purposes

after this class you’ll understand
why data models help with subtle design decisions
how to model some tricky cases
how to derive a schema from a model

key ideas
thinking about state abstractly with sets and relations,
where structure is expressed in relations alone
data models for design

a role for data models
express subtle properties of parts of a design
can clarify the design options

an example
Gmail’s labels and conversations
organizing messages with labels

Google

label:hacking

Gmail

COMPOSE

Alyssa P. Hacker

Inbox

0 GB (0%) of 15 GB used
Manage

Terms - Privacy

Last account activity: 14 hours ago
Details

hacking
meetups
todo
More
surprising behavior #1

No messages matched your search. Try using search options such as sender, date, size and more.
what's going on

turn off conversation view
then see individual messages
surprising behavior #2
data model

Impact of immutability?

Conversation

messages

Message

clabels

mlabels

Label
exercise:
subtle design
write a textual constraint for this model that explains Gmail’s behavior

use this Alloy syntax

all m: Message | ...
c.messages: set of msgs in conversation c
C.messages: set of msgs in all conversations in set C
the constraint

all c: Conversation | c.messages.mlabels = c.clabels

consequences

what happens if you delete a label from a conversation then add it back again?
modeling tricks
the magic of relations

note that atoms are
distinguishable: have an identity

but they’re also
immutable: don’t change
indivisible: not structured

so how to express structure & mutability?
relations!
kinds of relationship

- "containment"
  - Message
  - Body
  - File
  - attachmens

- "naming"
  - User
  - User Name

- "association"
  - User
  - groups
  - Group

- "properties"
  - File
  - perms
  - Permission
  - grantedTo
    - User
    - ReadPerm
    - WritePerm
subsets instead of booleans

why?
flag is low level way to represent
obscures dynamic classification
prevents recording multiplicity graphically
3-way relations

Why?

student-class-grade is a 3-way relation
need a “tuple” type such as Enrollment
exercise:
tricky model
modeling a crime database

design a data model for an app that tracks criminals and their recent locations which criminals know each other when they met and were last seen together
This says that if two pairs have the same criminals, then they’re the same pair.

\[ \forall p, p' : \text{Pair} \mid p.(\text{fst} + \text{snd}) = p'.(\text{fst} + \text{snd}) \Rightarrow p = p' \]

\[ \forall p : \text{Pair} \mid p.\text{since} \leq p.\text{last} \]
deriving schemas
start with abstract model

example
our friend the URL shortener
add attributes

for each set box in turn
ask: if just a primitive value, will app work well?
if not, enrich by adding attributes
pick primitive types

for each attribute & unenriched set
pick a primitive type to represent it

- User
  - name
  - UserName (String)

- Shortcut (String)
  - owns
  - expires
  - Date (Date)

- URL
  - expandsTo
  - dom
  - path
  - Domain (String)
  - Path (String)
reverse relations

check each relation
if points from one to many, reverse it
group into tables

break graph into little trees
just one relation deep

User
  └── UserName (String)

Shortcut (String)
  ├── ! owner
  │    └── User
  ├── expires
  │    └── Date (Date)
  └── expandsTo
      └── URL
          ├── dom
          │    └── Domain (String)
          └── path
              └── Path (String)
write out tables

for each table
root becomes the key of the table
relations to other tables become references
sets without primitive types become ids

CREATE TABLE users (  
id int primary key,  
name varchar(20)  )

CREATE TABLE shortcuts (  
shortcut varchar(10) primary key,  
owner int references users(id),  
expandsTo int references urls(id),  
expires date  )

CREATE TABLE urls (  
id int primary key,  
domain varchar(50),  
path varchar(100)  )
tricky case: many to many

what to do about users relation?
turn relation into a table of its own

CREATE TABLE accountUsers (  
    account int references account(id),  
    user int references user(id)  
)
tricky case: subsets

options
one table for the superclass
one for each subclass
one table for each subclass + one for superclass
tricky case: subsets (ctd)

one table for the superclass
CREATE TABLE shortcuts (shortcut, owner, expires, expandsTo)

one table for each subclass
CREATE TABLE permShortcuts (shortcut, owner)
CREATE TABLE expShortcuts (shortcut, expires)

one table for superclass and each subclass
CREATE TABLE shortcuts (shortcut, expandsTo)
CREATE TABLE permShortcuts (shortcut, owner)
CREATE TABLE expShortcuts (shortcut, expires)
other considerations

split tables up
keep immutables together
for security (eg, passwords)
frequency of updates

redundancy
add redundant fields to avoid joins

normal form?
almost always produces schemas in 3NF
exercise: data model to database
crime database schema

**Pair**
- **fst, snd**
- **since, last ?**

**Date**

**Criminal**
- recent

**Location**

- no \( p, p' : \text{Pair} \mid p.fst + p.snd = p'.fst + p'.snd \)
- all \( p : \text{Pair} \mid p.since \leq p.last \)

**design a database schema for this model**
**go through each step, submit just the final result**
deriving a schema

CREATE TABLE criminals (  
id int primary key,  
name varchar(255)  
)  

CREATE TABLE locations (  
id int primary key,  
zipcode char(5)  
)  

CREATE TABLE pairs (  
id int primary key,  
fst int references criminals(id),  
snd int references criminals(id),  
since date, last date  
)  

CREATE TABLE recents (  
criminal int references criminals(id),  
location int references locations(id),  
)
comparing notations
origins of our modeling notation

- logic diagrams (Euler, Venn, Peirce)
- ZF set theory
- relational calculus (Tarski)
- relational model (Codd)
- Z notation
- ER & other data models
- model checking
- Alloy Language
- object model notations (OMT etc)
- Unified Modeling Language
- Alloy Diagrams

Keywords:
- mathematical logic
- object-oriented development
- software verification
- relational databases

Timeline:
- 1700
- 1900
- 1940
- 1970
- 1980
- 1990
- 2000
cf: entity relationship diagram

- **Shortcut** expands to **URL**
- **User** owns **Permanent Shortcut**
- **User** owns **Expiring Shortcut**
- **Expiring Shortcut** from, to **Date**

**URL**
- relates: need not be binary
- 'crows feet' multiplicities
- no subsets
- **Shortcut** isPermanent
  - has attributes:
    - **From Date**, **To Date**
cf: unified modeling language

- User
- Shortcut
- Permanent Shortcut
- Expiring Shortcut
- Date
- URL

Classes with attributes & methods:
- User owns shortcuts
- Shortcut expandsTo URL
  - null association
  - named ends

From: Date
To: Date
automatic analysis of data models
alloy: a language & tool for relational models

about alloy

Alloy is a language for describing structures and a tool for exploring them. It has been used in a wide range of applications from finding holes in security mechanisms to designing telephone switching networks.

An Alloy model is a collection of constraints that describes (implicitly) a set of structures, for example: all the possible security configurations of a web application, or all the possible topologies of a switching network. Alloy's tool, the Alloy Analyzer, is a solver that takes the constraints of a model and finds structures that satisfy them. It can be used both to explore the model by generating sample structures, and to check properties of the model by generating counterexamples. Structures are displayed graphically, and their appearance can be customized for the domain at hand.

At its core, the Alloy language is a simple but expressive logic based on the notion of relations, and was inspired by the Z specification language and Tarski's relational calculus. Alloy's syntax is designed to make it easy to build models incrementally, and was influenced by modeling languages (such as the object models of OMT and UML). Novel features of Alloy include a rich subtype facility for factoring out common features and a uniform and powerful syntax for navigation expressions.

The Alloy Analyzer works by reduction to SAT. Version 4 was a complete rewrite that included Kodkod, a new model finding engine that optimizes the reduction, and a new front end.

news

ASM, Alloy, B and Z Conference: 6 new publications on Alloy.

A Japanese translation of book published!


Software Abstractions

Logic, Language, and Analysis

Revised edition

Daniel Jackson
alloy in action: the model

model, in textual form

```alloy
open util/relation
sig Address {}
abstract sig Object {}
sig Message extends Object {
  from, to: Address
}
sig Unread in Message {}
abstract sig Folder extends Object {
  contents: set Object
}
sig UserDefined, Predefined extends Folder {}
one sig Inbox, Sent, Trash extends Predefined {}

fact {
  -- contents in Folder lone -> Object
  no Predefined.contents & Folder
  -- no contents.Predefined & Folder
}

no Sent.contents & Unread
acyclic [contents, Object]
}

run {
  some contents.contents
  -- some Unread
} for 6
```
alloy: showing graphical model

diagram showing relationships between objects:
- Folder
  - UserDefined
    - Trash
  - Predefined
    - Sent
    - Inbox
  - Object
    - Message
      - Unread
      - Address

Graphical model, generated from text
alloy: an instance

generated automatically

UserDefined

contents

Inbox

contents

Message

to

from

Address

Sent

Trash

contents: 2
from: 1
to: 1

oops! we need more constraints
our graphical notation has a textual version is formal, so analyzable

in 6170 you’ll write textual constraints as addenda to diagrams can use Alloy if you like or just English...
family relationships (circa 1947)

abstract sig Person {
    father: lone Man,
    mother: lone Woman,
    parents: set Person
}

sig Man extends Person { wife: lone Woman }  // marriage constraints 1947
sig Woman extends Person { husband: lone Man }

fact {
    no p: Person | p in p.^{(mother+father)}  // can’t be your own ancestor
    wife = ~husband  // your husband’s wife is you etc
    no wife & *(mother+father).mother  // don’t marry your mother etc
    no husband & *(mother+father).father  // don’t marry your father etc
    parent = mother + father + father.wife + mother.husband  // define parents
}

run { } for 4

exercise for the reader: update this model to reflect our society’s more inclusive notions

note: your parent is your mother, father or their spouse
first instance Alloy finds

Man0 is Man1’s parent, and vice versa!
a similar instance...

https://www.youtube.com/watch?v=eYlJH81dSiw

song by Dwight Latham & Moe Jaffe (1947)
the only YouTube video that shows an instance of a data model being constructed to music?