Finite State Verification: Models

Dr. Paul West

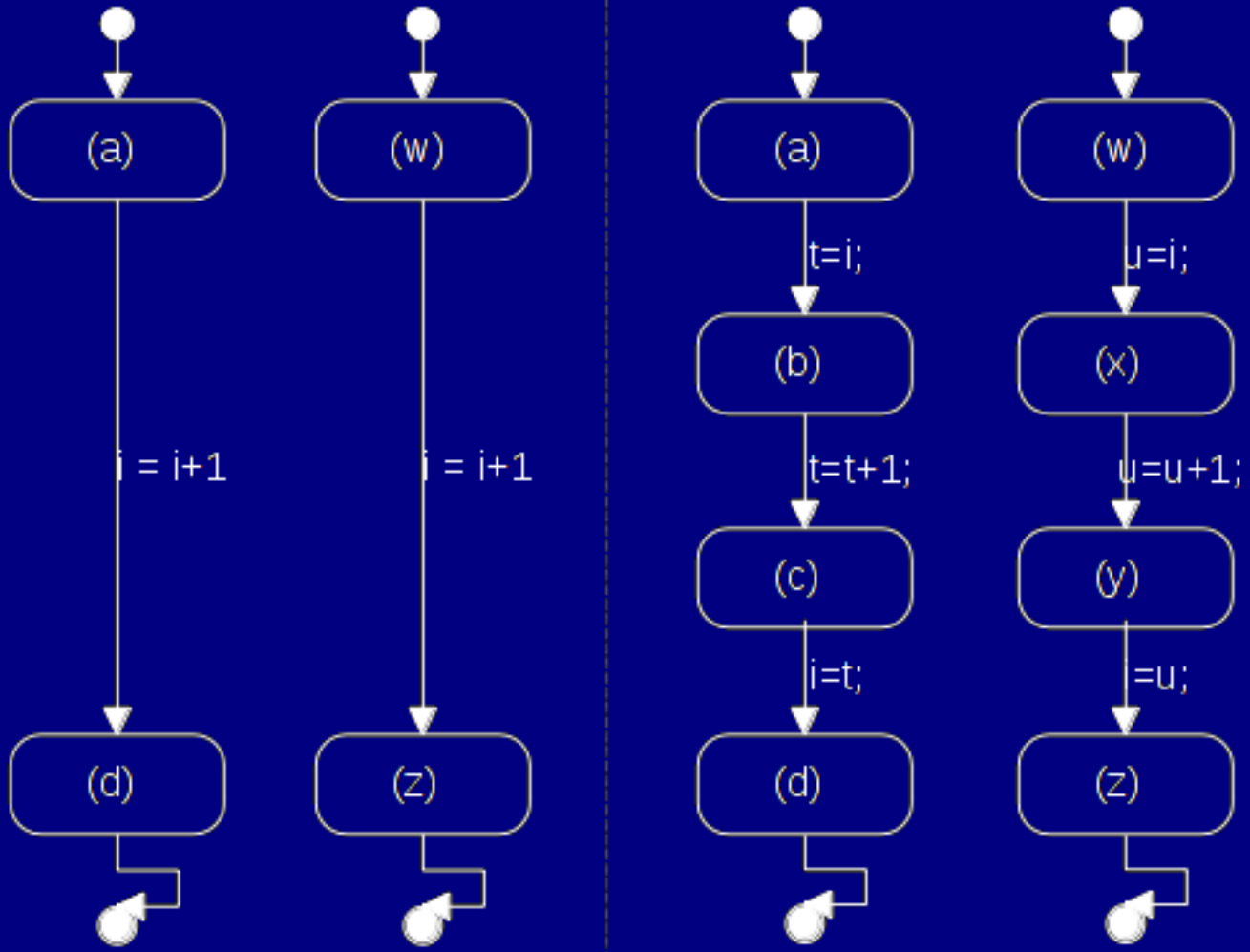
Department of Computer Science
College of Charleston

February 6, 2014

The Model Correspondence Problem

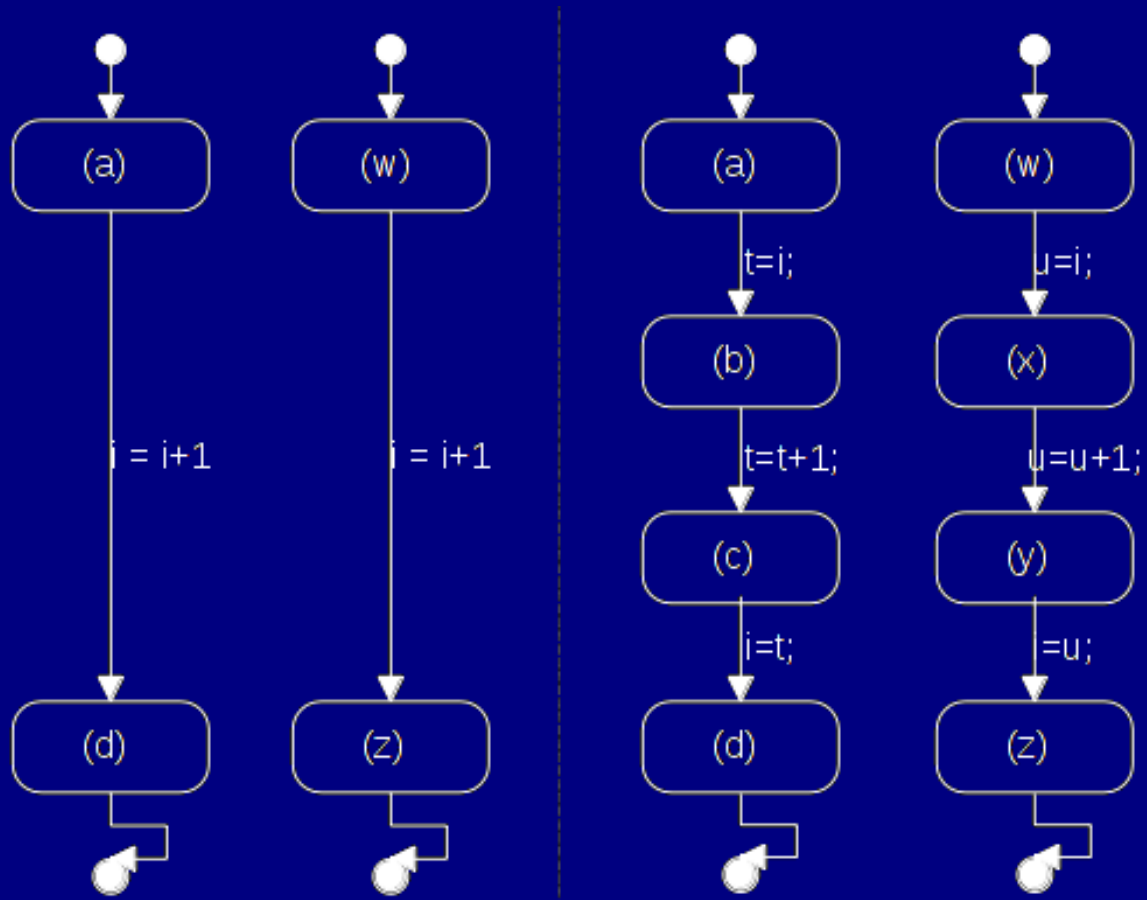
- verify correspondence between model and program:
  - extract the model from the source code with verified procedures
    - blindly mirroring all details state space explosion
    - omitting crucial detail “false alarm” reports
  - produce the source code automatically from the model
    - most applicable within well-understood domains
  - conformance testing
  - good tradeoff

Granularity of Modeling



Analysis of Different Models

we can find the race only with fine-grain models



Looking for the Appropriate Granularity

- Compilers may rearrange the order of instruction
  - a simple store of a value into a memory cell may be compiled into a store into a local register, with the actual store to memory appearing later (or not at all)
  - Two loads or stores to different memory locations may be reordered for reasons of efficiency
  - Parallel computers may place values initially in the cache memory of a local processor, and only later write into a memory area
- Even representing each memory access as an individual action is not always sufficient!

Example

- Suppose we use the double-check idiom only for lazy initialization
- It would still be wrong, but it is unlikely we would discover the flaw through finite state verification:
  - Spin assumes that memory accesses occur in the order given in the Promela program, and ...
  - we code them in the same order as the Java program, but Java does not guarantee that they will be executed in that order

Intentional Models

- Enumerating all reachable states is a limiting factor of finite state verification
- We can reduce the space by using intentional (symbolic) representations:
  - describe sets of reachable states without enumerating each one individually
- Example (set of Integers)
  - Enumeration {2, 4, 6, 8, 10, 12, 14, 16, 18}
  - Intentional rep. $\{xN | x mod 2 = 0 \text{ and } x < 0 < 20\}$
- Intentional models do not necessarily grow with the size of the set they represent

A useful intentional model: OBDD

- Ordered Binary Decision Diagrams
  - A compact representation of Boolean functions
- Characteristic function for transition relations
  - Transitions = pairs of states
  - Function from pairs of states to Booleans:
    - True if the there is a transition between the pair
  - Built iteratively by breadth-first expansion of the state space:
    - creating a representation of the whole set of states reachable in k+1 steps from the set of states reachable in k steps
    - the OBDD stabilizes when all the transitions that can occur in the next step are already represented in the OBDD

From OBDDs to Symbolic Checking

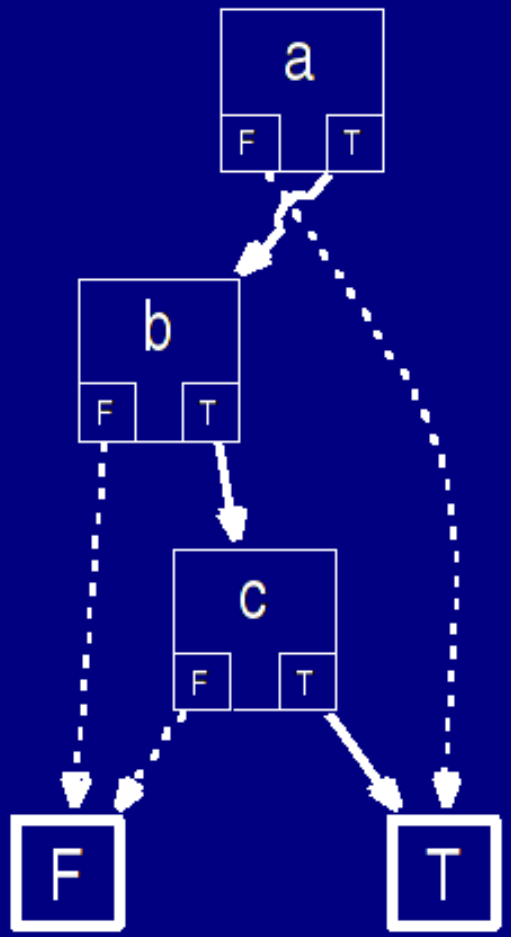
- An intentional representation is not enough
- We must have an algorithm for determining whether that set satisfies the property we are checking

Example:

- OBDD to represent
  - the transition relation of a set of communicating state machines
  - a class of temporal logic specification formulas
- Combine OBDD representations of model and specification to produce a representation of just the set of transitions leading to a violation of the specification
  - If the set is empty, the property has been verified

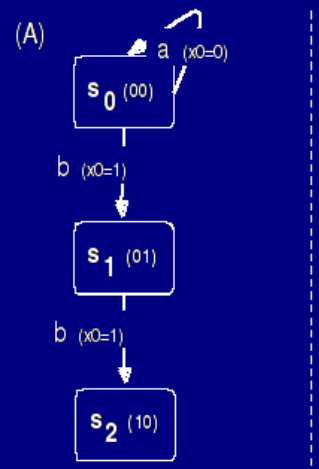
Represent Transition Relations as Boolean Function

a -> b and c
not(a) or (b and c)
the BDD is a decision tree that has been transformed into an acyclic graph by merging nodes leading to identical subtrees



Representing Transition Relation as Boolean Functions

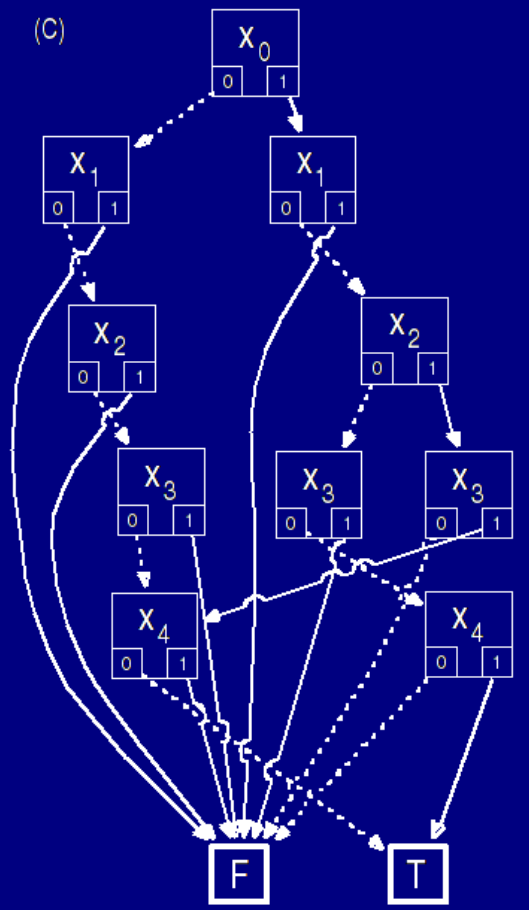
1. Assign a label to each state
2. Encode transitions
3. The transition tuples correspond to paths leading to true; all other paths lead to false



(B)

x_0	x_1 x_2	x_3 x_4
0	0 0	0 0
1	0 0	0 1
1	0 1	1 0

sym from state to state



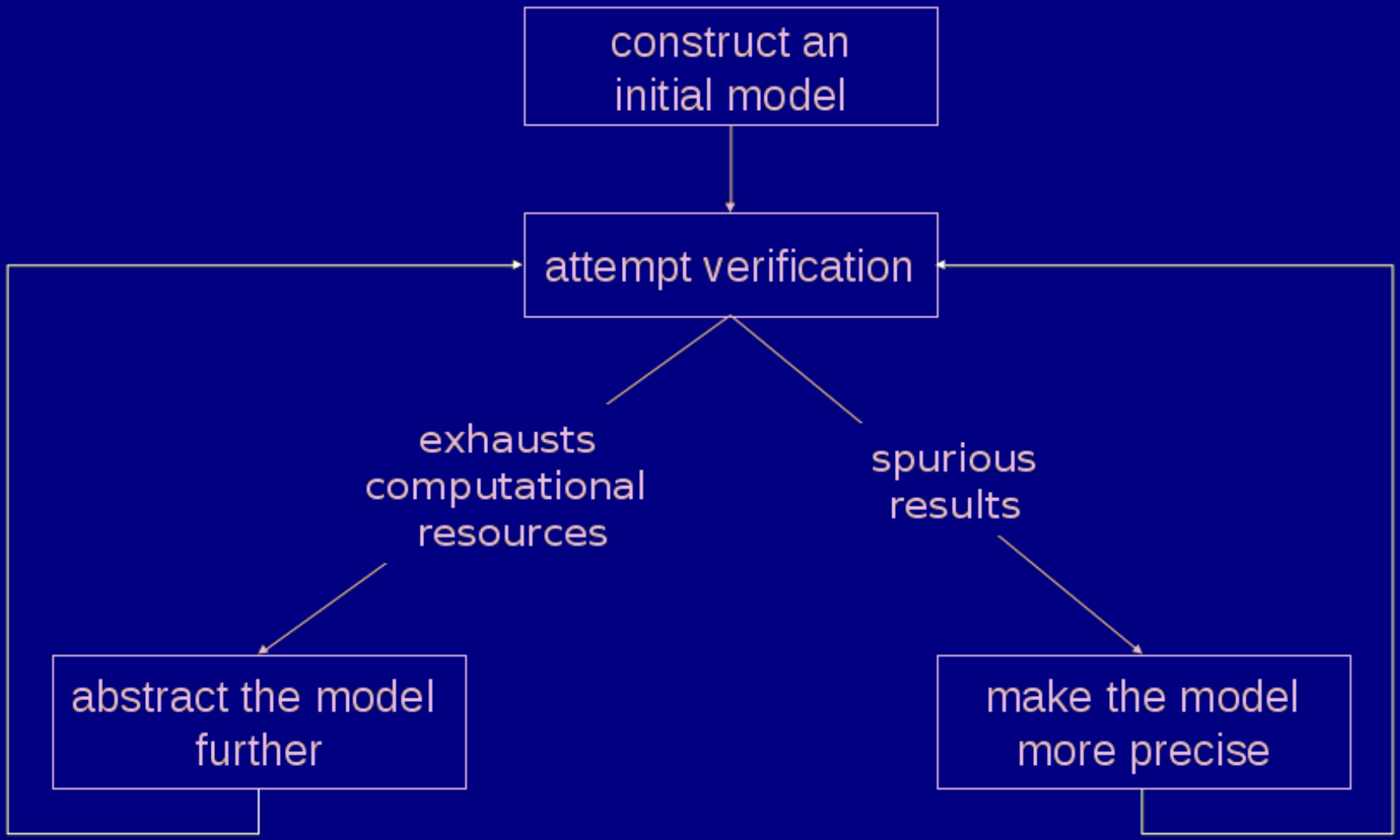
Intentional vs Explicit Representations

- Worst case:
- given a large set S of states
  - a representation capable of distinguishing each subset of S item cannot be more compact on average than the representation that simply lists elements of the chosen subset.
- Intentional representations work well when they exploit structure and regularity of the state space

Model Refinement

- Construction of finite state models
  - balancing precision and efficiency
- Often the first model is unsatisfactory
  - report potential failures that are obviously impossible
  - exhaust resources before producing any result
- Minor differences in the model can have large effects on tractability of the verification procedure
- finite state verification as iterative process

Iterative Process



Refinement: Adding details to the model

M1 |= P initial (coarse grain) model
(the counter example that violates P is possible in M1, but does not correspond to an execution of the real program)

M2 |= P refined (more detailed) model (the counter example is not possible in M2 but a new counter examples violates M2 but does not correspond to an execution of the real program)

....

Mk |= P (the counter example that violates P in Mk corresponds to an execution in the real program)

Example: Boolean Programs

- Initial Boolean program
  - omits all variables
  - branches if, while,.. refer to a dummy Boolean variable whose value is unknown
- Refined Boolean program
  - add ONLY Boolean variables, with assignments and tests
- Example: pump controller
  - a counter-example shows that the waterLevel variable cannot be ignored
  - a refined Boolean program adds a Boolean variable corresponding to a predicate in which waterLevel is tested (waterLevel < highLimit) rather than adding the variable waterLevel itself

Other Domains for Finite-State Verification

- Concurrent systems are the most common application domain
- But the same general principle (systematic analysis of models, where thorough testing is impractical) has other applications
- Example: Complex data models
  - Difficult to consider all the possible combinations of choices in a complex data model

Data Model Verification and Relational Algebra

- Many information systems are characterized by
  - simple logic and algorithms
  - complex data structures
- Key element of these systems is the data model (UML class and object diagrams + OCL assertions)
  = sets of data and relations among them
- The challenge is to prove that
  - individual constraint are consistent and
  - together they ensure the desired properties of the system as a whole

Example: A Simple Web Site

Singature = Sets + Relations

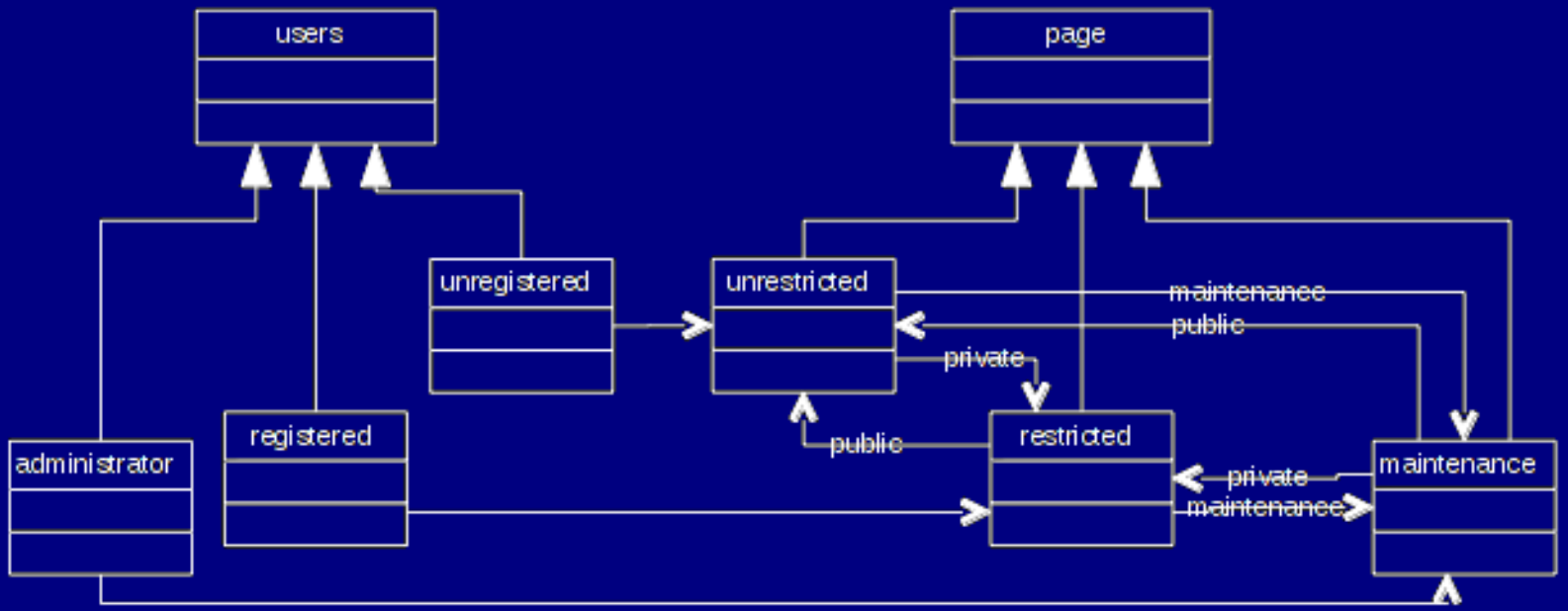
- A set of pages divided among restricted, unrestricted, maintenance pages
  - unrestricted pages: freely accessible
  - restricted pages: accessible only to registered users
  - maintenance pages: inaccessible to both sets of users
- A set of users: administrator, registered, and unregistered
- A set of links relations among pages
  - private links lead to restricted pages
  - public links lead to unrestricted pages
  - Maintenance links lead to maintenance pages
- A set of access rights relations between users and pages
  - unregistered users can access only unrestricted pages
  - registered users can access both restricted and unrestricted pages
  - administrator can access all pages including maintenance pages

Complete a Specification with Constraints

Example constraints for the web site:

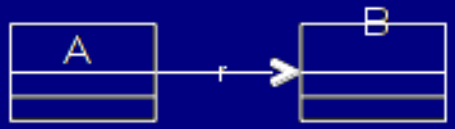
- Exclude self loops from links relations
- Allow at most one type of link between two pages
- NOTE: relations need not be symmetric:
- $< A, B > \rightarrow < B, A >$
- Web site must be connected
- ...

The Data Model for the Simple Web Site



LEGEND

Set B specializes set A



There is a relation r between sets A and B

Relational Algebra to Reason About Sets and Relations

- set union and set intersection obey many of the same algebraic laws as addition and subtraction of integers:
- commutative law
  $A \cup B = B \cup A$
  $A \cap B = B \cap A$
- associative law
  $(A \cup B) \cup C = A \cup (B \cup C)$
  $(A \cap B) \cap C = A \cap (B \cap C)$
- distributive law
  $A \cap (B \cup C) = (A \cap B) \cap (A \cap C)$

A Relational Algebra Specification (Alloy): Page

```
module WebSite

// Pages include three disjoint sets of links
sig Page {disj linksPriv, linksPub, linksMain: set Page }

// Each type of link points to a particular class of page
fact connPub {all p:Page, s: Site | p.linksPub in s.unres }
fact connPriv {all p:Page, s: Site | p.linksPriv in s.res }
fact connMain {all p:Page, s: Site | p.linksMain in s.main }

// Self loops are not allowed
fact noSelfLoop {no p:Page| p in p.linksPriv+p.linksPub+p.linksMain }
```

A Relational Algebra Specification: User

```
// Users are characterized by the set of pages that they can access
sig User {pages: set Page }
// Users are partitioned into three sets
part sig Administrator, Registered, Unregistered extends User {}
// Unregistered users can access only the home page, and unrestricted pages
fact accUnregistered {
all u: Unregistered, s: Site|u.pages = (s.home+s.unres)
}
// Registered users can access the home page,restricted and unrestricted pages
fact accRegistered {
all u: Registered, s: Site|u.pages = (s.home+s.res+s.unres)
}
// Administrators can access all pages
fact accAdministrator {
all u: Administrator, s: Site|
u.pages = (s.home+s.res+s.unres+s.main)
}
```

Analyze Relational Algebra Specifications

- Overconstrained specifications are not satisfiable by any implementation,
- Underconstrained specifications allow undesirable implementations
- Specifications identify infinite sets of solutions
  ... so ...
  Properties of a relational specification are undecidable
- A (counter) example that invalidates a property can be found within a finite set of small models
  ... so ...
- We can verify a specification over a finite set of solutions by limiting the cardinality of the sets

Checking a finite set of solutions

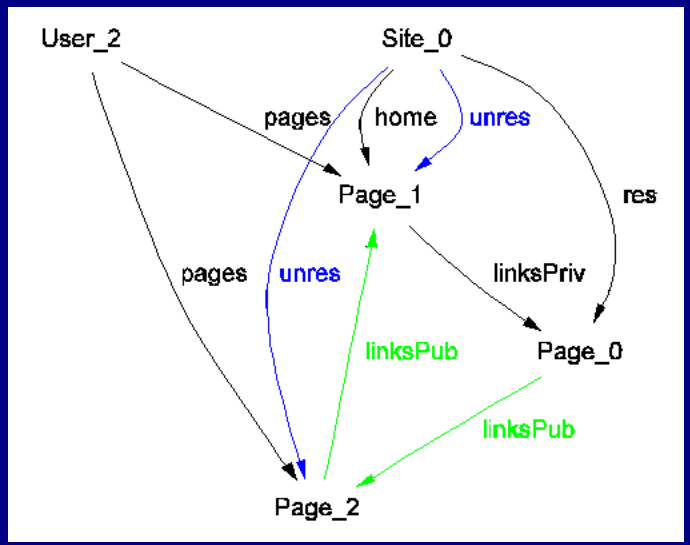
- If an example is found:
  - There are no logical contradictions in the model
  - The solution is not overconstrained
- If no counterexample of a property is found:
  - no reasonably small solution (property violation) exists
  - BUT NOT that NO solution exists
    - We depend on a “small scope hypothesis”: Most bugs that can cause failure with large collections of objects can also cause failure with very small collections (so it’s worth looking for bugs in small collections even if we can’t afford to look in big ones)

Analysis of Web Site Specification

```
run init for 5 //Cardnilaity limit: consider up to 5 object of each type

// can unregistered users
//      visit all unrestricted pages?
assert browsePub {//Property to be checked
all p: Page, s: Site|
       p in s.unres implies s.home in p.* linksPub
                                    //* is transitive closure (including home)
}
check browsePub for 3
```

Analysis Result



Counterexample:

- Unregistered User_2 cannot visit the unrestricted page page_2
- The only path from the home page to page_2 goes through the restricted page page_0
- The property is violated because unrestricted browsing paths can be interrupted by restricted pages or pages under maintenance

Correcting the Specification

- We can eliminate the problem by eliminating public links from maintenance or reserved pages:

```
fact descendant {
   all p:Pages, s:Site|p in s.main+s.res
      implies no p. links.linkPub
}
```

- Analysis would find no counterexample of cardinality 3
- We cannot conclude that no larger counter-example exists, but we may be satisfied that there is no reason to expect this property to be violated only in larger models

Summary

- Finite state verification is complementary to testing
  - Can find bugs that are extremely hard to test for
    - example: race conditions that happen very rarely, under conditions that are hard to control
  - But is limited in scope
    - cannot be used to find all kinds of errors
- Checking models can be (and is) automated
- But designing good models is challenging
  - Requires careful consideration of abstraction, granularity, and the properties to be checked. Often requires a cycle of model / check / refine until a useful result is obtained.

Book Assignment

Choose and complete any two Chapter 8 Exercises (pg 147)
Due in the dropbox by February 6, 2014 2359

Reading Assignment

Chapter 10