Software Abstractions

By
Isaac Yoshino
Sahar Jambi

CSCI 5828 – Software Engineering
Spring 2010

Introduction

- Software development is difficult.
- Choosing correct abstractions to base your design around is a tricky process.
- Simple and robust designs can turn out to be incoherent and even inconsistent when you come to implement them.

Abstraction

- Software is built on abstractions.
- Abstraction is a way of hiding details in order to make it easier to see the big picture model.
- Good abstraction will generalize that which can be made abstract while allowing specificity where abstraction doesn't work.
- Abstraction programming is the process of identifying common patterns that have systematic variations.

Formal Specification

- An approach to attack the design of abstraction.
- Describes what the system should do.
- Uses of notations for ease of expression and exploration.
- These notations are precise, expressive, and unambiguous.
- Given a specification, it is possible to use formal verification techniques to demonstrate that a candidate system design is correct.

Alloy

- A language and software modeling approach.
- Based on *formal specification* with features for fully automated analysis.
- Designed by the Software Design Group at MIT lead by Professor Daniel Jackson in 1997.

Why Alloy?

- The ability for incremental analysis.
- High performance and scalability.
- Applied to many fields including scheduling, cryptography and instant messaging.

Alloy: Main Features

- Alloy is a lightweight modelling language for software design.
- It is amenable to a fully automatic analysis.
- Using the Alloy Analyzer, it provides a visualizer for making sense of solutions and counterexamples it finds.
- The key elements of the approach are: logic, language, and analysis.

Alloy: Key Elements

- Alloy = logic + language + analysis
- logic

first order logic + relational calculus

- language

syntax for structuring specifications in the logic

- analysis

bounded exhaustive search for counterexample to a claimed property using SAT

Logic

- The logic provides the building blocks of the language.
  - Structures are represented as relations.
  - Structural properties are expressed with a few simple and powerful operators.
  - States and conditions are described using formulas called constraints

Logic: relations of atoms

- Atoms are Alloy's primitive entities
  - indivisible, immutable, uninterpreted
- Relations associate atoms with one another
  - set of tuples, tuples are sequences of atoms
- Every value in Alloy logic is a relation!
  - relations, sets, scalars all the same thing

Language

- The language adds a small amount of syntax to the logic for structuring descriptions.
- Supports classification and incremental refinement with a flexible type system.

Analysis

- The analysis brings software abstractions to life.
- A form of constraint solving.
  - Simulation involves finding instances of states that satisfy a given property.
  - Checking involves finding a counterexample-an instance that violates a given property
  - The search for instances is specified by a scope.

Example: Hotel Room Locking

- Most hotels now issue disposable room keys.
- All rooms have recodable locks.
- A new key is issued to a new occupant which recodes the lock so the previous keys will no longer work.

Example: Hotel Room Locking

- To represent the key generators:
  - Declare signatures for the keys and time instants:

sig Key {}

sig Time {}

  - A signature defines the vocabulary for the model.
  - When you declare a signature, you are defining an atom.

Example: Hotel Room Locking

- To represent the key generators:
  - Each room has a set of keys, a current key at a given time:

```
sig Room {
  keys: set Key,
  currentKey: keys one -> Time
  }
```

Example: Hotel Room Locking

- To represent the key generators:
  - No key belongs to more than one room lock:

```
fact DisjointKeySets{
    Room <: keys : Room lone -> Key
    }
```

Example: Hotel Room Locking

- To represent the key generators:
  - The front desk is a singleton signature, that groups two relations:
    - *Lastkey* maps a room to the last key, and
    - *Occupant* maps a room to the guest.

```
one sig FrontDesk {
    lastKey: (Room -> lone Key) -> Time,
    occupant: (Room -> Guest) -> Time
    }
```

Example: Hotel Room Locking

- To represent the key generators:
  - A guest holds a set of keys at a given time:

```
sig Guest {
  keys: Key -> Time
  }
```

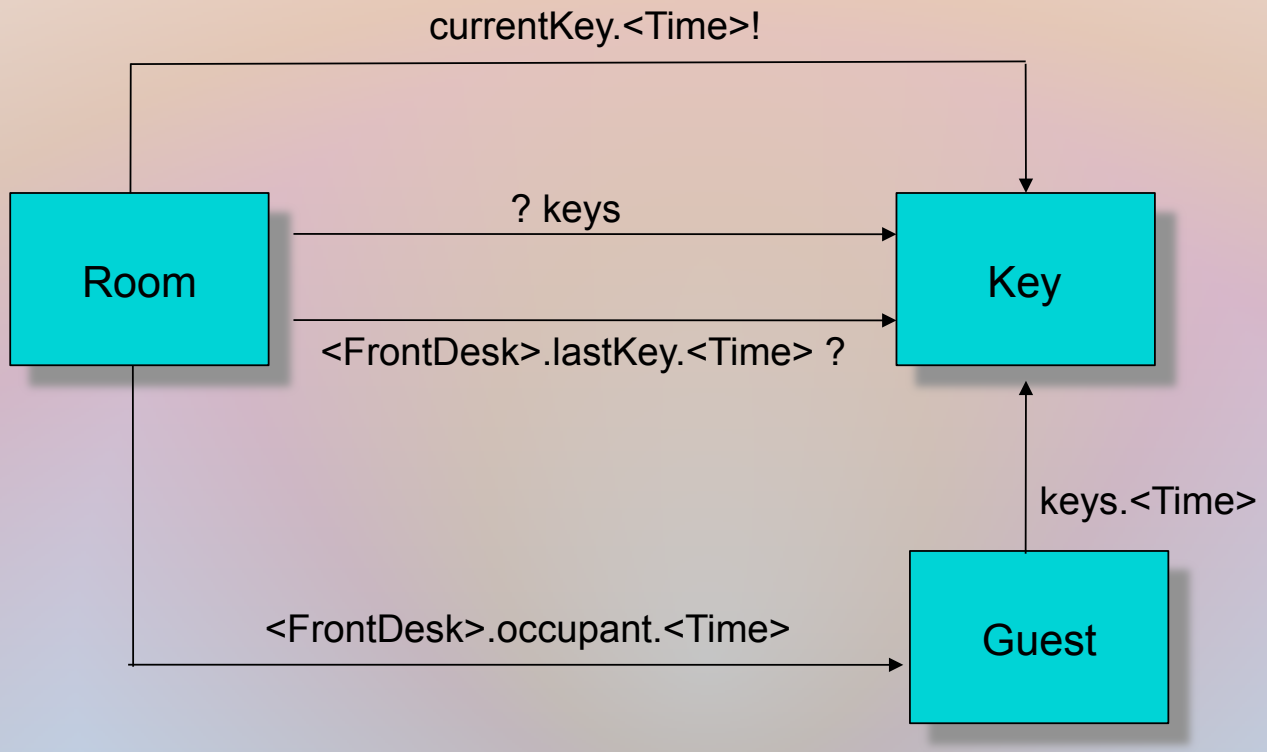
Example: Hotel Room Locking

- To generate the successor key:

```
fun nextKey (k: Key, ks: set Key): set Key {
  min [k.nexts & ks]
  }
```

Example: Hotel Room Locking

- The model diagram of the declaration:



Example: Hotel Room Locking

- Hotel Operations:
  - In the initial state, no guests hold keys

```
pred init [t: Time] {
  no Guest.keys.t
  no FrontDesk.occupant.t
  all r: Room | FrontDesk.lastKey.t [r] =
    r.currentKey.t
}
```

Example: Hotel Room Locking

- Hotel Operations:
  - Checking out requires that the room be occupied by the guest.

```
pred checkout [t, t': Time, g: Guest] {
  let occ = FrontDesk.occupant {
    some occ.t.g
    occ.t' = occ.t - Room ->g }
  FrontDesk.lastKey.t = FrontDesk.lastKey.t'
  noRoomChangeExcept [t, t', none]
  noGuestChangeExcept [t, t', none] }
```

Example: Hotel Room Locking

- Hotel Operations:
  - Checking in requires that the room have no current occupant.

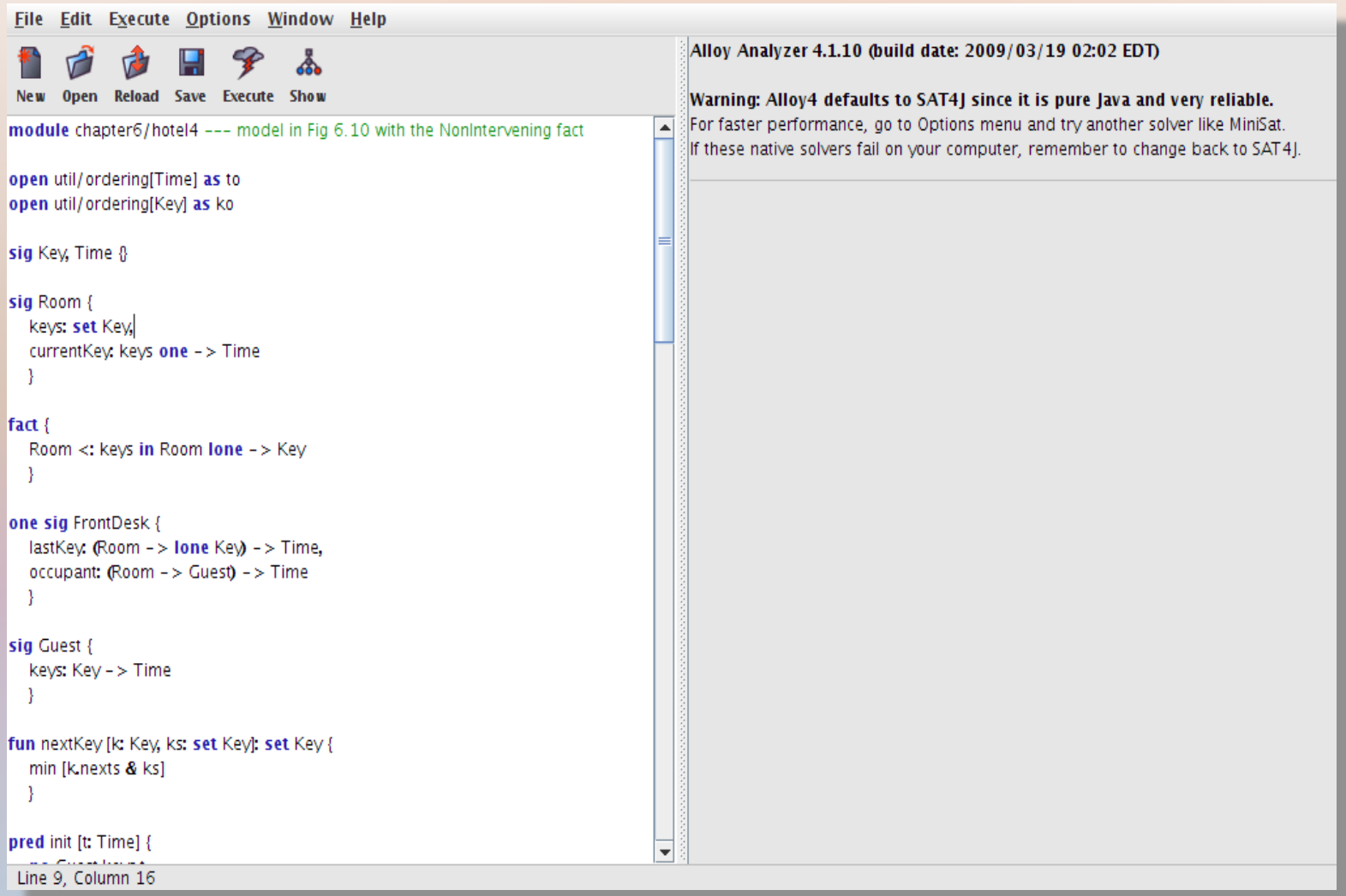
```
pred checkin [t, t': Time, g: Guest, r: Room, k: Key] {
    g.keys.t' = g.keys.t + k
    let occ = FrontDesk.occupant {
        no occ.t [r]
        occ.t' = occ.t + r -> g}
    let lk = FrontDesk.lastKey {
        lk.t' = lk.t ++ r -> k
        k = nextKey [lk.t [r], r.keys}
        noRoomChangeExcept [t, t', none]
        noGuestChangeExcept [t, t', g]}
```

Example: Hotel Room Locking

- Analysis:
  - Check that no unauthorized entries can occur.

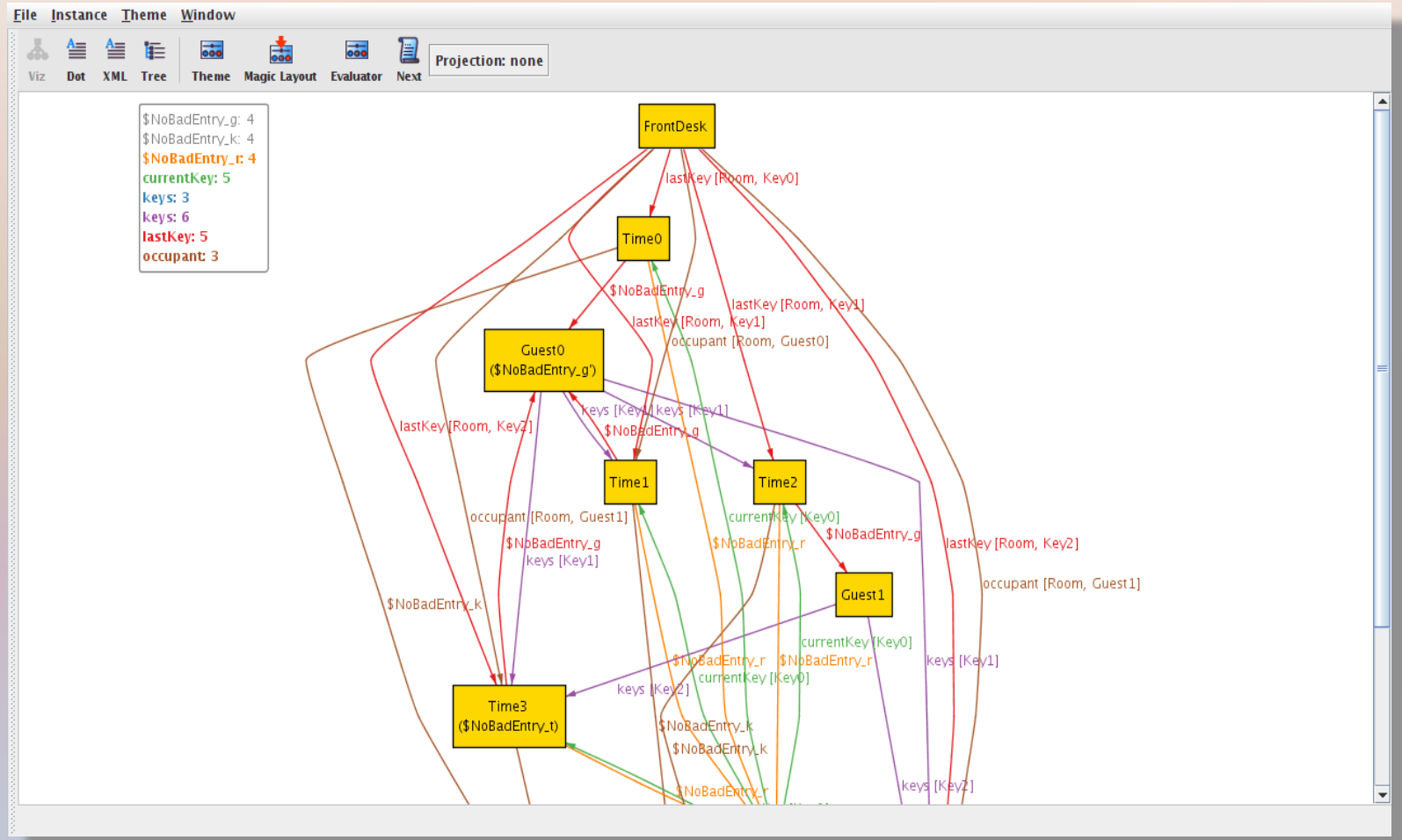
```
assert NoBadEntry {
  all t: Time, r: Room, g: Guest, k: Key |
    let t' = t.next, o = FrontDesk.occupant.t[r] |
      entry [t, t', g, r, k] and some o => g in o
}
```

Alloy Editor



Alloy Analyzer

Alloy Analyzer



Conclusion

- The Alloy language provides an automated and visual interface for formal specification.
- Alloy is a relation based modelling notation.
- It’s syntax and semantics is easy.
- The automatic analysis is a great aid for developers.

Reference

- Jackson, Daniel (2006). Software Abstractions: Logic, Language, and Analysis. MIT Press. ISBN 978-0-262-10114-1.