Validating Programs with Examples

Philip Zigoris
CMPS203 Project, Fall 2004
Motivation for Project

How do we know a program is correct?

- Hoare Triples
- Informally verify
- **Test with examples**

Given a program and some set of $m$ input/output pairs, what can be said about the correctness of the program? Or rather, the probability it is correct. And how dependent are these results on the language itself?
We consider these variants of IMP:

1. ARITH(n,k): standard arithmetic and assignment operations
2. ARITH_BOOL(n,k): add boolean operations to ARITH (effectively including if/then/else).
3. IMP(n,k): add while
4. IMP_t(n,k): IMP programs with running time bounded by t

Also include return statement in each language. Bound size of programs by n, size of inputs by k.

Simplify analysis (a lot) if constrain return values to boolean (in case of ARITH we simply allow expressions to be compared to 0 for inequality).

Begs more questions: what if we can return numerical values, arrays of values, objects...
What is our notion of program size?

Natural notion of program size is the length of string representing it. Then each language only includes a finite number of programs (albeit, an exponentially large number).

Notion we use is the number of nodes in the program AST. Now even ARITH has an infinite number of programs (but that's ok).
What now...

How can we formally capture the complexity of these languages? It is clear that they are capable of representing different things....

We borrow ideas from the wonderful world of learning theory.
Think of programs as a function from $\mathbb{R}^k \rightarrow \{0,1\}$. Given a set of inputs, how many different ways can we label them?

Example: $k=2$, language $\text{CIRCLE} = \{h_{(x,y,r)}\}$

What is the maximum size set $S$ that can be labeled in all $2^{|S|}$ ways? This is the idea behind the VC dimension. If no such set exists, $\text{VCDim} = \infty$.

Example: $\text{VCDim(CIRCLE)} = 3$
Example: $\text{VCDim(RECTANGLE)} = 4$
What does the VCDim tell us?

Characterizes uniform convergence. Essentially, if our language has finite VC dimension then the probability that a program is wrong decreases exponentially as we check it on more and more examples (beyond a certain threshold).

Knowing the VC dimension allows us to make a statement of the form: The probability that the program will make a mistake during $m$ runs is less than...

If VC dimension is infinite then we're SOL (well...)
Analysis of Various Languages

Some Results:

\[
\begin{align*}
VCDim(\text{ARITH}(n,k)) & \leq 2n(n + \log(8e)) \in O(n^2) \\
VCDim(\text{ARITH BOOL}(n,k)) & \leq 2n(n + \log(8ne)) \in O(n^2) \\
VCDim(\text{IMP}(n,k)) & = \infty \\
VCDim(\text{IMP}_t(n,k)) & \leq 2n(2t + \log(8e)) \in O(nt)
\end{align*}
\]

Observations:
1. Adding boolean operations to our language doesn't make it 'more complex'.
2. And in some sense, neither does \texttt{while}.
3. VC dimension grows reasonably with size of program.
Essentially, we can calculate polynomials with these languages. Bounding the running time bounds the degree of the polynomials we are able to calculate.

If unbounded then we could calculate a fractal to an arbitrary degree of precision, thereby shattering any finite size set (hence $\text{VCDim}=\infty$).
Recursive Checkerboard

depth=3
But don't get too excited…

Bounding running time isn't sufficient for a finite VC dimension.

Consider adding trigonometric functions to the list of atomic operations: VC dimension is infinite.

Slightly disheartening. There are ways to address this but... it gets really difficult to understand.
Conclusion

For programs that rely on simple operations and whose running time is 'fixed', there is some theoretical justification for extensive testing.

But this justification can break down with the addition of simple operations such as `while` and `sin(x)`.

Corollary: Programs are learnable (up to some precision).