Project Milestone 2

Justin Frank Pierce Darragh

The second project milestone requires a status report on the previous milestone's next steps, a new list of next steps from this point, and a plan for evaluating the project at the end. Along the way, we detail a little bit of some of our recent challenges.

Review of Previous Next Steps

1. Thoroughly read and discuss all our related literature. We have read our literature (in varying degrees of thoroughness, but thoroughly enough for our purposes so far).

2. Establish the language to be implemented. We have written down a specification of the grammar of our new language, Chump, and we have discussed at length what features it shall support. We will not have structs or functions of any kind, but we will have looping, variable assignments and updates, arithmetic, checkpoints, and both input and output operations.

3. Develop CESK-like abstract machine. We have defined the extension of CESK that we believe will carry us through to the completion of the project. The CESK tuple is extended with P, which represents the state saved by the last checkpoint. It is a 3-tuple consisting of a continuation, a lesser store, and an environment, all of which encode the state that should be resumed when a reboot occurs after power failure. Each checkpoint operation that succeeds drops the old P and installs a new one.

4. Formally specify the semantics. We have begun handwriting our semantics. Much of this is standard and so we have not written it down explicitly; rather, we have focused our orthographic efforts on the novel aspects of our system, such as the rules that update the new P element of the CESKP machine (most ignore it). We have also sketched out our approach to formally specify other aspects of our system, which shall be expanded on a bit in the next section.

5. Mechanically verify the above. We have begun the work of mechanization by defining the syntax of the language and the shape of the semantic relation. However, this has all been in flux over the past few days as we have identified and overcome a series of interesting challenges in the work (again, expanded on in the next section).

Ongoing Challenges

In our work so far, we have identified challenges to our approach. Some of these have been overcome already, and others are in-progress but not yet resolved.

Incorporating Output

The original Curricle model (and others like it) seem to leave out modeling output. This may be because it is difficult. However, since we are severely limiting the functionality of the language we are modeling, we thought we'd like to incorporate a semantics for output.

Curricle's correspondence theorem essentially states that a given intermittent execution that does not experience power failure is equivalent to a continuous execution. However, we have decided to reformulate the

thought process: given an intermittent execution with any arbitrary power failures, there exists a continuous execution with equivalent observable behavior. We have theorized a mechanism for tracing input consumption and output generation relative to time-steps and checkpoints such that any intermittent execution will produce an I/O trace, a subset of which corresponds to a plausible continuous execution. (We don't have the formal statement together just yet, though.)

Memory Model

Curricle relied on some of the behavior of Rust, atop which it was built. We are formalizing the language entirely, so we have to model memory a bit differently. However, since we aren't dealing with structs or functions, the modifications needed to regular CESK semantics should not be too extreme. Most of what we need to focus on is tracking checkpointed values, but we would also like to handle some notion of "garbage collection" in the sense that variables that fall out of scope free up space in the store.

Loops with Breaks

We are implementing a generic, unparameterized `loop` form instead of the more traditional `while` loop. To permit termination, we will provide an indexed `break` form, the index of which indicates how many looping levels to break out of. This is another feature we chose so we could add just a little complexity to our system, as a treat. However, while we do not anticipate the semantics of this construct to pose too much of an issue, it is as yet unclear to what extent the type system will make us regret this choice.

New Next Steps

On our horizon:

1. Finish implementation of semantics in Rocq.
2. Implement type system in Rocq.
3. Work through/develop formal statements of correctness and correspondence.

Further Steps

We have also imagined some "stretch goals" in case everything else wraps up quickly enough:

1. Add (non-recursive?) functions.
2. Add structs.
3. Implement a DSL for Chump in Racket (because we both like Racket).
4. Write up the system to submit to a workshop, if that seems reasonable.

Evaluation Strategy

(See what we did there?)

In the shortest terms, the evaluation of our system comes down to whether the Rocq type-checker accepts it. That said, this evaluation metric does not exactly accommodate the practical time constraints, nor does it provide anything other than a binary assessment.

Considering where we are now and how things are currently going, the following elements seem to be within the specification of a Minimum Viable Product:

- Full syntax and semantics specified in Rocq, including fully updated CESKP machine and IO handling.
- Base Curricle type-checker implemented in Rocq. (Not including new features.)
- Formal statements of correctness worked out and at least stated in Rocq, though not yet proved.

Beyond that, we believe we may be able to accomplish the following before the presentation date:

- Full type-checker implemented in Rocq.

- Base type inference system implemented in Rocq (*à la* Curricle).
- Significant portion of correctness proof worked out and needed revisions implemented.

And, lastly, we would *like* to have:

- Full type inference.
- Full correctness proof.
- Functions and structs.

But these last goals seem somewhat lofty, given the timeline, so we do not feel that they would make for a fair assessment in the course. Additionally, time will need to be carved out to prepare a decent presentation for the class.