Alloy

Wenxi Wang

University of Virginia

wenxiw@virginia.edu

Alloy

Overview

Alloy Language:

build models, requirements, specifications, software design

1) Lightweight: small and easy to use, and capable of expressing common properties tersely and naturally

2) Precise: having a simple and uniform mathematical semantics

Alloy Analyzer:

fully automated software model analysis

Alloy

Why we need Alloy?

Alloy Language:

- Provides precise description of artifacts
- Good Documentation
- Provides higher level of abstraction
- Helps describe properties that we cannot (easily) express in source code

Declarative Language!

Alloy Analyzer:

- Enables machine reasoning
- Helps eliminate/reduce ambiguities, inconsistencies, and incompleteness

Alloy

Why we need Alloy?

"Everybody likes a winner"

- Ambiguous?
- Incomplete?

Precise meaning?

- all p: Person | some w: Winner | p.likes(w)
- all p: Person | all w: Winner | p.likes(w)
- some w: Winner | all p: Person | p.likes(w)

Alloy

A lot of Applications over the past two decades

1. Network and Web Security Modeling and Analysis
2. Formal modeling and analysis of a flash filesystem in Alloy
3. Efficient re-resolution of specifications for evolving software architectures
4. Specification of a distributed spanning tree
5. Declarative testing for distributed programs
6. Analyzing the Fundamental Liveness Property of the Chord Protocol

… …

Alloy

Alloy in General

Alloy is general enough that it can model

- any (finite) domain of individuals and
- any relations between them

Alloy

Overview

Alloy Language:

build models, requirements, specifications, design

1) Lightweight: small and easy to use, and capable of expressing common properties naturally

2) Precise: having a simple and uniform mathematical semantics

Alloy Analyzer:

fully automated software model analysis

Alloy Language

Atoms and Relations

An atom is a primitive entity that is
– indivisible: it cannot be broken down into smaller parts
– immutable: it does not change over time
– uninterpreted: it does not have any built-in property (the way numbers do for example)

A relation is a structure that relates atoms
– It is a set of tuples of the same type

Alloy Language

Atoms and Relations: Example

FriendBook	WorkBook
Ted -> ted@gmail.com Ryan -> ryan@hotmail.com	Pilard -> pilard@uiowa.edu Ryan -> ryan@uiowa.edu

- Unary relations: a set of names, a set of addresses and a set of books

  Name = { (N0), (N1), (N2) }

  Addr = { (D0), (D1) }

  Book = { (B0), (B1) }

- A binary relation from names to addresses

  address = { (N0,D0), (N1,D1) }

- A ternary relation from books to name to addresses

  addr = { (B0,N0,D0), (B0,N1,D1), (B1,N1,D2) }

Atoms

Tuples

Everything in Alloy is built from relations

Alloy Language

Main components of Alloy Model

1. Signatures and Fields
2. Predicates
3. Facts
4. Commands and scopes

Alloy Language

Main components of Alloy Model

1. Signatures and Fields
2. Predicates
3. Facts
4. Commands and scopes

Alloy Language

Signatures and Fields

1. Signatures: introduces a set of atoms — Unary relation
2. Fields: declares relations

Example:

1. Introduces three sets named A, B, C, respectively

   sig A {}   sig B {}   sig C {}

2. Declare Binary Relation:

   sig A { **f1**: B }   // f1 is a field, a binary relation of type A x B

3. Ternary Relation:

   sig A { **f2**: B -> C } // f2 is a field, a ternary relation of type A x B x C

Alloy Language

Cardinality Constraints: constrain the sizes of sets

- **some** e //e is non-empty
- **no** e //e is empty
- **lone** e //e has at most one tuple
- **one** e //e has exactly one tuple

Example:

```
one sig List {                  // Declare one single linked-list
   header: lone Node            // with at most one header n
}
sig Node {
  next: lone Node               // each node has at most one next node
}
```

Alloy Language

Cardinality Constraints: constrain the sizes of sets

- **some** e //e is non-empty
- **no** e //e is empty
- **lone** e //e has at most one tuple
- **one** e //e has exactly one tuple

Example:

```
one sig List {                  // Declare one single linked-list
  header: lone Node             // with at most one header n
}
sig Node {
  next: lone Node               // each node has at most one next node
}
```

Alloy Language

Main components of Alloy Model

1. Signatures and Fields
2. Predicates
3. Facts
4. Commands and scopes

Alloy Language

Facts and Predicates

Alloy models can be refined further by adding formulas expressing additional constraints over signatures and relations

- **Facts**: the constraints that Alloy model must satisfy
- **Predicates**: optional constraints that Alloy model can satisfy

Example:

```
one sig List {
  header: lone Node
}
sig Node {
  next: lone Node
}
```

```
// All nodes are reachable from the header node
fact Reachable {

}
```

Alloy Language

Facts and Predicates

Relational Operators

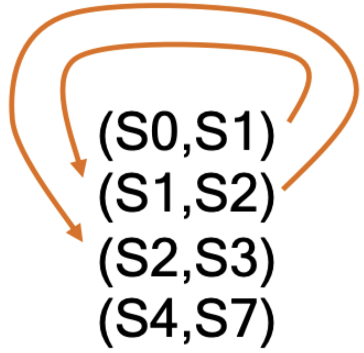
`_ -> _`	arrow (cross product)
`~ _`	transpose
`_ . _`	dot join
`_[_]`	box join
`^ _`	transitive closure
`* _`	reflexive-transitive closure
`_ <: _`	domain restriction
`_ :> _`	image restriction
`_ ++ _`	override

Alloy Language

Relational Operators: transitive closure

Given a binary relation r, the **transitive closure** of r, denoted **^r**, includes all elements x and y such that x can reach y by following **one or more steps** of r.

`r : S -> S`



(S0,S1)
(S1,S2)
(S2,S3)
(S4,S7)
(S0,S2)
(S0,S3)
(S1,S3)

`^r`

Alloy Language

Relational Operators: reflexive-transitive closure

reflexive-transitive closure of a relation ***r** is the transitive closure that also includes **reflexive connections**.

`r`	(S0,S1) (S1,S2) (S2,S3) (S4,S5)
`S`	= { S0, ..., S5 }

`*r`	(S0,S1) (S1,S2) (S2,S3) (S4,S5) (S0,S2) (S0,S3) (S1,S3)	`^r`
	(S0,S0) (S1,S1) (S2,S2) (S3,S3) (S4,S4) (S5,S5)	`iden :> S`

Alloy Language

Relational Operators

Example:

```
one sig List {
  header: lone Node
}
sig Node {
  next: lone Node
}
```

```
// All nodes are reachable from the header node
fact Reachable {
  Node = List.header.*next
}
```

Alloy Language

Relational Operators: dot join

- What is the join of theses two tuples?

  $(a_1, ..., a_m)$ and $(b_1, ..., b_n)$

  – If $a_m \neq b_1$ then the join is undefined
  – If $a_m = b_1$ then it is: $(a_1, ..., a_{m-1}, b_2, ..., b_n)$

  Example

  – (a,b).(a,c,d) undefined
  – (a,b).(b,c,d) = (a,c,d)

- What about (a).(a)? Not defined !

  $t_1.t_2$ is not defined if t_1 and t_2 are **both** unary tuples

Alloy Language

Facts and Predicates

Logical Operators

<table>
<tr><td>not _</td><td>! _</td><td>(Boolean) negation</td></tr>
<tr><td>_ and _</td><td>_ && _</td><td>conjunction</td></tr>
<tr><td>_ or _</td><td>_ || _</td><td>disjunction</td></tr>
<tr><td>_ implies _</td><td>_ => _</td><td>implication</td></tr>
<tr><td>else _</td><td></td><td>alternative</td></tr>
<tr><td>_ iff _</td><td>_ <=> _</td><td>equivalence</td></tr>
</table>

Set Operators

_ + _	union
_ & _	intersection
_ - _	difference
_ in _	subset
_ = _	equality
_ != _	disequality

Alloy Language

Main components of Alloy Model

1. Signatures and Fields
2. Predicates
3. Facts
4. Commands and scopes

Alloy Language

Run Commands and Scopes

To analyze a model, you add a run command and instruct Alloy Analyzer to execute a **predicate**

– the **run command**

tells the tool to search for an instance that satisfy all facts and the predicate

– you may also give a **scope** to signatures

bounds the size of instances that will be considered

Example:

```
one sig List {
  header: lone Node
}
sig Node {
  next: lone Node
}
```

```
Predicate Reachable {
  Node = List.header.*next
}
```

```
run RepOk for 3
```

Alloy

Overview

Alloy Language:

build models, requirements, specifications, design

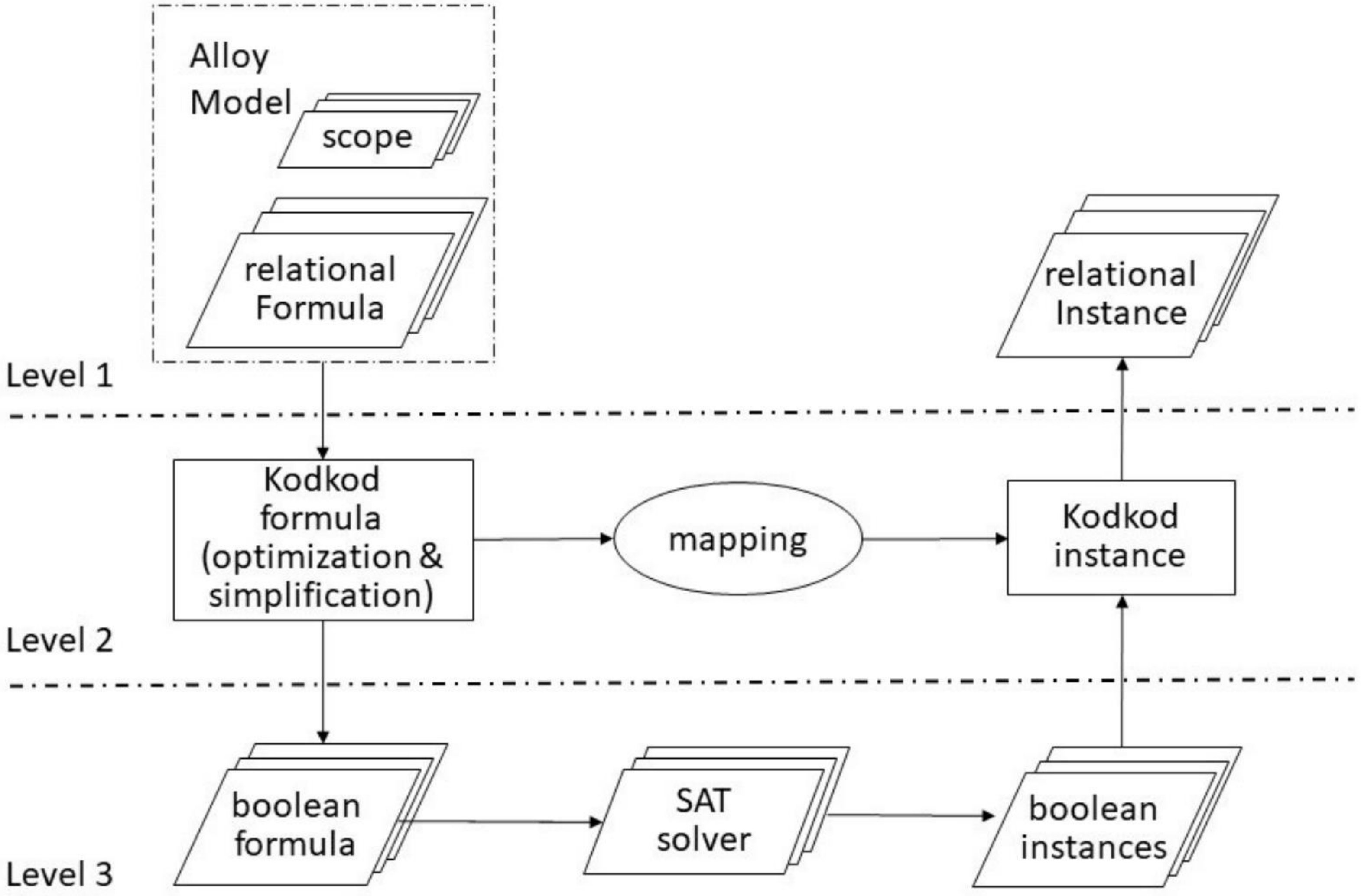
1) Lightweight: small and easy to use, and capable of expressing common properties tersely and naturally

2) Precise: having a simple and uniform mathematical semantics

Alloy Analyzer:

fully automated software model analysis

Alloy Analyzer



Alloy Analyzer

Let’s implement an Alloy model!

Alloy

Research Topics

Improve analysis using ML

- Compiler optimizations
- Incremental analysis
- SAT optimizations
- Symmetry breaking

Solve new ML applications using Alloy

- Can you model ML-related problems from your domain?
- ML applications:
  - verify a certain property of Neural network models
  - Synthesis NN model with Alloy
  - Generate test cases for testing NN models