ALLOY

Arjav Dave
Jitendra Gupta
Nishit Shah

Agenda

- Overview
- Alloy Architecture
- Alloy Specification Language
- Alloy Analyzer Demo
- Comparisons
- Conclusion

History

- The Alloy language was designed by Daniel Jackson, with help from Ilya Shlyakhter and Manu Sridharan at MIT.
- Analyzer developed by Software Design Group
- Alloy 1.0 [1999]
- Alloy 2.0 [2001]
- Alloy 3.0 [2004]
- Alloy 4.0 [2006]

Overview

- Alloy is a Light-weight, structural modeling language which is based on First Order Logic
- Z has a major influence on Alloy
- Alloy has its own automatic analysis tool
- Alloy is similar to OCL but it has a more conventional syntax and a simpler semantics

Key Concepts

1.Alloy consists of atoms and relations:

- An atom is a primary entity with indivisible, immutable and un-interpreted as its properties.
- Relations are structures that relate atoms where each relation is a set of ordered tuples.

2. Non-specialized Logic:

- Alloy does not need special logic for state machines, traces, synchronization and concurrency.
- Here everything can be represented by relations and atoms.

Key Concepts

3. Alloy has counter examples and finite scope

- provides counter examples for invalid models
- checks models for finite scope only

4. Alloy analyzes using SAT:

- SAT – Boolean Satisfiability Problem
- Analyzer based on KODKOD *(Alloy 4.0)*
- KODKOD – an efficient SAT-based constraint solver
- Allows analyzing of partial models

Difference Between Alloy Analyzer and Model Checkers

- The Alloy Analyzer is designed for analyzing state machines with operations over complex states
- Model checkers are designed for analyzing state machines that are composed of several state machines running in parallel, each with relatively simple state

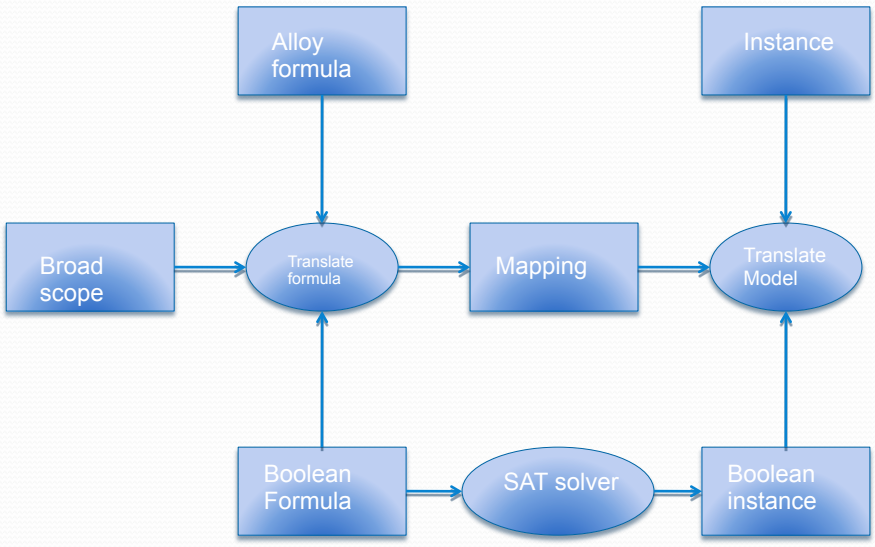
Difference Between Alloy Analyzer and Theorem Provers

- The Alloy Analyzer's analysis is fully automatic, and when an assertion is found to be false, the Alloy Analyzer generates a counterexample. It's a "refuter" rather than a "prover".
- When a theorem prover fails to prove a theorem, it can be hard to tell what's gone wrong: whether the theorem is invalid, or whether the proof strategy failed.

Alloy Architecture

- Conversion to negation normal form and removing all the existential quantifiers from the formula.
- The formula is translated for a chosen scope to a boolean formula.
- Boolean formula is converted to a conjunctive normal form
- Boolean formula is presented to the SAT solver.
- Incase the solver finds a model, a model of the relational formula is then reconstructed from it.

Alloy Architecture



Module Elements

1. Signature Declarations
   - Declaration of sets and relation
2. Constraint Paragraphs
   - Consists of facts, predicates and functions to form constraints
3. Assertions
   - Properties expected to hold
4. Commands
   - Run and check which instruct the analyzer to perform verification

Signature

- Signatures represent sets of objects in the system
- Relations are defined in the signature body
- Relation=Field of Class
- Example:

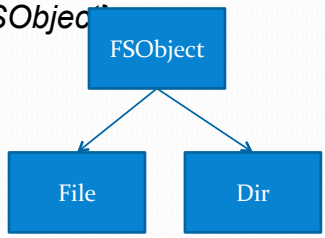
```
sig FSObject{
    parent: lone Dir
    }
```

Signature

sig FSObject{}

sig dir extends FSObject{contents : set FSObject}

sig file extends FSObject{}



Extends means

- File is a subset of all FSObjects
- Files are disjoint from Dir

Fact

- Avoid examples that violate fact
- d must be the parent of o
- Example:

```
fact{
    all d: Dir, o: d.contents | o.parent = d
}
```

Assertion

- Claims something must be true due to the behavior of the system
- Example:

```
assert acyclic {
    no d:  Dir | d in d.^contents
}
check acyclic for 5
```

- No directory is a content of itself

Assertion

- Given an assertion we can check it as follows:
  - Negate the assertion and conjunct it with the rest of the specification
  - Look for a model for the resulting formula, if there exists such a model (i.e., the negation of the formula is consistent) then we call such a model a counterexample

Fact and Assert

Fact

- Force a constraint
- If false then no model is generated
- All facts can be validated together

Assert

- Check for a condition
- If false counterexample are generated
- Each assert statement is done at a time

Constraint Problems

- Over-Constrained: Eliminate possibilities intended to allow
- Under-Constrained: Specification allows some unintended behaviors
- Run Command finds both over-constraint and under-constraint errors

Predicates

- Constraints to be used in different contexts are packaged as predicates
- A predicate is similar to function except that its output is true or false and not an expression or term

```
Pred move [fs,fs': FileSystem, x:FSObject, d:Dir]{
        (x+d) in fs.objects
         fs'.parent = fs.parent – x->(x.(fs.parent)) + x->d
}
```

Run and Check

Run:

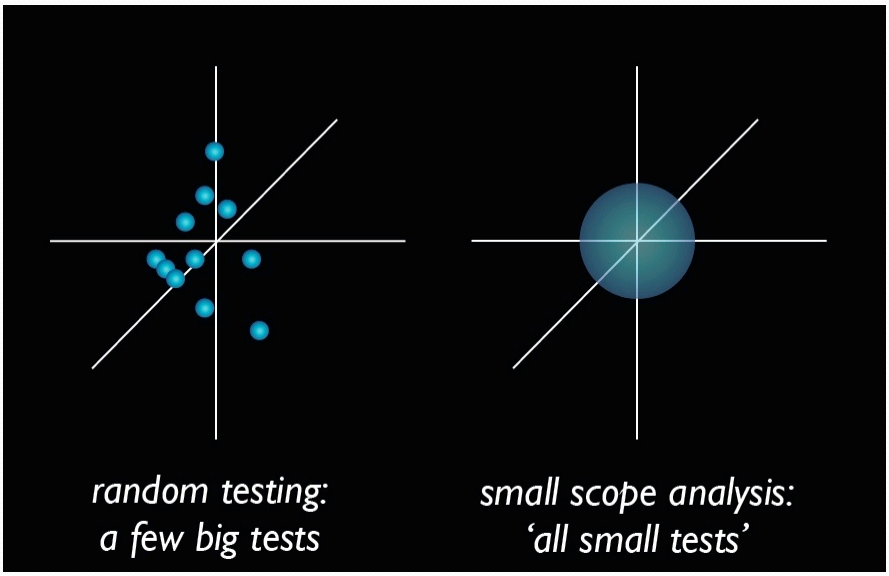
- Run is executed on the predicate
- Run shows the instance for which predicate is true

Check:

- Check is written for assertion statement
- Outcome is counterexample if check is false.

Scope

- Maximum number of objects for which model is evaluated
- Maximum scope 2^32
- Keyword “exactly” can be used in scope specification
- Example
  - Check A for 5
  - Check A for exactly 3 Directory, exactly 5 File



Relational Operators

- Constants

none	*empty set*
univ	*universal set*
iden	*identity relation*

- Set Operators

+	*union*
&	*intersection*
-	*difference*
in	*subset*
=	*equality*
→	*product operator*

Relational Operators

- Join operators

.	*dot join*
[ ]	*box join*

- Unary operators

~	*transpose*
^	*transitive closure*
*	*reflexive transitive closure*

Relational Operators

- Boolean operators

!	**not**	*negation*
&&	**and**	*conjunction*
||	**or**	*disjunction*
=>	**implies**	*Implication*
,	**else**	*alternative*
⇔	**iff**	*bi-implication*

Relational Operators

- quantifiers

all	*F holds for **every** x in e*
some	*F holds for **at least one** x in e*
no	*F holds for **no** x in e*
lone	*F holds for **at most one** x in e*
one	*F holds for **exactly one** x in e*

- set declarations

set	*any number*
one	*exactly one*
lone	*zero or one*
some	*one or more*

File System Example

```
sig FSObject { parent: lone Dir }                    // A file system object
sig Dir extends FSObject { contents: set FSObject }  // A directory
sig File extends FSObject { }                        // A file in the file system
fact { all d: Dir, o: d.contents | o.parent = d }    //directory is parent of its contents
fact { File + Dir = FSObject }                       // All FSO are files or directories
one sig Root extends Dir { } { no parent }           // There exists one root directory
fact { FSObject in Root.*contents }                  // File system is connected
assert oneRoot { one d: Dir | no d.parent }          // File system has one root
check oneRoot for 5                                  // Now check it for a scope of 5
```

ALLOY ANALYSER DEMO

Type of Analysis

- Two types of analysis:
  - ✓ *Simulation,* in which consistency of an invariant or an operation is demonstrated by generating an environment that models it
  - ✓ *Checking,* in which a consequence of the specification is tested by attempting to generate a counter-example
- Simulation is for determining consistency (i.e., satisfiability) and Checking is for determining validity

Type of Analysis (2)

- If a formula has at least one model then the formula is ***consistent*** (i.e., ***satisfiable***)
- If every (well-formed) environment is a model of the formula, then the formula is ***valid***
- Alloy analyzer restricts the simulation and checking operations to a finite scope
- Validity and consistency problem within a finite scope are decidable problems

Comparison with UML

- Has similarities (graphical notation, OCL constraints) but it is neither lightweight, nor precise
- UML includes many modeling notions omitted from Alloy

Comparison with OCL

ALLOY	OCL
✓ Simple but less expressive semantics	✓ More expressive but challenging semantics
✓ No support for data types such as Strings, Real no., etc	✓ Supports different data types unlike Alloy
✓ Support for Partial Models	✓ Does not support partial models
✓ Automatic tool support	✓ Lacks Automatic tool support

Comparison with Z

ALLOY	Z
• Written in ASCII format and notations are easily available	• Z notations are not available easily
• Simple and restrictive semantics	• Complex and expressive semantics
• Tool support is readily available for free	• Different tools in the market but not free
• Finite Scope checking	• Scope need not be known

Applications

- Scheduling
- Document structuring
- Cryptography
- File System synchronization
- Network configuration protocols

Application In Industry

- Animating requirements (Venkatesh, Tata)
- Military simulation (Hashii, Northtrop Grumman)
- Generating network configurations (Narain, Telcordia)

Application In Research

- Exploring design of switching systems (Zave, AT&T)
- Checking refinements (Bolton, Oxford)
- Security features (Pincus, MSR)

Good Things

- conceptual simplicity and minimalism
  - very little to learn
  - WYSIWYG: no special semantics (e.g., for state machines)
- High Level Notation
  - constraints -- can build up incrementally
  - relations flexible and powerful
- Automatic Analysis
  - counterexamples are never spurious
  - visualization a big help

Limitations (Bad Things)

- Poor scaling
- No notion of time and hence unsuitable for reachability analysis or chain of operations
- Since the analyzer depends on user specifications and scope, false positives are possible
- The only primitive type supported is integers
- Recursive functions hard to express
- Module system doesn't offer real encapsulation

Conclusion

- Easy to understand paragraph structure
- Graphical representation of the Analyzer is really helpful in easily understanding instances
- We found that Tool support for static variables is easily available but the Dynamic variables are left neglected
- Alloy has limited expressiveness and is designed for semantic analysis

References

- http://alloy.mit.edu/
- http://alloy.mit.edu/alloy4/quickguide/gui.html
- http://alloy.mit.edu/kodkod/
- http://en.wikipedia.org/wiki/Skolemization
- http://ww1.ucmss.com/books/LFS/CSREA2006/SER4949.pdf
- http://alloy.mit.edu/community/talks

Thank You

QUESTIONS???