Meeting Notes

Tuesday 17th March, 2020

1 Discussions about the notes

- Uninitialised capabilities

  We agree that it looks good! Armaël suggested it might be easier to make store initialise instead of storeu (where it currently increments it if we are on top). Would it make it complicated to just have store do the increment instead? proposal: not have a special case for storeU, but rather have a special case for store for U- capabilities that do the increment. The intuition is that we offset the load for proving everything in storeU, and split the work more evenly between store and storeU. Another alternative, is to enable store to write to U- capabilities, but have storeU be the specific store + increment. (Related work is making sure that all the regular stores be replaceable with storeU). For now we leave it like it is, but if the FTLR for storeU becomes too big, we might want to consider a split.

- Device drivers

  Comments from Lars:

  - The semantics are non deterministic because of the IOload, we should take note of all sources of non-determinism since they can cause trouble later on.
  - What is the simplest possible client program that one could consider? where one can think about instantiating P with something concrete (e.g. even numbers). It might be worth considering, since it forces us to compose these different parts. The original client Dominique presented is that we are not accessing certain parts of memory. A similar but contrived examples is that every element of the trace does not go above a certain threshold. Even though P is simple, it is interesting to see how these specs might fit together.
  - Reactive programming uses streams of outputs. An alternate to the non determinism is that Load takes the next element in some oracle stream.

    Non determinism can be one way of modelling input, streams could be an alternative. In CakeML there is an abstract type for the state of the outside world, and an oracle and operations that change the state of the outside world. Whenever an IO operation happen, we keep track of it in a trace. This means we keep a model of the external world in the semantics. Which model is going to make the rest of the story the simplest here? What is the impact of non determinism? Here it's still internally deterministic, but it's with the external interactions with the world that non-determinism happen. There might a way to relax the determinism restriction for the WP lemmas we will need. To make it deterministic, we can make x be a parameter of the step (this parameter can be given by some kind of oracle).

    Non determinism can lead to challenged down the road. We should be aware of this!

    Let's consider making it deterministic using some kind of oracle to avoid such challenges.

  Discussion about the spec on page 10/11:

  - If the boot code is doing IO, then we also need to state that the boot code produces a trace t for which P(t) holds.
  - We assume that code can produce multiple entry and exit points. Proposed alternative: one entry point and a registers that select what kind of thing we want to do.
  - It might make things easier to read if we use notation for the fragmental trace resources.
  - Some comments from Dominique:

    Here the theorem does not make any assumptions about memory. (here we know nothing about memory. If memory contains any capabilities, the boot code will need to clear memory in order to satisfy the spec. What if we do know that memory is all 0's at boot? then we cannot prove that the boot satisfies the step without clearing. This might be an issue? We need to mention that as a guarantee in the theorem. Let's figure out what happens in real life before we add such an assumption).

    spec on page 11: universal quantification over all W. No link between the world the boot code sets up and the world the expression relation is prover over. We need to consider all private future worlds of the world the boot code allocates.

    page 10: we will need some assumptions over the memory that the entry points are valid.

can we generalise the entry points to be any permission, not just E-capabilities (i.e. closures). We could generalise the statement by stating validity. This might simplify the condition, the PC is kept as only of the registers.

- Some comments from Frank: Does the boot code need to do IO? depending on the goal of the work: if we want to model what OS boot code does on IO, then yes, boot code does initialisation of devices which might leave a trace). Do we want to see this evolve into proving an actual boot code correct? Or do we want to focus on something where more fancy Iris stuff is needed. These might be two different directions for the project.

2 Next steps for the driver

- fix the statement of the theorem such that the world produced by the boot code is linked to the world the expression relation must hold over.
- think of a client and a proof sketch of that minimal client (on paper) to get an idea of how these moving parts fit together.

3 Next steps for the uninitialized capability

- prove the awkward example without clearing the stack upon return. This step will need further discussions (Aina will send an email thursday if such a discussion can already happen on friday)
- start proving WP's and FTLR for the now implemented semantics