Outline

• Announcements:
  – Homework I on web, due Fri., 5PM by e-mail
  – LAST DAY TO ADD/DROP!
• What do I mean by “formal methods?”
• Specification
• Verification
• Practical formal methods
• Language issues

Formal Methods

• As scientists, we should be accustomed to precision
  – You must be able to describe the exact procedures used in your experiment/analysis
  – This is essential for reproducibility
• Reproducibility is equally important for computational work
  – Formal methods are a collection of techniques to describe precisely what a program should do

Importance of Formalism in Scientific Software Development

• A Scary Story:
  – You write a program to implement algorithm X (you think)
  – But, you actually implemented algorithm Y
  – It is possible that two similar algorithms can produce very different results (think chaos)
  – You publish a paper describing your results (from running algorithm X), but in your methods you describe algorithm Y
  – The results are spectacular. You get your Ph.D. and a tenure-track job.
  – However, just as you’re being reviewed for tenure, a grad student tries to repeat your experiments. Based on your paper, she correctly implements algorithm X and gets very different results.
  – Your tenure is denied, no one will hire you, you walk around campus with a "Will code for food" sign while the student takes your position.
Formal Methods

Formal methods can be divided into two steps:
- Specification: a precise statement of what a program (or subroutine) should do
- Verification: a demonstration that the actual program satisfies the specification

Math/Logic is the preferred method
- rigorous
- precise
But, English has its place
- Some would say, English is not formal
- My view: a good English spec. is better than nothing at all
  - You may actually write a spec in English
  - Can include in comments

Specification Methods

Keys to Specification

- Describe the properties of the inputs and outputs
  - Data structures
  - Precision/type
  - Assumptions: sorted? symmetric? ≠0?
- Describe what will happen if assumptions violated
  - return error
  - return null value, NaN, -999
  - throw exception
Formal Specification

- $I \land P \Rightarrow O$
  - $I$ = formal statement about input
  - $O$ = formal statement about output
  - $P$ = program
  - Says that if program $P$ is correct and the input conditions are satisfied, then the output conditions will be true

- This just says what a program should do, but says nothing about how it will get done
  - No details of $P$

Specification

- A specification can take many forms:
  - English: "Given a sorted array of integers an an integer $k$, the routine will return the location of $k$ in the array"
  - Logic:
    $$\forall x, y, z \in \mathbb{Z} \land y \leq x \land (y \neq z) \land \text{binarysearch}(x, y, z) \implies (k \in x \land k = y \lor (k \notin y \land z = -1))$$

Specification

- English: Given real-valued parameters $a$, $b$, and $c$, the routine returns the roots of the quadratic equation
- Math:
  $$(a, b, c \in \mathbb{R} \land a \neq 0) \land \text{roots}(a, b, c, x_1, x_2) \implies$$
  
  $$x_0 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
  
  $$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$
Verification

- Demonstrate (prove) that your program satisfies specification
- Keep the specification in mind as you develop your code
  - Lines or sections of code will establish different parts of the specification
  - Sometimes, it is easier to refine I (strengthen assumptions) than to satisfy it

Formal Verification

- There are rigorous approaches to verification
  - Special types of logic that describe what code does
- Theoreticians have built “theorem provers” that can be used to automate this process
  - Theorem: Specification
  - Proof: Uses logic to establish that the specification (including details of P) is true.

Formal Verification

- This is a really cool idea
  - See NuPRL web sit
    - http://www.cs.cornell.edu/Info/Projects/NuPr/Intro/intro.html
- Not practical for most scientific programs
- Essential for software controlling costly or important actions
  - Airplanes
  - Space probes
  - Stock trades
Free Verification

- Typing is very simple form of verification
- In a strongly-typed language (Java)
  - X=Y is allowed only if X and Y are the same type
  - This is very helpful, but doesn’t come close to guaranteeing that your program is correct:
    ```java
    void BadFunction(int[] big){
        int[] small=new int[5];
        for(int j=0;j<big.length;j++){
            small[j]=2*big[j];
        }
    }
    ```

Practical Formal Methods

- Ideally we would all conduct rigorous specification/verification of our programs
- But who are we kidding?
- My advice,
  - 1) Write a specification as formally as you can, and put an English approximation in the comments
  - 2) As you write your code, prove to yourself that you are actually solving the problem
    - You should include comments like "this line only works if X is true" or "this line makes sure X is true"

A Useful Technique

- A standard technique in formal methods is to search for invariant properties
  - An invariant is a property that doesn’t change
  - If a line of code violates the invariant, next one should reestablish it
Specifying the Model Problem

- English: Given initial distribution of $C$ defined on an evenly spaced grid of $m$ points starting at 0 and ending at $L$ (etc. for $u$, $k$ and reaction), a time step $dt$, and an ending time $T$, RAD1d finds an approximate solution for PDE at time $T$

$$\frac{\partial C}{\partial t} = u \frac{\partial C}{\partial x} + \frac{\partial}{\partial x} \left( k \frac{\partial C}{\partial x} \right) + r(C, x, t)$$

Specifying RAD1d

- We know that $C$ provided by user is an approximate solution at time 0. This suggests an invariant:
  - $C$ is a solution for PDE starting from $C_0$ at current time $t$
- We can use the invariant to help develop the code
  Get $C$ at time 0
  $t=0$ --- establishes invariant at start of loop
  while ($t<T$)
  Build $A$, $b$
  Solve $AC=b$ --- invariant violated, $C$ is sol'n at $t+dt$
  $t=t+dt$ --- invariant reestablished

Language Criteria

- You may not have a choice
  - If you are extending a program written in C, use C!
- You may only know one language
  - This is not a reason, it’s an excuse!
  - But, there’s a lot to be gained by sticking with what you know
Language Criteria

- Several things to keep in mind when picking a language
  - Libraries and legacy code: Can you easily utilize code that is already written, tested & debugged (and hopefully specified)?
  - Portability: is the language standardized so that it is easy to compile and run on several platforms?
  - Elegance: will your program be easier to understand (and debug and extend) in one language
  - Future: will compilers still be available in 10 years? Will future users be comfortable in this language?

Languages for Scientific Computing

- Programming languages can be categorized
  - Procedural: Programs consist of one or more procedures (aka functions or subroutines). Data objects are passed to the procedures (FORTRAN, C, Matlab)
  - Object-Oriented: Programs are composed of several objects that encapsulate related data and the procedures to manipulate them (C++, Java)
  - Functional: Programs are functions that operate on data or other functions—highly recursive (LISP, ML)

Languages for Scientific Computing

- Language phylogeny
FORTRAN

- FORmula TRANslator
- One of the first programming languages
- Most common strain was standardized in 1977
- Designed for "Scientific Computing" (e.g. physics)

FORTRAN

- Types:
  - integer, float, double, complex, char
- Data structures:
  - arrays

FORTRAN: Key Advantages

- complex type fully implemented, integrated
- lots of legacy code
- simple
- fast!
**FORTRAN: Disadvantages**

- F77 is ancient
  - Missing "modern" features like pointers, novel data structures (or even records)
  - Missing not-so-modern features like recursion!
  - Encourages bad programming:
    - heavy use of goto-statement
    - common blocks

**FORTRAN90**

- Modernizes F77 while maintaining backward compatibility
  - Dynamic allocation of arrays (size set at run-time, not at compilation)
  - "Vectorized" operations:
    - \( c = a[:]+b[:] \)
- I'm not a fan (just added stuff to F77), but some folks really like it
  - An attractive option for extending legacy code

**C**

- In many ways, C is similar to FORTRAN
  - Procedural
  - few built-in types and data structures
- But more modern
  - pointers--allows you to manipulate memory directly
  - structs--allows you to implement records
  - Together, pointers and structs allow you to create new data structures
  - supports recursion
C: Key advantages

- Common—good compilers available for all platforms, often for free
- Legacy code—lots of stuff already written
- Good performance—comparable for FORTRAN, especially for simple, array-based code
- Very modular—library-concept tightly integrated through #include directive
- Modern—can do everything you could ever want to do (math, CS, graphics)

C: Key disadvantages

- Programming with pointers can be complicated and even dangerous
- No complex type or functions

Matlab

- Matlab is a "programming environment" for scientific computing
  - Lots of built-in functions
  - Easy to program, especially if you are comfortable with procedural programming
  - Data analysis/visualization tools make it easy to develop/debug code
  - Excellent system for building prototypes, but not suitable for production runs of large, computationally intensive code
Java

- Java is the current standard for object-oriented programming
  - objects are "classes"
    - fields: data
    - methods: functions
  - classes should encompass data-types with functions that operate on them

Java: Key Advantages

- Object-oriented—encapsulation of data and functions simplifies programs, makes management easier
- Popular—lots of available code, especially for graphics and common CS algorithms and data structures
- Standardized—very complete specification of language means that all Java code will run on all Java compilers
  - Several versions though, make sure your compiler is current
- Strongly-typed—many bugs are caught at compile time
- Run-time checks on array bounds avoids "segmentation faults" (returns an intelligent error message)

Java: Key Disadvantages

- Performance: object-oriented languages are complicated, so it is hard for compilers to make smart optimizations (applies to C++, too)
  - Also, standard Java is interpreted, not compiled so optimization is out of the question!
- Main Audience: scientists are no longer the main driving force behind computers.
  - Java’s main audience are commercial developers (especially web)
  - Even so, there is still a lot of scientific code out there
My Advice

- For extending old code, stick with the original language
- For new code, I highly recommend Java
  - code is cleaner, and requires less debugging (reduces development time)
  - Especially true if your work doesn't require heavy computation
- For any project, consider developing a prototype first (Matlab or Java) and then translate to C or FORTRAN for max performance