
Suggestions for Projects

Mike Spivey, September 2003

This list contains a number of suggestions for projects related to the lecture courses I give, to my current research, and to random ideas that I think it would be interesting to implement. Many of the projects are suitable for several purposes: some would even be suitable either as undergraduate or as M.Sc. projects, with differences in scope and emphasis.

The projects differ significantly in their technical difficulty. Some require you to use little in the way of tools. Others involve the use of software tools (like theorem provers) that are almost guaranteed to drive you to distraction. Some general observations:

- I won't supervise projects whose goal is to "specify an X in Y ", for some values of X and Y (including the values $X = X$ and $Y = Z$, and especially the value $Y = \text{UML!}$). These projects lack a concrete focus and often end up with a specification of something that is significantly less than X in a notation that is not quite Y .
- The project descriptions listed here are just suggestions, and an indication of the kinds of project that interest me. If a different project along the same lines appeals to you, please do talk to me about it. In any case, most people start a project with one set of plans and end it having done something different. This is a very healthy sign that our projects involve an exciting element of research.
- Few of the project descriptions mention an implementation language, although for some of them the choice is pretty clear. The choice of implementation technology is a matter for discussion and agreement.

Many of these projects concern extensions and adaptations to the picoPascal compiler that is used as a source of examples in my course on Compilers, or the very similar Oberon compiler OBC that we use for teaching programming. You don't have to have taken the course to do these projects, but it certainly helps a lot. I've suggested this framework as a host for various ideas in language implementation because it is simpler than commercial frameworks like Java and .NET, but much of the experience gained here applies also in those more complex settings.

1 Better garbage collection for OBC

Traditionally, Pascal-like languages have pointers and dynamically-allocated storage, but they do not have a garbage collector. This makes programming with pointers unnecessarily difficult, because each program must keep careful track of all the storage that has been allocated, in order that it can be released later. Oberon breaks this mould by requiring that heap-allocated storage be garbage collected.

At present, OBC uses a relatively simple garbage collector that scans the entire heap on each collection, and can cause long pauses in execution. The aim of this project is to investigate more sophisticated garbage collection techniques such as generational and incremental collection, and to implement and evaluate one of them for use with OBC.

Reading: Jones and Lins, *Garbage Collection*.

2 Animating object code

One of the big difficulties in understanding how compilers work is to see how the bits and bytes present in run-time memory are related to our ideas about the actions of a high-level program. The aim of this project is to build a teaching tool that shows the execution of an object program on a hypothetical machine, and interprets the run-time data structures in terms of what they mean in high-level terms.

3 A machine-independent code generator

In their book *A retargetable C compiler*, Fraser and Hanson describe a scheme for generating object code that is based on matching patterns that describe instructions against expression trees that represent the program to be compiled. The aim of this project is to produce a clean implementation of this idea using ML as the implementation language.

A suitable starting point would be the postfix code that is output by the front end of the picoPascal compiler. Your code generator could reconstruct expression trees from this, then use the pattern-matching technique to produce near-optimal code from them.

Alternatively, the abstract machine that underlies OBC is designed so that procedures compiled to bytecode can coexist with native-code procedures. It would be possible to design a JIT-style code generator that is able to input bytecode and translate it into native code that performs the same task.

4 A debugger for Oberon

The aim of this project is to add a source-level debugger to the bytecode execution environment of OBC. The OBC compiler outputs symbol information for the programs it compiles. The aim of this project would be to implement a program that works alongside the OBC bytecode interpreter and uses the symbol information to display the values of variables in a running Oberon program, allowing the program to be executed line by line, and allowing the

programmer to set breakpoints where the program will stop.

5 Fun in Oberon

My course on Programming Languages begins with a simple functional language called Fun that is implemented by an interpreter written in Haskell. Initially, this interpreter uses the higher-order features of Haskell to implement similar features of Fun, but it is possible to eliminate the use higher-order features from the interpreter by systematically introducing explicit representations such as closures. The aim of this project is to implement Fun in Oberon, either by means of a source-level interpreter that uses explicit closures, or by means of a compiler that translates Fun into an abstract machine code which is then interpreted.

An extension of this project might compile Fun into the same abstract machine code that is used in OBC. Any of these Fun implementations would provide an excellent benchmark for the OBC garbage collector (see Section 1). Alternatively, substitute Java for Oberon throughout!

6 A minimal object-oriented language

Smalltalk is an extremely simple object-oriented programming language in which literally everything is an object. The aim of this project is to construct an implementation of a minimal Smalltalk-like language, perhaps as a source-level interpreter in the style of my course on Programming Languages, and to use it to run some example programs that illustrate aspects of object-oriented programming.

Reading: Tim Budd, *A Little Smalltalk*, Addison-Wesley, 1987.

7 Graphical call-graph profiling

In call-graph profiling, information about the time and other resources consumed by subroutines in a program is related to the call-graph of the program, in which the nodes are subroutines and the arcs show the relationship between each routine and the other routines it calls. Recently, I have developed a technique for instrumenting programs so as to gather more accurate and extensive profiling information than has so far been available.

The aim of this project is to create a graphical tool that assists with the presentation and exploration of this profiling data. The starting point is a text-based application that produces a profiling report in a fixed format. The graphical tool can share the parts of this application that read and analyse the profiling data, but will substitute an interactive, graphical display for the textual report.

Reading: J. M. Spivey, 'Fast, accurate call-graph profiling', accepted for publication in *Software - Practice and Experience*, <http://spivey.orient.ox.ac.uk/mike/profiling.pdf>.

8 A linear programming assistant

Linear programming is a technique for posing and solving optimization problems that covers many situations of commercial importance. However, many of its potential users may be discouraged by the fact that their problem has to be reduced to a system of linear equations and inequalities before linear programming can be used to solve it; they may prefer to think in terms of flows and conversion factors that are closer to what happens in their industrial plant.

The aim of this project is to design a systematic way of representing linear programming problems graphically, and to build a prototype environment for creating these graphical representations of linear programs, converting them to systems of equations, and solving the resulting problems.

It might be a good idea to use a toolkit like Tk for implementing the graphical part. It would be nice to couple this to a clear implementation of the classical simplex algorithm, perhaps written in a functional language like Gofer.