Cb (C-flat) is an intermediate-level subset of the C language. In IDRS S515 this summer, I specified Cb's syntax and (somewhat informally) its semantics, as well as the state of a virtual machine that executes the language natively [see Appendix A]. I showed that a parametrization of the machine can be plugged into a framework developed by Alex Vaynberg and Professor Shao (the subject of an upcoming paper tentatively titled Simplifying Program Certification through the Use of Abstraction-Specific Languages and Certifying Compilers) and that the framework can be used to certify simple Cb programs.

Providing performs $\iota$, branch performs $\beta$, and state specification $\mathcal{S}$, one reconstructs in their framework program segments in a linearized form (i.e. with cycles removed and, in some cases, complicated expressions unfolded) as trees ($f$, $I$ below) in a meta-language with instructions in the original language replaced by performs and branch-performs. Progress and preservation properties are guaranteed within the framework for any language parametrization accompanied by a sanity proof. The power of the framework lies in the comparatively small amount of effort that should ultimately be required to certify programs and libraries once the parametrization has been accepted.

\[
\text{(State)} \quad \mathcal{S} ::= (\text{abstract})
\]
\[
\text{(Program Segments)} \quad I ::= (\beta,I_1,I_2) \mid \iota ; I \mid \text{call } f ; I \mid I_{\text{init}}
\]
\[
\text{(Performs)} \quad \iota ::= (\text{abstract})
\]
\[
\text{(Branch Perfs)} \quad \beta ::= (\text{abstract})
\]

I now propose to

- refine the syntax and semantics of Cb for the sake of developing a cleaner, more restricted intermediate language, about which it is easier to reason
  - (specifically, Cb currently allows multiple consecutive return statements (or none at all) in a given function, giving the programmer too much power to explicitly manipulate the call stack),
- develop a translation (or, if possible, a bijection) conducive to a high confidence of adequacy between native Cb and the (linearized) meta code accepted by Vaynberg & Shao's framework [see Appendix B for my current progress on this task],
- develop and certify libraries in Cb, including implementations of malloc and a linked list data structure, as a demonstration of the power of the framework,
- define extensions of the Cb language in which already-certified library functions are included as primitive operations, and show adequacy of reasoning in terms of the new primitives as a substitute for reasoning in terms of libraries, and
- explore the possibility of extending Cb to include type definition and function pointers (so as to fully capture the power of higher level languages such as C) without significantly increasing the difficulty of the code certification process.

Deliverables will include definitions and parametrizations of Cb in its original and extended forms, Cb code and meta-code of all libraries I certify, the coq code certifying the libraries, and a written report fully explaining the implemented libraries, their certifications, and extensions of the Cb language.