Finite Models

Dr. Paul West

Department of Computer Science
College of Charleston

January 23, 2014

Why Models?

- Theoretically, we should test every piece of software thoroughly from the beginning to the end.
- Realistically, we do not have the resources.
- Models provide a means to do analysis of software from the early stages and throughout the software life cycle.

Properties of Good Models

- Compact: representable and manipulable in a reasonably compact form
  - What is reasonably compact depends largely on how the model will be used
- Predictive: must represent some salient characteristics of the modeled artifact well enough to distinguish between good and bad outcomes of analysis
  - no single model represents all characteristics well enough to be useful for all kinds of analysis

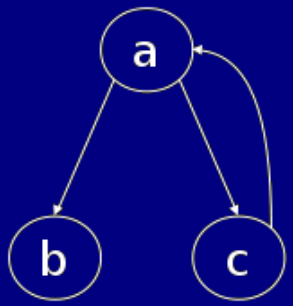
More Good Properties

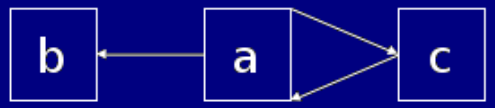
- Semantically meaningful: it is usually necessary to interpret analysis results in a way that permits diagnosis of the causes of failure
  - Example: If a model predicts concurrency failures, the model's description of that failure should aid in producing a solution.
- Sufficiently general: models intended for analysis of some important characteristic must be general enough for practical use in the intended domain of application
  - Example: Modeling C better include pointer analysis!
- Note that UML diagrams are generally what people use as models.

Graph Representations: Directed Graphs

- Directed graph:
  - N (set of nodes)
  - E (relation on the set of nodes) edges

Nodes: {a, b, c}
Edges: {(a,b), (a, c), (c, a)}

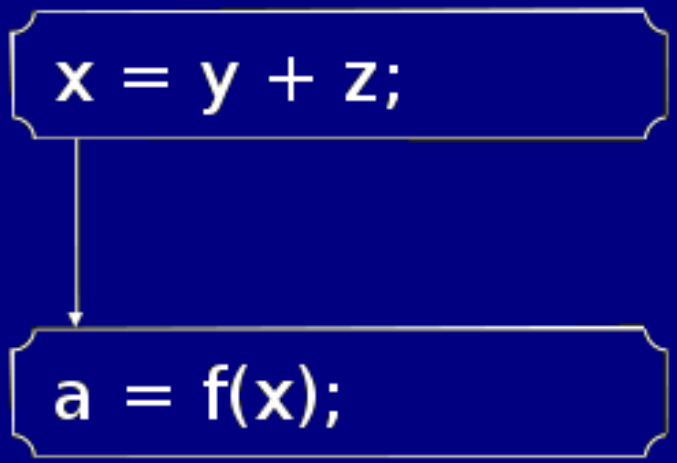




- We are saying either code b or c can follow code a, and code a can follow c.

Graph Representations: Labels and Code

- We can label nodes with the names or descriptions of the entities they represent.
  - If nodes a and b represent program regions containing assignment statements, we might draw the two nodes and an edge (a,b) connecting them in this way:



Multidimensional Graph Representations

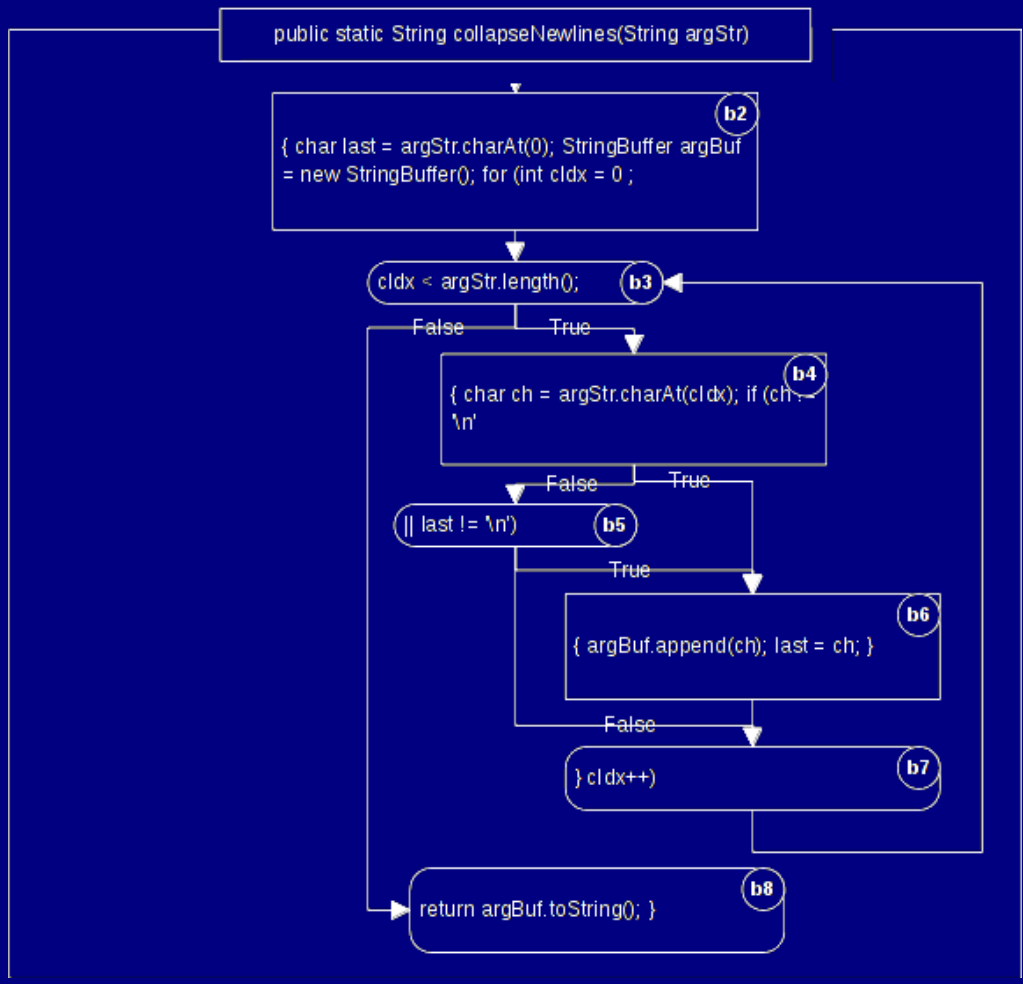
- Sometimes we draw a single diagram to represent more than one directed graph, drawing the shared nodes only once
  - class B extends (is a subclass of) class A
  - class B has a field that is an object of type C

(Intraprocedural) Control Flow Graph

- nodes = regions of source code (basic blocks)
  - Basic block = maximal program region with a single entry and single exit point
  - Often statements are grouped in single regions to get a compact model
  - Sometime single statements are broken into more than one node to model control flow within the statement
- directed edges = possibility that program execution proceeds from the end of one region directly to the beginning of another

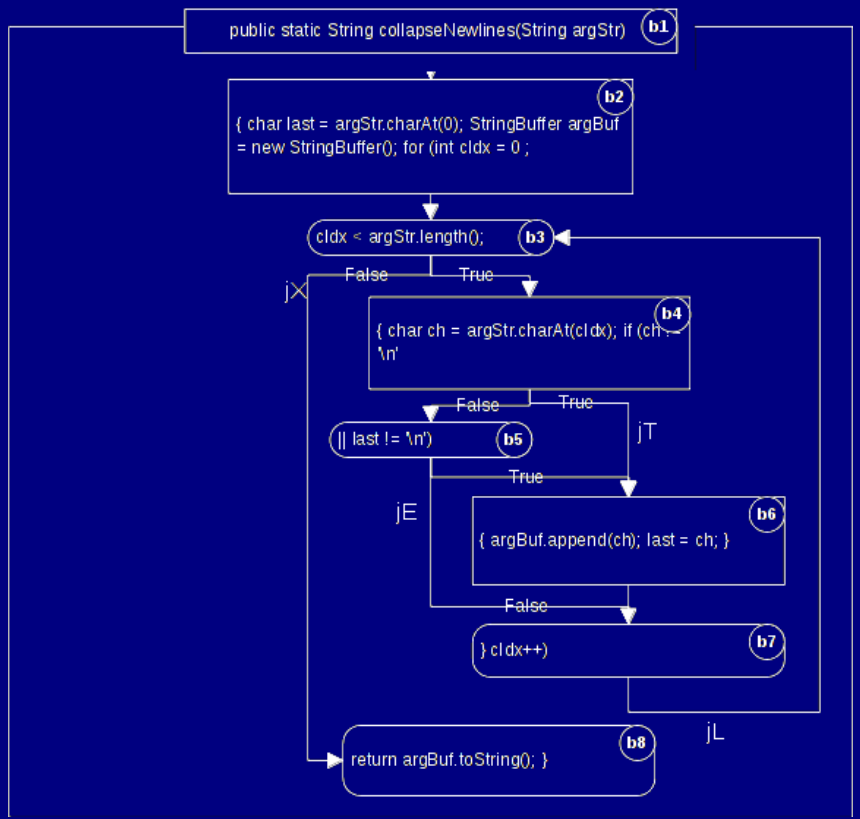
Example of Control Flow Graph

```
public static String collapseNewlines(
        String argStr){
    char last = argStr.charAt(0);
    StringBuffer argBuf = new StringBuffer();

    for (int cIdx = 0;cIdx < argStr.length();
            cIdx++){
        char ch = argStr.charAt(cIdx);

        if (ch != '\n' || last != '\n'){
            argBuf.append(ch);
            last = ch;
        }
    }

    return argBuf.toString();
}
```



Linear Code Sequence and Jump (LCSJ)

Essentially subpaths of the control flow graph from one branch to another



From	Sequence Blocks	To
Entry	b1 b2 b3	JX
Entry	b1 b2 b3 b4	JT
Entry	b1 b2 b3 b4 b5	JE
Entry	b1 b2 b3 b4 b5 b6 b7	JL
JX	b8	ret
JL	b3 b4	JT
JL	b3 b4 b5	JE
JL	b3 b4 b5 b6 b7	JL

Note: Not all paths are illustrated, what about b4:charAt(int)?

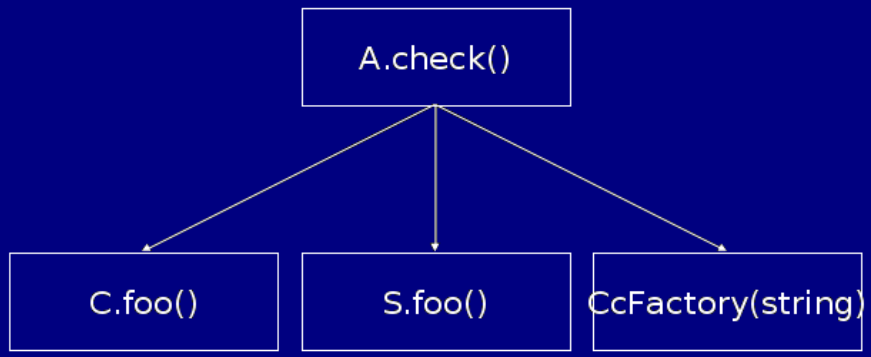
Interprocedural Control Flow Graph

- Call graphs
  - Nodes represent procedures
    - Methods
    - C functions
    - ...
  - Edges represent calls relation
- More difficult in OO languages with inheritance

Overestimating the Calls Relation

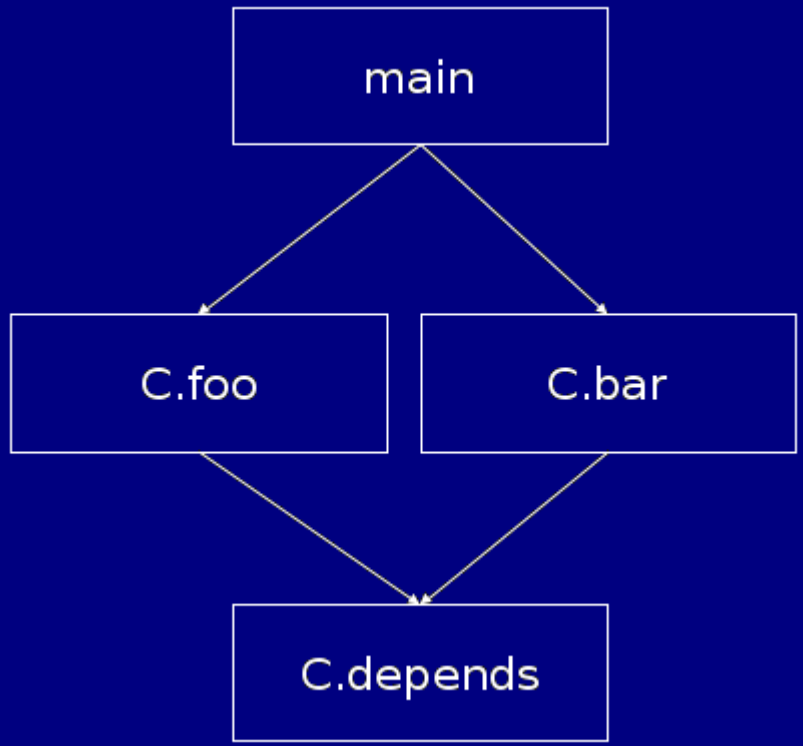
The static call graph includes calls through dynamic bindings that never occur in execution.

```
1   public class C {
2       public static C cFactory(String kind) {
3           if (kind == "C") return new C();
4           if (kind == "S") return new S();
5           return null;
6       }
7       void foo() {
8           System.out.println("Parent");
9       }
10      public static void main(String args[]) {
11          (new A()).check();
12      }
13  }
14  class S extends C {
15      void foo() {
16          System.out.println("Child");
17      }
18  }
19  class A {
20      void check() {
21          C myC = C.cFactory("S");
22          myC.foo();
23      }
24  }
```



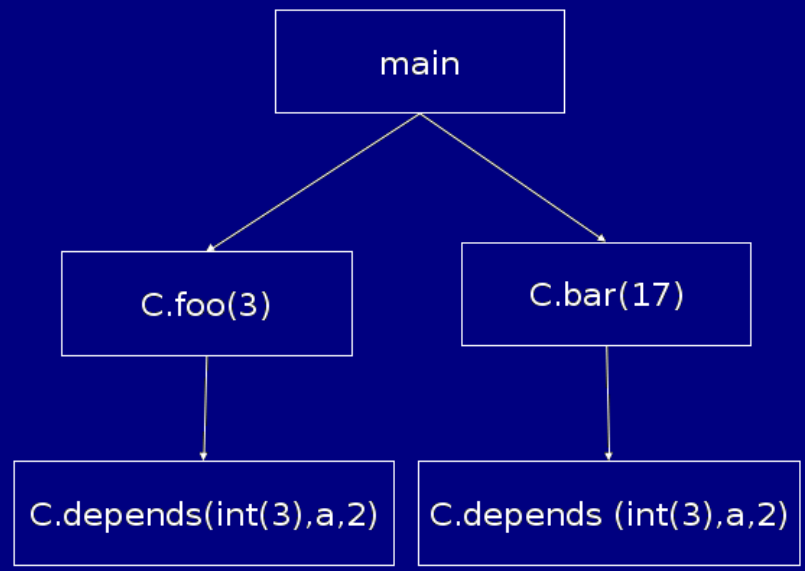
Context Insensitive Call Graphs

```
1   public class Context {
2       public static void main(String args[]) {
3           Context c = new Context();
4           c.foo(3);
5           c.bar(17);
6       }
7
8       void foo(int n) {
9           int[]  myArray = new int[ n ];
10          depends( myArray, 2) ;
11      }
12
13      void bar(int n) {
14          int[]  myArray = new int[ n ];
15          depends( myArray, 16) ;
16      }
17
18      void depends( int[] a, int n ) {
19          a[n] = 42;
20      }
21  }
```

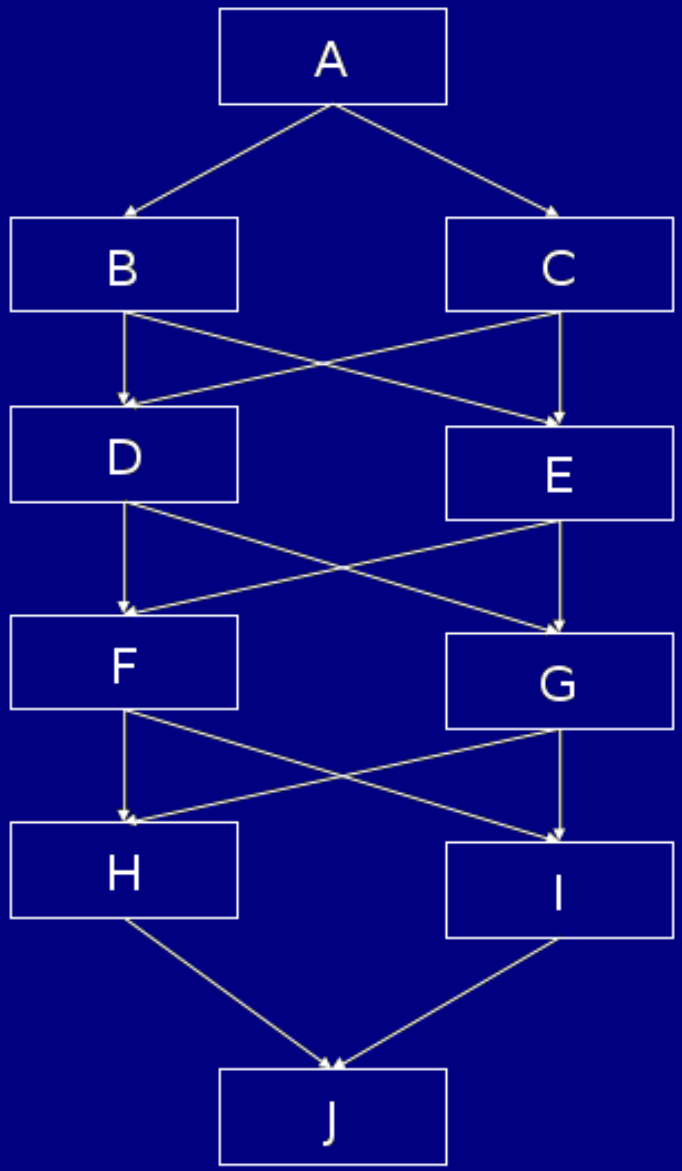


Context Sensitive Call Graphs

```
1   public class Context {
2       public static void main(String args[]) {
3           Context c = new Context();
4           c.foo(3);
5           c.bar(17);
6       }
7
8       void foo(int n) {
9           int[]  myArray = new int[ n ];
10          depends( myArray, 2) ;
11      }
12
13      void bar(int n) {
14          int[]  myArray = new int[ n ];
15          depends( myArray, 16) ;
16      }
17
18      void depends( int[] a, int n ) {
19          a[n] = 42;
20      }
21  }
```



Context Sensitive CFG Exponential Growth



1 context A

2 contexts AB AC

4 contexts ABD ABE ACD ACE

8 contexts ...

16 calling contexts ...

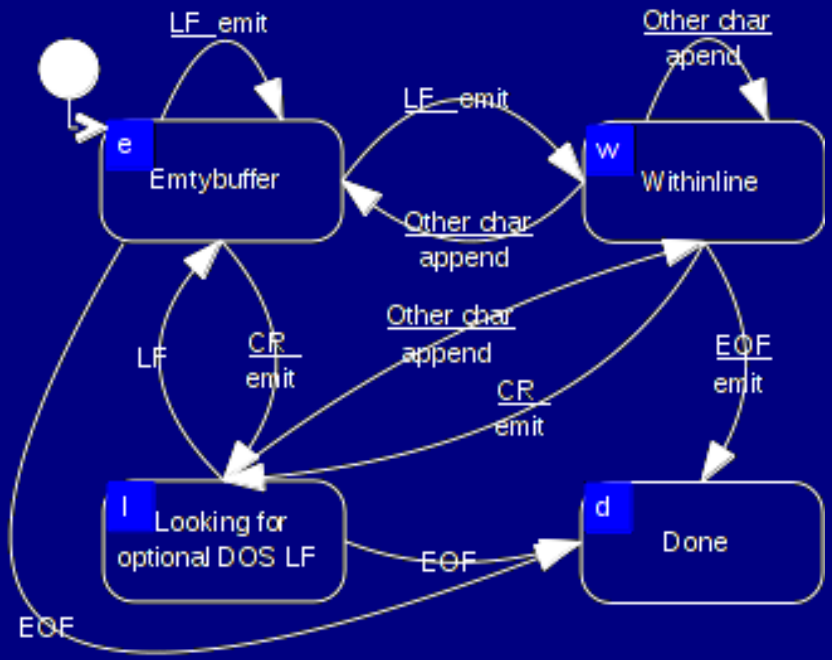
Java Example

- Java Rule: all exceptions should be handled (try-catch blocks) or thrown.
- Java compilers use a CFG to confirm the rule.
- If programmers were not required to throw, then we have the exponential growth (since exceptions can have their own inheritance hierarchy).

Finite State Machines

- finite set of states (nodes)
- set of transitions among states (edges)
- generally created before code and sometimes from use cases

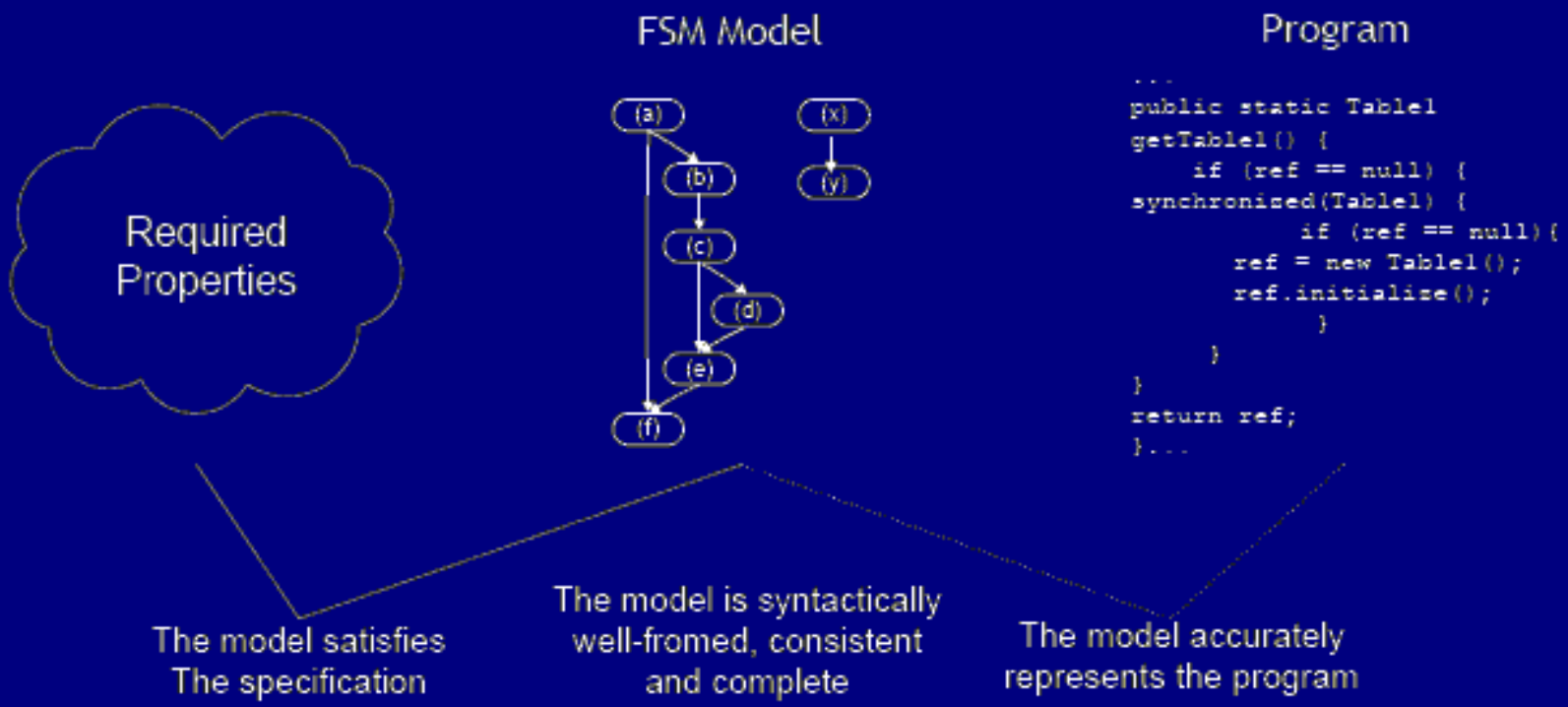
Graph Representation (Mealy machine) Incomplete?



Tabular Representation

	LF	CR	EOF	other
e	e/emt	l/emt	d/-	w/appe
w	e/emt	l/emt	d/emt	w/appe
l	e/-		d/-	w/appe

Using Models to Reason about System Properties



Abstraction Function

```
1  /** Convert each line from stdin **/
2  char buf[1000]; /*Accumulate line buffer*/
3  int pos = 0; /*index for next char*/
4  int atCR; /*0=within line, 1=DOS LF*/
5  char inChar; /*Next char from input*/
6  void transduce(){
7      while((inChar = getchar()) != EOF){
8          switch(inChar){
9              case LF:
10                 if(atCR){/*DOS LF*/
11                 atCR = 0;
12             }else{/*CR within line*/
13                 emit(buf, pos);
14                 pos = 0;
15             }
16             break;
17             case CR:
18                 emit(buf, pos);
19                 pos = 0;atCR = 1;
20             break;
21             default:
22                 if(pos >= BUFLEN-2)
23                     fail("Overflow");
24                 buf[pos++] = inChar;
25         }
26     }
27     if(pos > 0)
28         emit(buf, pos);
29 }
```

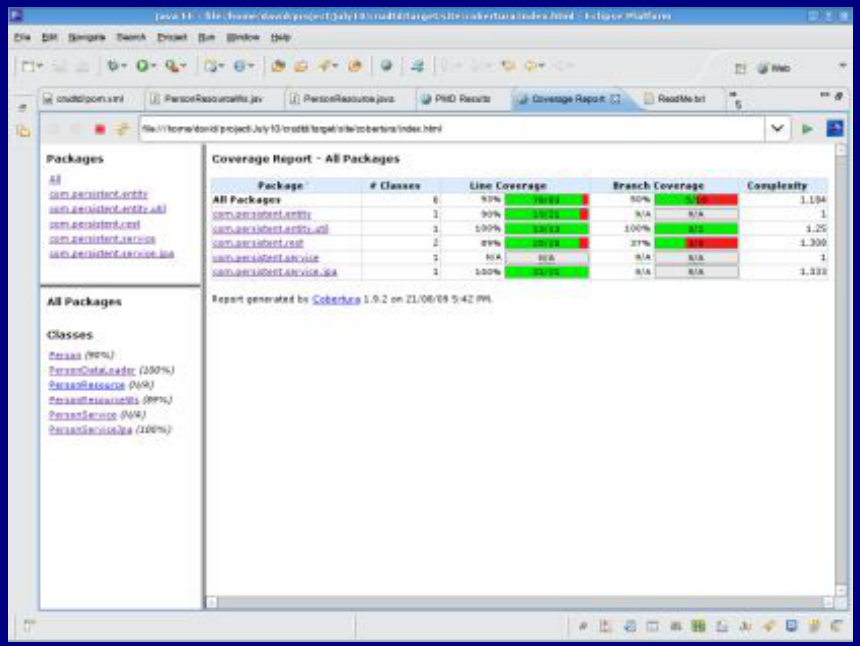
Abstraction function maps a concrete state to a state in the FSM.

	Lines	atCR	pos
e (Empty buffer)	2 - 8	0	0
w (Within line)	8	0	> 0
l (Looking for LF)	8	1	0
d (Done)	29	-	-

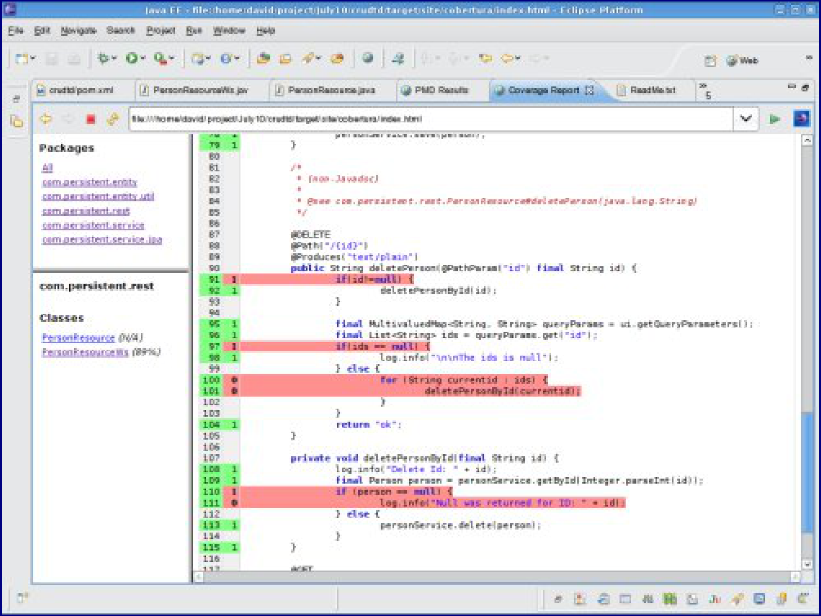
	LF	CR	EOF	other
e	e/emit	l/emit	d/-	w/append
w	e/emit	l/emit	d/emit	w/append
l	e/-	l/emit	d/-	w/append

Summary

- Models must be much simpler than the artifact they describe to be understandable and analyzable
- Must also be sufficiently detailed to be useful
- CFG are built from software
- FSM can be built before software to document intended behavior



- Cobertura will run with unit tests to find which lines of code were tested (and how often).
- While Cobertura does not construct a CFG, it can aid in selecting new tests just like a CFG.
- Taken From `http://persistentdesigns.com/wp/2009/08/quality-control-get-rid`


Java EE - file:/home/david/project/july10/crudtd/target/site/cobertura/index.html - Eclipse Platform
File Edit Navigate Search Project Run Window Help
Web
crudtd/pom.xml
PersonResourceWs.java
PersonResource.java
PMD Results
Coverage Report
ReadMe.txt
file:///home/david/project/July10/crudtd/target/site/cobertura/index.html
Packages
All
com.persistent.entity
com.persistent.entity.util
com.persistent.rest
com.persistent.service
com.persistent.service.jpa
com.persistent.rest
Classes
PersonResource (N/A)
PersonResourceWs (89%)
/*
* (non-Javadoc)
*
* @see com.persistent.rest.PersonResource#deletePerson(java.lang.String)
*/
@DELETE
@Path("/{id}")
@Produces("text/plain")
public String deletePerson(@PathParam("id") final String id) {
if(id!=null) {
deletePersonById(id);
}
final MultivaluedMap<String, String> queryParams = ui.getQueryParameters();
final List<String> ids = queryParams.get("id");
if(ids == null) {
log.info("\n\nThe ids is null");
} else {
for (String currentid : ids) {
deletePersonById(currentid);
}
}
return "ok";
}
private void deletePersonById(final String id) {
log.info("Delete Id: " + id);
final Person person = personService.getById(Integer.parseInt(id));
if (person == null) {
log.info("Null was returned for ID: " + id);
} else {
personService.delete(person);
}
}

Book Assignment

Choose and complete any two Chapter 5 Exercises (pg 74-75).
Due in the dropbox by January 30 2014 2359