UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Computer Science Tripos Part II

Optimising Compilation

http://www.cl.cam.ac.uk/Teaching/2000/OptComp/

Alan Mycroft am@cl.cam.ac.uk

2000–2001 (Lecture on Decompilation)

Part D: Decompilation and Reverse Engineering

This final lecture considers the topic of *decompilation*, the inverse process to compilation whereby assembler (of binary object) files are mapped into one of the source files which could compile to the given assembler.

Note in particular that compilation is a many-to-one process—a compiler may well ignore variable names and even compile `x<9` and `x<=10` into the same code. Therefore we are picking a *representative* program.

There are three issues which I want to address:

- The ethics of decompilation;
- Control structure reconstruction; and
- Variable and type reconstruction.

You will often see the phrase *reverse engineering* to cover the wider topic of attempting to extract higher-level data (even documentation) from lower-level representations (such as programs). Our view is that decompilation is a special case of reverse engineering.

Ethics

Well it's OK sometimes ... . Note that local laws vary (so do it where it is legal!).

Control Structure Reconstruction

Extracting the flowgraf from an assembler program is easy. The trick is then to match *intervals of the flowgraph* with higher-level control structures, e.g. loops, if-the-else. Note that non-trivial compilation techniques like loop unrolling will need more agressive techniques to undo. Cifuentes and her group have worked on many issues around this topic.

Variable Reconstruction

This is trickier than one might first think, because of register allocation (and even CSE). A given machine register might contain, at various times, multiple user-variables and temporaries. Worse still these may have different types. Consider

```
f(int *x) { return x[1] + 2; }
```

where a single register is used to hold `x`, a pointer, and the result from the function, an integer. Decompilation to

```
f(int r0) { r0 = r0+4; r0 = *(int *)r0; r0 = r0 + 2; return r0; }
```

is hardly clear. Mycroft uses transformation to SSA form to undo register colouring and then type inference to identify possible types for each SSA variable.

Further reading

References

[1] Cifuentes, C. et al. "The decompilation page".
URL `http://www.csee.uq.edu.au/~csmweb/decompilation/`

[2] Cifuentes, C. "Reverse compilation techniques". PhD thesis, University of Queensland, 1994.
URL `http://www.csee.uq.edu.au/~csmweb/dcc.html`

[3] Mycroft, A. Type-Based Decompilation. Lecture Notes in Computer Science: Proc. ESOP'99, vol. 1576: 208–223, Springer-Verlag, 1999.
URL `http://www.cl.cam.ac.uk/users/am/papers/esop99.ps.gz`