**Exercise 6.14** Types play an important role in compilation, which is not reflected in the development of this chapter. Ideally, we would like to take advantage of type information as much as possible in order to produce more compact and more efficient code [**?**]. This is most easily achieved if the type information is embedded directly in expressions (see Section **??**), but at the very least, we would expect that types can be assigned to intermediate expressions in the compiler.

1. Define typing judgments for de Bruijn expressions, environments, and values for the language of Section 6.2. You may assume that values are always closed.
2. Prove type preservation for your typing judgment and the operational semantics for de Bruijn expressions.
3. Prove type preservation under compilation, that is, well-typed Mini-ML expressions are mapped to well-typed de Bruijn expressions under the translation of Section 6.2.
4. What is the converse of type preservation under compilation. Does your typing judgment satisfy it?
5. Implement the judgments above in Elf.
6. Implement the proofs above in Elf.

**Exercise 6.15** As in Exercise 6.14:

1. Define typing judgments for expressions, values, instructions, continuations, and machine states for the continuation machine of Section **??**.
2. Prove type preservation for your typing judgment under the operational semantics of the machine.
3. Prove a *progress lemma* for well-typed programs: Any valid machine state is either the final answer or there is a possible next transition.
4. Implement the judgments above in Elf.
5. Implement the proofs above in Elf.
6. Extend the typing judgment, proofs, and implementations to include **letcc** $k$ **in** $e$.

**Exercise 6.16** As in Exercise 6.14:

1. Define a typing judgment for evaluation contexts. It should only hold for valid evaluation contexts.
2. Prove that splitting a well-typed expression which is not a value always succeeds and produces a unique context and redex.

3. Prove that splitting a well-typed expression results in a valid evaluation context and valid redex.

4. Prove the correctness of contextual evaluation with respect to the natural semantics for Mini-ML.

5. Implement the judgments above in Elf. Evaluation contexts should be represented as functions from expressions to expressions satisfying an additional judgment.

6. Implement the proofs above in Elf.