Project Milestone 1

Justin Frank Pierce Darragh

The first project milestone requires a literature review, a report of concrete steps taken so far, and any relevant notes on course correction.

Literature Review

"A Type System for Safe Intermittent Computing." This paper [4] presents Curricle, a system that claims to be the first type system approach to safe intermittent computation. The key insight lies in finding a balance in ease of programming within the system while maintaining enough safety guarantees to be useful, and this is accomplished by giving programmers the ability to explicitly annotate expectations of (non-)idempotence. The system then deduces which variables must be saved and at what points to guarantee these expectations are met, or else a compile-time error is raised to indicate that there does not exist a satisfactory solution.

Our project aims to reproduce much of the Curricle result with a new, small Imp-like language. However, we wish to add a formal model of checkpoint semantics derived from the CESK style, and then mechanically verify this model in Rocq.

"Towards a Formal Foundation of Intermittent Computing." This paper [3] provides a formal model of intermittent execution with a set of definitions for correctness, various types of errors that can occur during execution, and memory relations. It also outlines a set of reasonable assumptions and provides a strategy for proving an intermittent system is correct. We intend to use these definitions, assumptions, and proof strategy to guide the development of our own formal model, since the Curricle paper relies on some of these in its own definitions. Additionally, while the Curricle paper does provide new definitions and theorems, "Towards a Formal Foundation" is a bit more verbose in this regard and so serves as a good reference for shaping our own definitions and theorems for our proofs since we intend to deviate a bit from Curricle's specification.

Programming Languages and Lambda Calculi. PLLC [1] is a big book with a lot of words, but the words relevant to our use concern the definition of CESK machines and proofs regarding their operation. PLLC builds up the formal semantics of the CESK machine along with discussion and examples of proofs of the semantics. This gives us a point from which we can formally define our new CESK-like semantics and reason about proofs thereof.

"Abstracting Abstract Machines." The AAM paper [5] provides a significantly abridged look at the CESK machine compared to PLLC, and AAM also takes steps to walk through the extension of CESK through the lens of abstraction. The paper also demonstrates by example some of the particular characteristics of CESK compared to CEK. This last part is important to us because we will need to make an analogous step from CESK to account for having both volatile and non-volatile memory in our checkpoint system.

Software Foundations. The first volume of *Software Foundations* [2] defines a small imperative programming language called *Imp*. We will use this language definition as a basis for the language in our project. The most significant deviation from *Software Foundations*'s implementation is that our language's semantics will be modified to suit the context of intermittent computing (e.g., by specifying the semantics of checkpointing).

In the second volume of *Software Foundations*, a notion of program equivalence is defined for the Imp language, but the authors use a Hoare Logic for these results. We may reference these proofs in our work, but we intend to prove execution equivalence in intermittent contexts rather than program equivalence.

Concrete Steps Taken

So far, we have accomplished less than we had hoped. One of the project members (Pierce) was out of the country for an extended duration at an inconvenient time, causing delays in getting the project off the ground. However, what we've done so far is:

- Discussed course correction (detailed in the next section).
- Established an actual project scope with steps to follow (detailed below).
- Began the process of specifying our language and formal model (mostly verbal, but the point is that we're actively talking about it now that we know what we're really going to do).
- Set about reading the accumulated works more closely.

The new steps we have planned out for our project are:

1. Thoroughly read and discuss all our related literature.
2. Establish the language to be implemented. (It's a modified version of Imp, but we need to be more specific in describing its feature set.)
3. Develop a CESK-like abstract machine for undo checkpointing and intermittent computation.
4. Formally specify the semantics of our extended Imp language in the context of the new abstract machine.
5. Mechanically verify the above, probably in Rocq.

We expect step (2) to be completed "on paper" rather than in Rocq.

Although the final step, (5), is somewhat glossed over, we expect it to consume a significant portion of the total time spent on the project. The definitions and theorems (and possibly the implementations of the related proofs) will be developed in concert with the semantics, so really step (5) is part of steps (3) and (4).

Course Correction

Our project has changed trajectory slightly since our initial proposal.

The new goal is to re-implement the result of the Curricle paper, but with a slightly different model complete with a mechanized proof of its correctness. This topic is appealing to our team because we both enjoy working with type systems and semantics, and we both appreciate the value of mechanized proofs. It would be beyond the scope of a class project to develop an entirely novel type system in the area of PL+Arch, so we chose to avoid attempting such a feat; and separate papers often establish quite distinct definitions or assumptions in building their systems, so it did not seem feasible to set out to mechanize the proof of multiple disparate papers.

A significant upside of this minor change in course is that we have a very complete scope for our project. This establishes the goal by which we can judge our completion, and it also helps us to have a more concrete idea of what steps we will take. Additionally, it means we already have some conventions and definitions we can use as a starting point for our implementation. All together, these characteristics of the new project direction mean that we know what we are doing and what steps we need to take to do it, and really we aren't behind the timeline we had originally intended.

References

[1] Matthias Felleisen and Matthew Flatt 2003. *Programming Languages and Lambda Calculi.* Electronic textbook.

[2] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg and Brent Yorgey 2023. *Logical Foundations.* Electronic textbook.

[3] Milijana Surbatovich, Brandon Lucia and Limin Jia 2020. Towards a Formal Foundation of Intermittent Computing. *Proc. ACM Program. Lang.* 4, OOPSLA (Nov. 2020). DOI:https://doi.org/10.1145/3428231.

[4] Milijana Surbatovich, Naomi Spargo, Limin Jia and Brandon Lucia 2023. A Type System for Safe Intermittent Computing. *Proc. ACM Program. Lang.* 7, PLDI (Jun. 2023). DOI:https://doi.org/10.1145/3591250.

[5] David Van Horn and Matthew Might 2010. Abstracting Abstract Machines. *SIGPLAN Not.* 45, 9 (Sep. 2010), 51–62. DOI:https://doi.org/10.1145/1932681.1863553.