Introduction to Design Languages

Prof. Stephen A. Edwards

This Time

- General ideas behind languages
- Syntax, Semantics, and Models of Computation
- Specification versus Modeling
- Concurrency: Two things at once
- Nondeterminism: Unpredictability
- Types of communication: Memory, broadcasting
- Hierarchy

Syntax

- Formally:
  Language: infinite set of strings from an alphabet
  - DNA (A T G C)
  - Student Transcripts (w1007-02 w1009-01 w4559-02 ...)
  - English (aardvark abacus abalone ...)
  - Verilog (always module ...)

Last Time

- Introduction to the class
- Embedded systems
- Role of languages: shape solutions
- Project proposals due September 26
  - Do you know what you’re doing?

Syntax, Semantics, and Model

- Marionette model
  - You control the syntax
  - The semantics connect the syntax to the model
  - You ultimately affect a model

Computational Model

- What a string ultimately affects
- Does not need to be unique

- DNA Model
  - Proteins suspended in water

- Student Transcripts Model
  - Your knowledge
  - The admiration of others

- English Model
  - Natural language understanding

- Verilog Model
  - Discrete event simulator
  - Netlist of gates and flip-flops
Semantics

- How to interpret strings in the model
- Also not necessarily unique

Semantics
- DNA $[\text{AGA}] = \text{Arginine}$
  $[\text{TAG}] = \text{STOP}$
- Student Transcripts $[\text{wr1007-02}] = \text{Java}$
- English $[\text{Look out!}] = \text{I'm in danger}$
- Verilog $[\text{always @posedge clk}] = \text{FF}$

Defining Syntax

- Generally done with a grammar
- Recursively-defined rules for constructing valid sentences
- “Backus-Naur Form”

```latex
expr ::=
literal
  | expr + expr
  | expr* expr
```

- Not a focus of this class

Defining Semantics

- Operational Semantics
  - Abstract machine (memory, program counter)
  - Each statement advance machine state
  - Closest to implementation

- Denotational Semantics
  - Context domain (memory state)
  - Answer domain (result, I/O behavior)
  - Meaning function: $\text{Program} \rightarrow (\text{Context} \rightarrow \text{Answer})$
  - Much more mathematical
  - Able to deal with recursion and self-reference

Specification and Modeling

- How do you want to use the program?
- Specification languages say “build this, please”

- Modeling languages allow you to describe something that does or will exist
- Distinction a function of the model and the language’s semantics

Specification Versus Modeling

- C is a specification language
  - Semantics very operational
  - Clear how the language is to be translated into assembly language

- Verilog is a modeling language
  - Semantics suggestive of a simulation procedure
  - Good for building a model that captures digital hardware behavior (delays, race conditions, unknown values)
  - Not as good for specification: how do you build something with a specific delay?

Gratuitous Picture

- Chartres Cathedral, France
- Façade a renovation of and earlier Romanesque church (note rounded windows)
- Right tower considered the architectural gem
  - Starts complicated
  - Ends simple, to-the-point

Copyright © 2001 Stephen A. Edwards. All rights reserved
Concurrency

- Why bother?
- Harder model to program
- Real world is concurrent
- Good controller architecture: concurrently-running process controlling each independent system component
- E.g., process for right brake, process for left brake, process for brake pedal

The Challenge of Concurrency

- Synchronization
- How to arbitrate access to shared resources
  - Memory
  - I/O ports
  - Actuators
- Different approaches to concurrency a focus of the course

Approaches to Concurrency

- Shared memory / "Every man for himself"
  - Adopted by Java, other software languages
  - Everything’s shared, nothing synchronized by default
  - Synchronization through locks/monitors/semaphores
  - Most flexible
  - Easy to get wrong
- Synchronous
  - Global clock regulates passage of time
  - Very robust in the presence of timing uncertainty
  - Proven very successful for hardware design
  - Synchronization overhead often onerous

Communication and Concurrency

- Idea: let processes run asynchronously and only force them to synchronize when they communicate
- CAR Hoare’s Communicating Sequential Processes
  - Rendezvous-style communication:
    - Processes that wish to communicate both wait until the other is ready to send/receive
- Kahn Process Networks (later in the course)
  - Communicate through channels
  - Writer always continues
  - Reader waits until data has arrived

Nondeterminism

- Does a program mean exactly one thing?
- Example from C:
  ```c
  printf("%d %d %d", ++a, ++a, ++a);
  
  "Increment a, return result"
  ```
- Argument evaluation order is undefined
- Program behavior subject to compiler interpretation
- Are you sure your program does what you think?

Nondeterministic Is Not Random

- Deterministic: \(1+1 = 2\) always
- Random: \(1+1 = 2\) 50% of the time, 3 otherwise
- Nondeterministic: \(1+1 = 2\) or 3, but I’m not telling
- The nondeterministic behavior could look deterministic, random, or something worse
Nondeterminism Is Awful

- Much harder to be sure your specification or model is correct
- True nondeterministic language difficult to simulate
  - Should produce “any of these” results
  - Must maintain all possible outcomes, which grows exponentially
- Idiosyncrasies of a particular implementation of a nondeterministic language often become the de facto standard

Example From Verilog

- Concurrent procedure execution order undefined
  always @(posedge clk) $write("a")
  always @(posedge clk) $write("b")
- First implementation moved procedures between two push-down stacks. Result:
  a b b a a b b a b b a

Later simulators had to match now-expected behavior

Nondeterminism Is Great

- True nondeterministic specification often exponentially smaller than deterministic counterpart
- Implicit “all possible states” representation
- E.g., nondeterministic finite automata for matching regular expressions
- If system itself is truly nondeterministic, shouldn’t its model also be?
- Can be used to expose design errors
- More flexible: only there if you want to use it
- Correctness remains more elusive

Communication

- Memory
  - Value written to location
  - Value stays until written again
  - Value can be read zero or more times after write
  - No synchronization
- Buffer
  - Value written to buffer
  - Value held until read
  - Values read back in order they were written

Hierarchy

- Most languages have ability to create pieces and assemble them
- Advantage: Information hiding
  - User does not know details of a piece
  - Easier to change implementation of piece without breaking whole system
  - Easier to get small piece right
  - Facilitates abstraction: easier to understand the whole
- Advantage: Reuse
  - Pieces less application-specific; can be used elsewhere

Communication

- Wires
  - May or may not have explicit write operation
  - Value immediately seen by all readers
  - More like a system of equations than a sequence of operations
Summary

- Languages have syntax, semantics, and model
- Syntax usually defined with grammar
- Semantics can be defined operationally or denotationally
- Many possible models: A focus of this class
- You ask for something with a specification language
- You describe something that does or will exist with a modeling language

Summary

- Concurrency useful for handling real world
- Synchronization big challenge
  - Shared memory and locks
  - Synchrony
  - Rendezvous synchronization
  - Buffer synchronization
- Nondeterminism
  - Good for certain models
  - Can be very succinct
  - Makes specification hard
  - Makes verification harder

Summary

- Communication techniques
  - Memory
  - Buffered
  - Wired
- Hierarchy for information hiding