A type system framework for safe (a)synchronous concurrent programming

João Mota[1]*

NOVA LINCS & NOVA School of Science and Technology, Portugal jd.mota@campus.fct.unl.pt

Introduction Our objective is to develop a **type system framework** parametric over suitable *separation algebras* [3] (taking inspiration from Iris [11]) to facilitate the creation of new type systems that statically guarantee memory safety, thread-safety, and avoid memory leaks, in concurrent languages featuring asynchronous programming.[1] We desire to strike a *balance between proof ease and expressiveness*, while having built-in support for crucial features:

1. The ability to distinguish between affine and linear resources. The former would be allowed to be "dropped" and disposed by a garbage collector, while the latter would need to be explicitly disposed to prevent memory leaks.

2. The ability to distinguish between thread-local and thread-shared resources, allowing thread-local data to be accessed without synchronization.

3. Allow one to perform non-frame-preserving updates (for example on thread-local data) that break assumptions made by other parties sharing the same underlying state, and then permit one to restore consistency before changes become visible to others (similar to how one can open an invariant and then close it after restoring it).

4. Allow one to reason about fictional (weak) separation and real (strong) separation of resources, or even other degrees of separation.

We believe the proposal addresses concrete limitations in the state of the art. First, recent logics, such as Iris, have usually preferred to use affine resources. Although there are encodings to allow precise tracking of resources in affine logics, such as Iron [1], and even though there is work extending Iris with linear resources [10], it would be useful to have both notions as built-in and principled features. For example, even in garbage-collected languages, it is still important to guarantee that resources, like sockets, are closed, even though there is no instruction to actually free memory. So, one could imagine parameterizing the framework with separation algebras to reason about typestated objects [16, 7, 8], whose protocol must be completed, but allowing other non-typestated objects to be treated as affine resources.

Second, there is some work on capabilities which distinguish between thread-local from thread-shared data [19, 6, 5], but the set of available capabilities is fixed and there are limitations to how data may be transferred between threads. In Iris, it is possible to encode "thread-local invariants" which can be opened non-atomically [9], but doing the encoding is non-trivial and requires expertise. Combined with the ability to perform *non-frame-preserving updates on thread-local data*, one could leverage these features to reason about *asynchronous programming* in languages such as JavaScript [15], where control is only yielded from a task to another at specific points (i.e., we have concurrently without preemption [17]). Knowing that there is no

*This talk proposal intends to present ongoing PhD work supervised by António Ravara and Marco Giunti.

[1] A poster showcasing our objective, presented at the PLDI Student Research Competition, which includes a motivating example, may be found online [14], together with the submitted abstract.

interference when a task is running (in a single-thread), one can avoid the use of synchronization primitives to ensure mutual exclusion.

Finally, although there is work that supports, simultaneously, the distinction between fictional (weak) separation and real (strong) separation [2], one could generalize and develop a framework parametric over *separation degrees*, in the same way there are now separation logic frameworks parametric over separation algebras [11, 3].

Preliminary results To achieve our goal, we have started by developing a core separation logic framework, parametric over a separation algebra, with the usual separation logic connectives. This approach is very similar to Iris, but with some differences at the outset. First, we simplify the setting by remaining in the first-order realm, and avoid step-indexing.

Second, we allow one to define affine and linear resources by letting the user supply a preorder relation over resources, following the approach by Cao et al. [4] and Krebbers et al. [12]. If $a \sqsubseteq b$, then a proposition that holds for a, also holds for b (i.e., we can always add affine resources without invalidating a proposition). If $\epsilon \sqsubseteq a$, then it means a is completely affine.

Third, we allow the separating conjunction, the magic wand connectives, and the update modality (inspired in Iris), to be parameterized with *separation degrees*, which are relations over any two resources, following some properties. Naturally, we can derive a notion of "degree subtyping", defined mathematically as set inclusion. For example, real (strong) separation would be a "subtype" of fictional (weak) separation. We have also demonstrated the following distributivity property (like in [2]): If $l \subseteq l'$, then $(A *_{l'} B) *_l (C *_{l'} D) \models (A *_l C) *_{l'} (B *_l D)$.

Finally, we have specified a weakest-precondition, defined Hoare triples in terms of weakest-preconditions, proved that triples are sound and ensure safety, and derived a frame rule. These preliminary results have all been mechanized in the Rocq proof assistant.

Future work The plan is to then extract a decidable fragment of this core separation logic framework, and use it to create a type system framework. With the given safe and sound foundation, the *type safety* property of type systems, created with this framework, is naturally derived. Moreover, if more complex verification scenarios arise, one should be able to "drop down" to the core separation logic framework, and do the required proofs there.

However, the work is not complete. Namely, a crucial feature we are interested in is the ability to perform non-frame-preserving updates. For this, we are studying a new kind of update modality. Although there is already work on a non-frame-preserving update modality [18], the critical aspect we want to develop is the ability to recover consistency.

For this feature, we need some key ingredients. First, we need to track the resources that were under a non-frame-preserving update, and that must be restored. Second, we must "hide" resources that may be affected by a non-frame-preserving update. This approach is heavily inspired in the work my Militão [13] on Rely-Guarantee Protocols. In that work, one performs protocol steps by "focusing", and then "defocusing", where "focusing" produces a "guarantee" that must be fulfilled upon "defocus". In the middle of a step, inconsistencies are not visible to other parties sharing the same state. To allow for modular proofs, Militão presents a version of a frame rule that extends the memory footprint using a special operator that "hides" inconsistent resources to avoid reentrancy (i.e., those that share underlying memory with resources currently being modified). However, this "hiding" is coarse-grained. To address this, Militão [13] suggests to distinguish data that is fictionally separated from data that is truly separated, and then only "hiding" the former - a proposal that we plan to implement leveraging on the "separation degrees" we previously discussed. Finally, Militão's approach is syntactic, while we are interested in having a semantic model for these kinds of updates.

References

[1] Ales Bizjak, Daniel Gratzer, Robbert Krebbers, and Lars Birkedal. Iron: managing obligations in higher-order concurrent separation logic. *Proc. ACM Program. Lang.*, 3(POPL):65:1–65:30, 2019.

[2] James Brotherston, Diana Costa, Aquinas Hobor, and John Wickerson. Reasoning over permissions regions in concurrent separation logic. In *Proceedings of Computer Aided Verification*, volume 12225 of *Lecture Notes in Computer Science*, pages 203–224. Springer, 2020.

[3] Cristiano Calcagno, Peter W. O'Hearn, and Hongseok Yang. Local Action and Abstract Separation Logic. In *Proc. of the 22nd IEEE Symposium on Logic in Computer Science (LICS 2007), Poland*, pages 366–378. IEEE Computer Society, 2007.

[4] Qinxiang Cao, Santiago Cuellar, and Andrew W. Appel. Bringing Order to the Separation Logic Jungle. In *Proc. of Programming Languages and Systems - 15th Asian Symposium, APLAS 2017, China*, volume 10695 of *LNCS*, pages 190–211. Springer, 2017.

[5] Elias Castegren and Tobias Wrigstad. Reference Capabilities for Concurrency Control. In *30th European Conference on Object-Oriented Programming, ECOOP 2016, Italy*, volume 56 of *LIPIcs*, pages 5:1–5:26. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[6] Sylvan Clebsch, Sophia Drossopoulou, Sebastian Blessing, and Andy McNeil. Deny capabilities for safe, fast actors. In *Proc. of the 5th International Workshop on Programming Based on Actors, Agents, and Decentralized Control, AGERE! 2015, USA*, pages 1–12. ACM, 2015.

[7] Robert DeLine and Manuel Fähndrich. Typestates for Objects. In *Proc. of ECOOP 2004 - Object-Oriented Programming, 18th European Conference, Norway*, volume 3086 of *LNCS*, pages 465–490. Springer, 2004.

[8] Ronald Garcia, Éric Tanter, Roger Wolff, and Jonathan Aldrich. Foundations of Typestate-Oriented Programming. *ACM Trans. Program. Lang. Syst.*, 36(4):12:1–12:44, 2014.

[9] Iris Project. The Iris 4.1 Reference, 2023. Accessed: 2024-01-15.

[10] Jules Jacobs, Jonas Kastberg Hinrichsen, and Robbert Krebbers. Deadlock-Free Separation Logic: Linearity Yields Progress for Dependent Higher-Order Message Passing. *Proc. ACM Program. Lang.*, 8(POPL):1385–1417, 2024.

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

[12] Robbert Krebbers, Jacques-Henri Jourdan, Ralf Jung, Joseph Tassarotti, Jan-Oliver Kaiser, Amin Timany, Arthur Charguéraud, and Derek Dreyer. MoSeL: a general, extensible modal framework for interactive proofs in separation logic. *Proc. ACM Program. Lang.*, 2(ICFP):77:1–77:30, 2018.

[13] Filipe David Oliveira Militão. *Rely-Guarantee Protocols for Safe Interference over Shared Memory.* PhD thesis, Universidade NOVA de Lisboa (Portugal), 2015.

[14] João Mota. PLDI 2024 Student Research Competition Poster and Abstract. https://jdmota.github.io/#material, 2024.

[15] Mozilla. The event loop - JavaScript — MDN, 2024. Accessed: 2024-02-29.

[16] Robert E. Strom and Shaula Yemini. Typestate: A Programming Language Concept for Enhancing Software Reliability. *IEEE Trans. Software Eng.*, 12(1):157–171, 1986.

[17] Andrew S. Tanenbaum. *Modern operating systems, 4rd Edition.* Pearson Prentice-Hall, 2014.

[18] Simon Friis Vindum, Aïna Linn Georges, and Lars Birkedal. The Nextgen Modality: A Modality for Non-Frame-Preserving Updates in Separation Logic. In *Proc. of the 14th ACM SIGPLAN International Conference on Certified Programs and Proofs*, USA, 2025. Association for Computing Machinery.

[19] Tobias Wrigstad, Filip Pizlo, Fadi Meawad, Lei Zhao, and Jan Vitek. Loci: Simple Thread-Locality for Java. In *Proc. of ECOOP 2009 - Object-Oriented Programming, 23rd European Conference, Italy*, volume 5653 of *LNCS*, pages 445–469. Springer, 2009.