a linear substitution lemma and requires proofs of the structural properties governing used assumptions.

Cut Elimination. The final case study shows progress for a fragment of the session-typed process calculus CP [12]. CP is a proofs-as-processes interpretation of classical linear logic. Its processes are typed by list contexts that specify the types of communication channels. Proving progress for this calculus amounts to proving cut elimination for the underlying linear logic.

4 Future Work

There are several directions for extending and refining the CARVe Rocq library. First, we aim to provide a general API outlining all the operations and properties we support. Secondly, to cover a wider range of substructural type systems, we plan to support additional algebras, in particular those incorporating ordered resources. Further, we intend to generalize the infrastructure to handle other representations of contexts and intrinsically-typed representations of terms. Finally, an in-depth comparison of our development with similar endeavours such as [4, 5, 13] is left to future work. (A summary of related approaches may be found in [14, § 6].)

References

[1] Robin Adams. 2006. Formalized metatheory with terms represented by an indexed family of types. In *Proc. TYPES '04 (Lect. Notes Comput. Sci., Vol. 3839)*, Jean-Christophe Filliâtre, Christine Paulin-Mohring, and Benjamin Werner (Eds.). Springer, 1–16. doi:10.1007/11617990_1

[2] Robert Dockins, Aquinas Hobor, and Andrew W. Appel. 2009. A fresh look at separation algebras and share accounting. In *Proc. APLAS '09 (Lect. Notes Comput. Sci., Vol. 5904)*, Zhenjiang Hu (Ed.). Springer, 161–177. doi:10.1007/978-3-642-10672-9_13

[3] Derek Dreyer, Simon Spies, Lennard Gäher, Ralf Jung, Jan-Oliver Kaiser, Hoang-Hai Dang, David Swasey, Jan Menz, Niklas Mück, and Benjamin Peters. 2025. Semantics of type systems (Lecture notes). https://plv.mpi-sws.org/semantics-course/lecturenotes.pdf

[4] Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Ales Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *J. Funct. Program.* 28 (2018), e20. doi:10.1017/S0956796818000151

[5] Olivier Laurent. 2017. Yalla. https://github.com/olaure01/yalla/.

[6] Brigitte Pientka and Jana Dunfield. 2008. Programming with proofs and explicit contexts. In *Proc. PPDP '08*. 163–173. doi:10.1145/1389449.1389469

[7] Brigitte Pientka and Jana Dunfield. 2010. Beluga: A framework for programming and reasoning with deductive systems (system description). In *Proc. IJCAR '10 (Lect. Notes Comput. Sci., Vol. 6173)*, Jürgen Giesl and Reiner Hähnle (Eds.). Springer, 15–21. doi:10.1007/978-3-642-14203-1_2

[8] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, Andrew Tolmach, and Brent Yorgey. 2025. *Programming Language Foundations*. Software Foundations, Vol. 2. http://softwarefoundations.cis.upenn.edu (Version 6.7).

[9] Anders Schack-Nielsen and Carsten Schürmann. 2010. Curry-style explicit substitutions for the linear and affine lambda calculus. In *Proc. IJCAR '10 (Lect. Notes Comput. Sci., Vol. 6173)*, Jürgen Giesl and Reiner Hähnle (Eds.). Springer, 1–14. doi:10.1007/978-3-642-14203-1_1

[10] Kathrin Stark. 2019. *Mechanising syntax with binders in Coq*. Ph. D. Dissertation. Saarland University, Saarbrücken. https://www.ps.uni-saarland.de/~kstark/thesis/

[11] Kathrin Stark, Steven Schäfer, and Jonas Kaiser. 2019. Autosubst 2: Reasoning with multi-sorted de Bruijn terms and vector substitutions. In *Proc. CPP '19*, Assia Mahboubi and Magnus O. Myreen (Eds.). ACM, 166–180. doi:10.1145/3293880.3294101

[12] Philip Wadler. 2012. Propositions as sessions. In *Proc. ICFP '12*. 273–286. doi:10.1145/2364527.2364568

[13] James Wood and Robert Atkey. 2022. A framework for substructural type systems. In *Proc. ESOP '22 (Lect. Notes Comput. Sci., Vol. 13240)*, Ilya Sergey (Ed.). 376–402. doi:10.1007/978-3-030-99336-8_14

[14] Daniel Zackon, Chuta Sano, Alberto Momigliano, and Brigitte Pientka. 2025. Split decisions: Explicit contexts for substructural languages. In *Proc. CPP '25*. 257–271. doi:10.1145/3703595.3705888