Lecture 15
Operational Specifications

Kenneth M. Anderson

Foundations of Software Engineering

CSCI 5828 - Spring Semester, 1999

Today's Lecture

- Continue to discuss the Make example
  - It illustrates each of the three specification styles introduced in lecture 12
- Begin to explore Operational Specifications in more detail

The Make Example

- Lecture 12
  - We worked on an example specifying some properties of Make
- However, Make *is* a specification language itself
  - It specifies dependencies between artifacts
  - It specifies rules for creating new artifacts
  - It specifies actions to carry out the rules

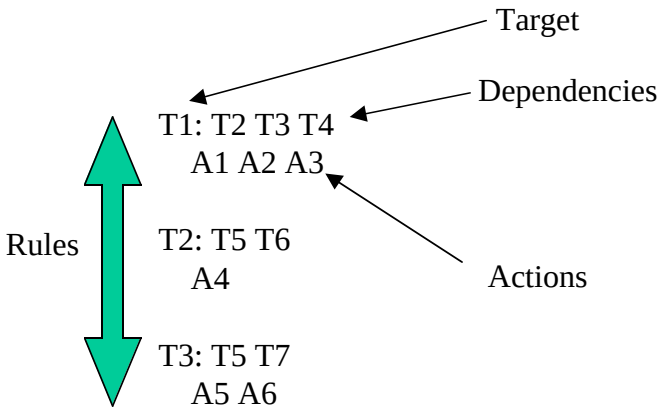
Make Specification Language

- Dependencies are Relational
  - Described according to desired relationships
  - Usually given in terms of multi/hyper graphs
- Rules are Declarative
  - Described according to desired properties
  - Usually given in terms of axioms or algebras
- Actions are Imperative
  - Described according to desired actions
  - Usually given in terms of an execution model

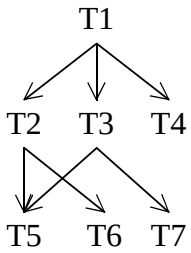
More on Make

- Make is well-integrated into a Unix/C environment
  - Primitive Components are Files
  - Actions are “shell commands”
  - Rules are placed in a file and denote the “specification”
    - Rules make explicit the dependencies of the system and what to do about them

Example “Makefile”



Rules can have interdependencies



… and shared dependencies!

Questions

- What is the concept of dependence in this system? How is it modeled?
- Why are rules considered declarative?

Hybrid Style Issues

- Consider programming languages
  - They are primarily operational
    - What about them are declarative or relational?
- Most languages will have a chief modeling style
  - Contrast statements in a program with Make's
    - S1 S2 S3… operational, do these statements in this order
    - Rules in a makefile: declarative, achieve this target
  - One style will lead you to ask different sorts of questions than with another style
    - Is there a unique way to achieve the target? Is a target feasible?

Operational Specification

- Focuses on Control Aspects
  - Here we choose to look at control issues rather than data issues
- Examples
  - Control the flight path of an airplane
  - Control the speed of a car
- Of course, there are data aspects to these problems. However we view them more as parameters that influence the actions of the system

Formalisms and Foundations

- Formalisms
  - Finite State Machines (FSMs)
  - Petri Nets
  - Statecharts - used in UML
  - Communicating Sequential Processes (CSP)
    - Latter three are different attempts to add concurrency to FSMs
- Mathematical Foundations
  - Graph theory, automata theory, modal logic

Finite State Machines (FSMs)

- Formal Definition

$M = \{Q, I, \delta\}$, where

Q is a finite set of *states*

I is a finite set of *inputs*

δ is a *transition function*

$$\delta : Q \times I \rightarrow Q$$

δ can be a partial function

Finite State Machines (FSMs)

- Graph Representation
  - Nodes represent states
  - Arcs are directed and labeled with element of I
  - Arc labeled i goes from state q_1 to state q_2 iff $\delta(q_1,i) = q_2$

Finite State Machines (FSMs)

- Execution Model
  - Machine in some state
  - Input causes state change according to δ
- Common Extensions
  - *Start* states and *stop* states
  - Output generated upon state transition

Advantages of FSM Model

- Simple
- Obvious graphical representation
- Easy to Build Support Tools
  - Transformers
    - Transform FSM Model into other representations
  - Analyzers
    - Will this FSM run forever? Is it possible for it to halt? Are the state sequences infinite?