Models of Software Systems

David Garlan
Jeannette Wing
Carnegie Mellon University

15-671
Fall 1995

© Garlan, 1995

Lecture 1
Course Introduction

- Course Mechanics
- Course Rationale & Objectives
- Overview of Course Topics
- What are Formal Models?

General Information

- **Instructors:**

  David Garlan
  WeH 8020
  268-5056
  garlan@cs
  Office: Fri 9:30-10:30

  Jeannette Wing
  WeH 8219
  268-3068
  wing@cs
  Office: W 12–1

- **TA:**

  James Ivers, WeH 4615, 268-5842, jivers@cs, Tue 4-5

- **Secretary:**

  Cary Lund, WeH 8106, 268-3853, cary@cs

Computing and Communication

- You will need an account on a SCS machine.
  - > Fill out form after class if you don’ t already have an account
- On-line materials
  - > URL: http://www.cs.cmu.edu/afs/cs/academic/class/15671-f95/www
  - > afs/cs/academic/class/15671-f95/www
- Class bboard: cmu.cs.class.cs771
  - > Read it regularly, post questions and ideas
- E-mail to instructors is fine anytime
- The Moose Lounge

Readings

- **Textbooks:**
  - > ***Software Engineering Mathematics*, by J.C.P. Woodcock and M. Loomis [SEM]**
  - > ***Concurrent Systems: Formal Development in CSP*, by M.G. Hinchey and S.A. Jarvis [HJ]**
- **Supplementary papers:**
  - > **Distributed as needed.**
- **Reading should be done before class.**
- **See course syllabus for details**

Homework

- **Weekly homework assignments**
  - > **Discussion with others encouraged**
  - > **Final write-up must be your own work**
  - > **Due at beginning of class - see syllabus**
  - > **Pass/no-pass (✔, –)**
  - > **Can redo by next class**
- **Lab exercises as needed**
  - > **To give practice with tools**
  - > **Ungraded**
- **Extra copies of handouts available from Cary Lund, WeH 8106**

Grading

- **Homework exercises (50%)**
  - > About one per week (4% each)
- **Exams (40%)**
  - > Midterm take-home (15%)
  - > Final in-class (25%)
- **Instructor discretion (10%)**
  - > participation in/out of class
  - > overall level of effort

Dates

- **Midterm (take-home)**
  - > handed out on October 11
  - > due back October 13
- **Final exam (in-class)**
  - > date to be determined
- **Two Friday lectures**
  - > Sept 1, Sept 22
  - > same time & place
  - > no class Aug 30 & Oct 2!

Course Rationale

- Engineering depends on use of precise models for characterizing systems
- Benefits:
  - > Abstraction
    - » highlight important details
  - > Analysis and reasoning
    - » understand the things we produce
  - > Communication
    - » unambiguous descriptions
  - > Design underpinnings
    - » expose choices within design space

Relationship to Other Core Courses

- **Methods**
  - > You will be learning to use formal methods, and, in particular, JSP, Z, and CSP.
- **Management**
  - > Some models used to represent/analyze processes
  - > Metric models
- **Analysis**
  - > Many of the analyses are formal
  - > Others build on the foundations of this course
- **Architecture**
  - > You will develop formal models of architectures

Relationship to Studio

- Formal models are particularly important for safety critical systems
- You will have the opportunity to find areas in which models are needed
- In the past, most studios have found direct application of formal modelling skills

Some of the Models and Logics Used in Software Engineering

- Propositional and Predicate Logic
- State Machines
- Abstract State Machines
- Algebraic Models
- Models of Concurrency
- Petri Nets
- Trace models and Process Algebras (CSP)
- Temporal Logics

This Course

- **Part 1: Models of Sequential Systems**
  - > **Propositional and Predicate Logic**
  - > **State Machines**
  - > **Z as a concrete notation for state machines**
  - > **Module interface notations**
  - > **Algebraic Models**
- **Part 2: Models of Concurrency**
  - > **Parallel state machines**
  - > **Trace models and Process Algebras (CSP)**
  - > **Petri Nets**
  - > **Temporal Logics**

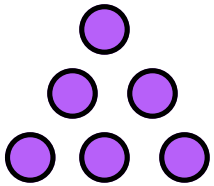
Objectives

- **At the end of the course you should be able to**
  - > **Understand the strengths and weaknesses of the models and logics covered in class.**
  - > **Select and describe formal models for certain classes of systems.**
  - > **Describe relations between different levels of abstraction.**
  - > **Prove elementary properties about systems described by the models covered in the course.**
  - > **Have prerequisite mathematical foundations for the other core courses.**

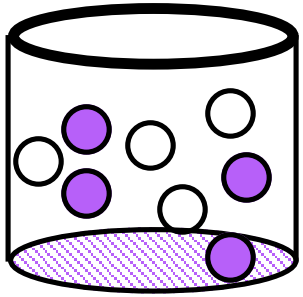
Viewpoint

- Mathematical abstractions (models) can be used to represent real systems
- Formal reasoning can improve our ability to design and build systems
  - > Uncover design flaws
  - > Precisely define requirements
  - > Produce rational implementations
- Different models and logics have different strengths and weaknesses

Example

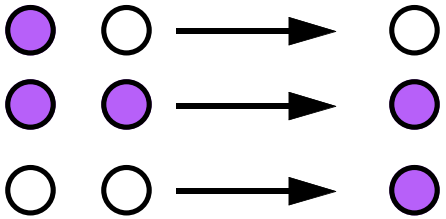


stock of black balls



jar of balls

Rules



The Problem

Question 1: Does the game stop?

Question 2: If so, can you say anything about the color of the last ball by knowing the original configuration?

Testing

A Formal Model

b = black w = white f = transition function

$$f(b, w) = \begin{cases} (b - 2 + 1,\ w) \\ (b - 1,\ w - 1 + 1) \\ (b + 1,\ w - 2) \end{cases}$$

$$f(b, w) = \begin{cases} (b - 1,\ w) \\ (b - 1,\ w) \\ (b + 1,\ w - 2) \end{cases}$$

$$f(0, 1) = (0, 1) \qquad f(1, 0) = (1, 0)$$

Example : Reasoning

Theorem 1: The game stops.

Theorem 2:

f* (b, w) = if odd (w)
then (0, 1)
else (1, 0).

(Qn: How might you prove this?)

Moral

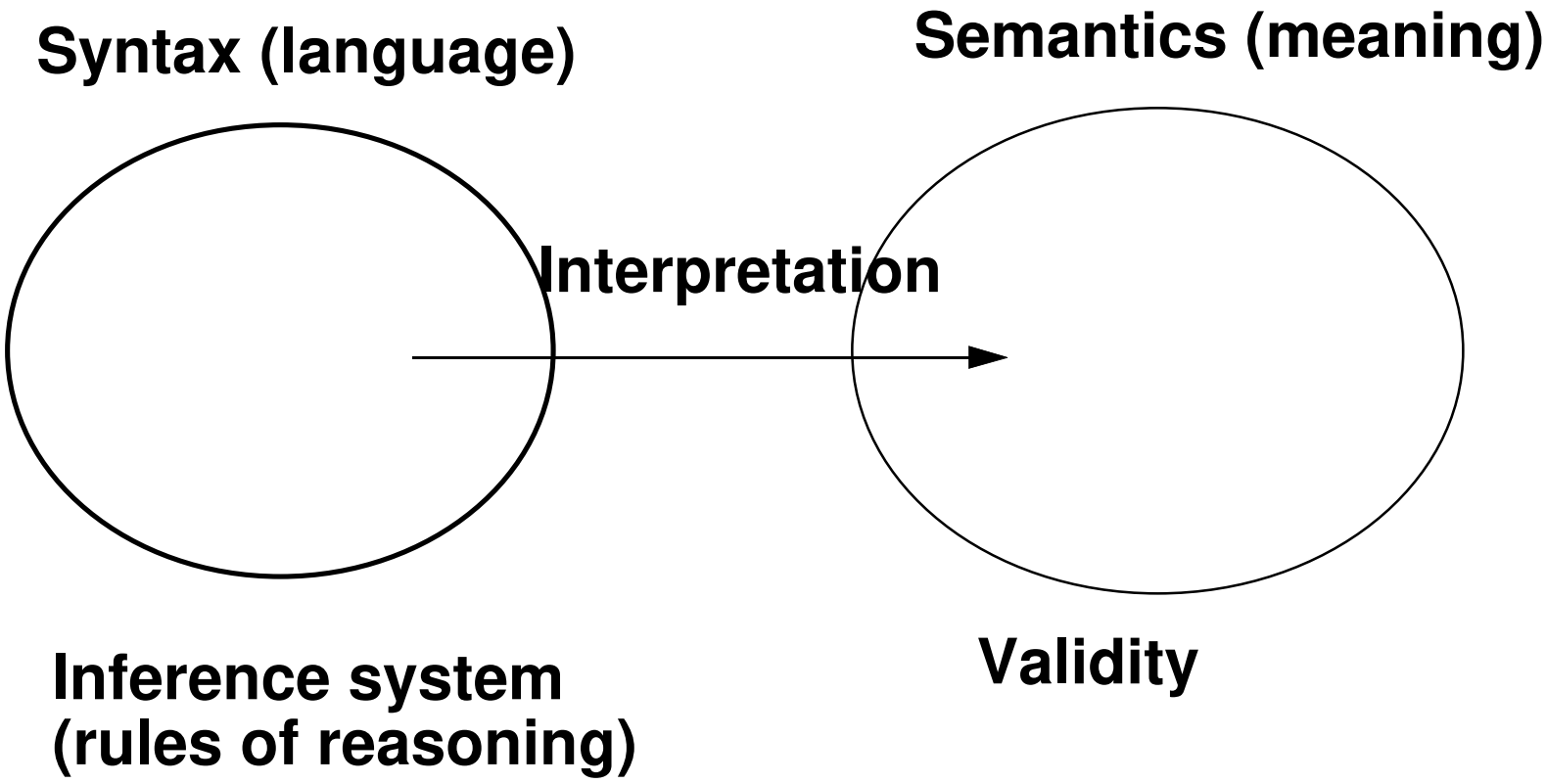
- **Abstraction allows you to focus on the important aspects of a problem.**
- **Mathematics allows you to reason about solutions to the problem.**
- **(You can earn lots of money if you know the right mathematics.)**

What is a Formal System?

Informally the basic elements are:

1. A *language* for describing things of interest
   example: (b, w) = (b - 1, w - 1 +1)
2. Some *rules* for reasoning about "statements" in the language
   example: (b - 1, w - 1 +1) = (b - 1, w)
3. Ways of *assigning meaning* to these statements
   example: b = number of black balls
4. A *body of results* that have already been demonstrated
   example: facts about numbers
   if b is even then b-2 is even

Basic Elements



Syntax: Formal Languages

- All formal systems start by giving

  1. an *alphabet* -- the symbols to be used

  2. syntactic *rules* -- a set of rules for constructing sentences in that system.

- Legal sentences are sometimes called the *well-formed formulae* (wff) of the language

- Examples of wffs:

  Sets: $(A \cup B) \cap (A \cup C)$

  Propositional logic: $(p \Rightarrow q) \wedge (\sim p \vee q)$

  Predicate Logic: $\forall$ x:S • Stooge(x)

Example

Two formal languages with the same alphabet but different syntactic rules:

1. language of real numbers

- **alphabet: {0,1,2,3,4,5,6,7,8,9,.}**
- **some wffs: “345.678” “2.123” “0.421”**
- **not wffs: “3.4.5”**

2. language of section numbers

- **alphabet: {0,1,2,3,4,5,6,7,8,9,.}**
- **some wffs: “3.4.5” “2” “0.421”**
- **not wffs: “.5” “3.”**

(Qn: is “.5” an real number wff?)

Grammars

- Syntactic rules need their own language for description -- called a “meta-language”
- We will use a kind of “BNF”

```
real     = integer | decimal | integer, decimal
decimal  = “.” , integer
integer  = digit | digit, integer
digit    = “0” | “1” | “2” | “3” | ... | “8” | “9”
```


© Garlan, 1995

Semantics

- To give meaning to the symbols we
  - pick a semantic domain
  - provide an interpretation (or meaning function) that explains how sentences in the syntactic domain are given meanings in the semantic domain
- Examples

  Number theory: "1" –> one

  Sets: {1,2} the collection containing the numbers 1 and 2.

  Propositional logic: R –> It is Raining.

  Predicate logic: duck(Martha) –> Martha is a duck.

Inference Systems

- **A deductive apparatus for manipulating wffs.**
  - > **Syntactic manipulation: no reference to any interpretation**
- **Two components:**
  - > **Axioms: wffs that can be written down without reference to any other wffs in the language.**
  - > **Inference rules: rules that allow us to produce wffs as *immediate consequences* of other wffs.**
- **Formal System =**

  language + inference system

Proofs and Theorems

- A *proof* in a formal system, F, is a sequence of wffs in the associated formal language in which each wff is
  - an axiom, or
  - an immediate consequence of one or more preceding wffs (as determined by the inference rules of the system).
- A wff that can be proved (i.e., appears in a proof) in F is called a *theorem*
- If W can be proven, we write: |- W

Example

Let F be the formal system with

alphabet = {*,@,#}

grammar =

sentence = stars, “@”, stars, “#”, stars

stars = “*” | stars, “*”

deductive apparatus =

Axiom: *@*#**

Rule: If a @ b # c is a wff (where a, b, c are stars lists), then a @ b * # c * is an immediate consequence

Example (continued)

Theorem 1.1: * @ ** # *****

Proof:

1. * @ * # **	Axiom
2. * @ ** # ***	Inf rule applied to 1
3. * @ *** # ****	Inf rule applied to 2
4. * @ **** # *****	Inf rule applied to 3

(Qn: can you prove ** @ * # ***** ?)**

Example (continued)

- We can give an interpretation to these symbols:

  * $\rightarrow$ 1

  ** $\rightarrow$ 2

  ...

  @ $\rightarrow$ +

  # $\rightarrow$ =

- Theorem 1.1 means 1 + 4 = 5
- For this interpretation every theorem is true
- Question: can every true addition fact can be proved in this system?

Consistency and Completeness

- **An interpretation of a formal system in which wffs denote statements which can be true or false is**
  - > ***consistent*** **(or *sound*) if every theorem of the system interprets to a true statement**
  - > ***complete*** **if every true statement can be proved as a theorem**
- **Most formal systems we encounter will be sound for the standard interpretations**
- **Many of the systems will not be complete**

Derivations

- (Informal) A derivation of the wff W in formal system F from a set P of wffs (called premises), is a proof in which a wff in the proof can be
  - > an axiom
  - > a premise
  - > an immediate consequence of previous wffs
- A derivation is like a proof in a formal system whose axioms have been augmented with the premise.
- We write P |- W

Example

- **Derive the following:**

  *** @ ** @ ***** |- *** @ *** @ ******

  Derivation:

  1. *** @ ** @ ***** **Premise**
  2. *** @ *** @ ****** **2 Rule 1**

- **Derive the following:**

  * @ ** @ ***** |- * @ *** @ ******

  Derivation:

  1. * @ ** @ ***** **Premise**
  2. * @ *** @ ****** **2 Rule 1**

- **Note: We can prove nonsense from false premises**

For Next Time

- Next class: Friday 10:30-12:00
- Reading [SEM 1-4]
  - > Treats formal systems and proof
  - > Much of this material may be review
  - > Some of proof techniques may be unfamiliar
    - » don't worry if it is
    - » we will cover it partially in next two classes
    - » consider looking at reserve materials in library
- Homework
  - > Will require you to access the Web

Additional Readings (Mostly on Reserve)

- ***A Logical Approach to Discrete Math,*** **by David Gries and Fred B. Schneider**
- ***Discrete Mathematics for Computer Science,*** **by Rajagopal and Mason**
- ***The Science of Progreamming,*** **by David Gries**
- ***Foundations of Computing,*** **by Thierry Scheurer**