Homework 3: Parameterization

Due: 1(a)–1(c) on April 14; revision and 1(d) on May 6
In this homework, you will learn more about parameterization and the correspondence principles.
If you wish, you can work in groups, but I will expect more extensive languages from groups.

1. This is the first installment on a term project. Define the syntax, typing, and semantics for some language. You should include:

   (a) a paper design,
   (b) a sketch of the soundness proof for the type system,
   (c) an informal discussion of why your language satisfies the correspondence principle,
   (d) a Haskell implementation.
   (e) Eventually, I will ask you to provide a page or two of English explaining what you learned from this project, and what interesting aspects are found in its semantics. So you may want to start that now. But this is optional at the moment.

   As in homework 1, do the paper version first, with the concrete syntax, abstract syntax, type attributes, type checking rules, example type proofs, semantic domains, and the semantics. Then sketch the soundness proof for the type system and the correspondence property. Finally, code your semantics in Haskell. You may need to iterate and/or interleave these phases as you see fit. I'm happy to give feedback on ideas or partial designs.

   This project should be a pretty small language. It must include some kind of parameterization, but, to keep things small, don't include every kind. Plan on extending it when we get to chapter 4 with local declarations, so omit that for now.

   I would suggest that you extend (one of) your project(s) from the last problems of homeworks 1 and 2. However, if you'd rather, you can do something else. Here are some ideas for languages

   - Tempting though it may be, I suggest that you avoid languages with concurrency or distribution. We haven't covered the semantic ideas needed for such languages.
   - Make a selection from chapter 3 of Schmidt's book, and implement that building on the chapter 2 programs that I provide. If you do this, you should include arrays as well (see exercise 4.3 in the book), and also parameter lists (see section 3.8). You should also consider exercises 7.1 (especially parts d and e). I suggest that you do not include type structure parameters (see section 3.6 for why).
   - Martín Abadi and Luca Cardelli's object calculi are simple but can be extended to interesting languages (see the course web page for links, and also refer to sections 4.7–8 of the book).
   - The (object-oriented) language from the book *Essentials of Programming Languages* by Friedman, Wand, and Haynes.
   - Giuseppe Castagna's papers on the lambda-& calculus, which is also simple but could be extended to something more interesting (see the course web page for links).
   - Craig Chambers's papers on Cecil (see the course web page for links) or our own work (with Chambers) on BeCecil (see me for details).
   - Microsoft's Component Object Model (COM) and Object Linking and Embedding (OLE) might provide something to extend to a language (see the course web page for links).
   - Problem 10.2 in chapter 3 suggests a way to extend the language of boolean expressions to something more like Prolog.
   - You might do a language "inspired" by some existing language, by applying the principles to the core of that language. For example, take Java, boil it down, then apply the principles.
   - Kishore Dhara's recent Ph.D. thesis has a language that would be neat to implement. See me for details.