Formal methods

- The failure of proof of correctness to meet its promises caused a heavy decrease in interest in the late 1970's and the 1980's
- There has been a resurgence of interest in formal methods starting in the late 1980's and through the 1990's
  - Mostly due to potential usefulness in specification and a few success stories
  - Still not entirely compelling to me, in a broad sense, but definitely showing more promise
- Key issues to me include
  - Partial specifications ("proving little theorems about big programs instead of big theorems about little programs" –B. Scherlis) and incremental benefit
  - Tool support (making specifications "electric" — D. Jackson) and automated analysis
  - What domains, and applied by whom?

Potential benefits

- Increased clarity
- Ability to check for internal consistency
  - This is very different from program correctness, where the issue was to show that a program satisfied a specification
- Ability to prove properties about the specification
  - Related to M. Jackson's refutable descriptions
- Provides basis for falsification (a fancy word for "debugging")
  - Perhaps more useful than verification

C.A.R. Hoare, 1988

Of course, there is no fool-proof methodology or magic formula that will ensure a good, efficient, or even feasible design. For that, the designer needs experience, insight, flair, judgement, invention. Formal methods can only stimulate, guide, and discipline our human inspiration, clarify design alternatives, assist in exploring their consequences, formalize and communicate design decisions, and help to ensure that they are correctly carried out.

Observation

- From a specification of a small telephone system
  - "…a subscriber is a sequence of digits. Let Subs be the set of all subscribers …
    ...certain digit sequences correspond to unobtainable numbers, and some are neither subscribers, nor are they unobtainable."
- "Only a mathematician could treat the real world with such audacious disdain." —M. Jackson

Model-oriented

- Model a system by describing its state together with operations over that state
  - An operation is a function that maps a value of the state together with values of parameters to the operation onto a new state value
- A model-oriented language typically describes mathematical objects (e.g. data structures or functions) that are structurally similar to the required computer software

Z (“zed”)

- Perhaps the most widely known and used model-based specification language
- Good for describing state-based abstract descriptions roughly in the abstract data type style
  - Real ADT-oriented specifications are generally does as algebraic specifications
- Based on typed set theory and predicate logic
- A few commercial successes
  - I’ll come back to one reengineering story afterwards

Basics

- Static schemas
  - States a system can occupy
  - Invariants that must be maintained in every system state
- Dynamic schemas
  - Operations that are permitted
  - Relationship between inputs and outputs of those operations
  - Changes of states

The classic example

- A “birthday book” that tracks people’s birthdays and can issue reminders of those birthdays
  - There are tons of web-based versions of these now
- There are two basic types of atomic elements in this example
  - [NAME,DATE]
  - An inherent degree of abstraction: nothing about formats, possible values, etc.

Points about Z

- This isn't proving correctness between a specification and a program
  - There isn't a program!
- Even the specification without the implementation has value
- The most obvious example is when a theorem is posited and then is proven from the rest of the specification
  - known' = known $\cup$ {name?}
- The actual notation seems more effective that some others
- The Z is intended to be in bite-sized chucks (schema), interspersed with natural language explanations

Schema calculus: sweet!

- The schema calculus allows us to combine specifications using logical operators (e.g., $\wedge$, $\vee$, $\Rightarrow$, $\neg$)
  - This allows us to define the common and error cases separately, for example, and then just $\wedge$-ing them together
- In some sense, it allows us to get a cleaner, smaller specification

Z used to improve CICS/ESA_V3.1

- A broadly used IBM transaction processing system
- Integrated into IBM's existing and well-established development process
- Many measurements of the process indicated that they were able to reduce their costs for the development by almost five and a half million dollars
- Early results from customers also indicated significantly fewer problems, and those that have been detected are less severe than would be expected otherwise

1992 Queen's Award for Technological Achievement

- "Her Majesty the Queen has been graciously pleased to approve the Prime Minister's recommendation that The Queen's Award for Technological Achievement should be conferred this year upon Oxford University Computing Laboratory.
- "Oxford University Computing Laboratory gains the Award jointly with IBM United Kingdom Laboratories Limited for the development of a programming method based on elementary set theory and logic known as the Z notation, and its application in the IBM Customer Information Control System (CICS) product. …"

...

- “The use of Z reduced development costs significantly and improved reliability and quality. Precision is achieved by basing the notation on mathematics, abstraction through data refinement, re-use through modularity and accuracy through the techniques of proof and derivation.
- “CICS is used worldwide by banks, insurance companies, finance houses and airlines etc. who rely on the integrity of the system for their day-to-day business.”

Other success stories

- There are a few other success stories, too (not all Z!)
  - Ex: Garlan and Delisle. "Formal Specification of an Architecture for a Family of Instrumentation Systems" (1995)
  - Aided Tektronix in unifying their understanding and development processes for a broad range of oscilloscopes, function generators, etc.
- Clarke and Wing. Formal methods: state of the art and future directions. ACM Computing Surveys 28(4), 1996.
- Craigen, Gerhart, Ralston. An International Survey of Industrial Applications of Formal Methods, Volumes I & II (Purpose, Approach, Analysis and Conclusions; Case Studies), NIST, 1993.

Tool support for Z?

- Some commercial, some freeware
- Formatting (handling all those $\Rightarrow \bullet \oplus \Xi \Delta \notin \varnothing \theta$ characters)
  - html extensions
  - ZML
- Type checkers
- Proof editors, proof assistants, provers
- Specification animations
- …

Analyzing specifications

- It is easy to write specifications that are inconsistent
- Daniel Jackson and colleagues have developed a sequence of tools that check Z-like specifications for inconsistencies
- You feed a specification to the tool and it says either
  - Here's a problem, and here's a specific (counter)example of it, or
  - I can't find one (although there may be one)
- Examples include paragraph style mechanisms, telephone switch structures, many more (generally relatively small)
  - Pieces of the ideas appear in Jackson and Chapin. Redesigning Air-Traffic Control: A Case Study in Software Design. IEEE Software, May/June 2000
- His Alloy system is the most recent of these tools

An example (skipping *lots* of steps):

Jackson & Vaziri

```
class List {List next; Val val;}
…
void static delete (List l, Val v) {
  List prev = null;
  while (l != NULL)
    if (l.val == v) {
      prev.next = l.next ;
      return; }
    else {
      prev = l ;
      l = l.next ;
}
```

- Procedure for deleting all elements with a given value from a singly linked list
- Relational formulae are automatically extracted
- Fields of `List` treated as binary relations
  - `next: List` → `List`
  - `val: List` → `Val`

Desired properties of delete

1. No cells are added
   - `l.*next' in l.*next`
2. No cell with value v afterwards
   - `no c:l.*next'|c.val'=v`
3. All cells with value v removed
   - `l.*next' = l.*next-{c|c.val=v}`
4. No cells mutated
   - `all c|c.val = c.val'`
5. No cycles introduced
   - `no c:l.*next|c in c.+next ->`
     `no c:l.*next'|c in c.+next'`

The tool shows that
- Properties 1, 4 , 5 appear to hold
- But not properties 2 and 3
  - 2 fails because the first list cell cannot be deleted
  - Even a simple fix shows another error, in which the last two cells share a value equal to v

Underlying technologies

- The Jackson et al. tools have been based on (primarily) two different technologies
  - Model checking: explicit state space enumeration, BDD-based symbolic model checking
  - Constraint satisfaction (boolean satisfiability): stochastic (WalkSAT), deterministic (Davis-Putnam, SATO, RelSAT)
- They generally use some form of bounded checking based on the *small scope hypothesis*, which "argues that a high proportion of bugs can be found by testing the program for all test inputs within some small scope. … If the hypothesis holds, it follows that it is more effective to do systematic testing within a small scope than to generate fewer test inputs of a larger scope." [Andoni et al.]