**Domain-specific Languages**

Languages with constructs to address problems in specific application domains,
- Databases: SQL
- GUI: TCL/Tk
- Command Interfaces: csh, perl
- Networks: Router/Firewall configuration
- ... and many, many more ...

**Program Analysis**

Get compile-time information about run-time behavior of a program,
- **Testing**: Automatic generation of test cases,
- **Verification**: Prove/disprove specific properties of a program,
- **Debugging**: Identify potential programming errors (e.g., memory leaks)
- **Optimization**: Improve time/space performance of programs
- **Security Assurance**: Identify vulnerabilities in a program/system.

**Compiling high-level languages**

1. Translate functional programs to a core language that makes control flow explicit (e.g., continuation passing style)
2. Translate core language to an abstract machine
3. Build an emulator or a machine code generator for the abstract machine,
   - **Functional Languages**: e.g., Scheme, SML, Haskell
     Abstract machines: e.g., Combinator machine, G-machine,
   - **Logic Languages**: e.g., Prolog, Goedel
     Abstract machines: e.g., WAM,
Optimizations

- Branch prediction
- Register allocation heuristics
- Program specialization
- Garbage collection
- Just-in-time code generation
- Incremental compilation

Software Testing

Generate test cases to catch programming/design “bugs”.
Small number of tests to identify a large number of potential problems.

Example: Mutation-based testing.

```c
while (i <= N) {
    while (i < N) {
        ...
    }
}
```

Verification

Given
- system (usually as a high-level specification) and
- a property (such as “never deadlocks”),
prove (or disprove) that the system has that property.
Traditional application areas: hardware design, network protocols,
Approaches:
- Axiomatic proof systems
- Invariant inference
- Model checking