LOGIC & LANGUAGE

Daniel Jackson · Lipari Summer School · July 18-22, 2005
handshaking solution
what this lecture’s about

a relational logic
  › first-order logic + relational operators

Alloy language
  › signatures & fields
  › constraint packaging

mostly review? but ...
  › generalized join
  › sets & scalars as relations
  › first-order puns
introduction
why logic?

simplicity
  › close to phenomena being described
  › familiar syntax & semantics

one language
  › for system & properties
  › model checking research:
    focus on property language; machine language ignored

declarative
  › the more you add, the less happens
  › so good for partial descriptions (esp. environment)
  › and good for incremental modelling
imperative vs. declarative

anything can happen

…

…

nothing can happen

declarative

imperative
why relations?

simplest way to talk about structure?
▷ just like references in OOP

There is no problem in computer science that cannot be solved by an extra level of indirection

-- David Wheeler
3 logics

“everybody loves a winner”

predicate logic
\[ \forall w \mid \text{Winner}(w) \Rightarrow \forall p \mid \text{Loves}(p,w) \]

relational calculus
\[ \text{Person} ; \text{Winner}^{-1} \subseteq \text{loves} \]

my relational logic
\[ \text{all } p: \text{Person}, w: \text{Winner} \mid p \rightarrow w \text{ in loves} \]
\[ \text{Person } \rightarrow \text{ Winner } \text{ in loves} \]
\[ \text{all } p: \text{Person} \mid \text{Winner in } p.\text{loves} \]
semantic basis: relations
everything’s a relation

Z based on ZF set theory

Alloy based on relational calculus
atoms & relations

**atoms** are individuals that are

- **indivisible**
  - can’t be broken into smaller parts
- **immutable**
  - don’t change over time
- **uninterpreted**
  - no built-in properties

A **relation** is a table

- set of tuples of atoms
- **arity** = number of columns, $\geq 1$
- **size** = number of rows, $\geq 0$
in standard set theory: scalar \( a \), tuple \( (a) \), set \( \{a\} \), relation \( \{(a)\} \)

in Alloy logic: scalar, tuple, set, relation \( \{(a)\} \)
typical kinds of relation

containment
  msgs: Buffer -> Message

grouping
  group: Graphic -> Group
  sameGroup: Graphic -> Graphic

indirection
  style: Paragraph -> Style

naming
  addr: Alias -> Address

ordering
  next: Time -> Time
constants, ops, quantifiers: a lightning tour
constants

universal  univ  \{(x) \mid x \text{ is an atom}\}

identity  iden  \{(x,x) \mid x \in \text{univ}\}

empty     none  \{\}
set operators

union \quad p + q \quad \{t \mid t \in p \lor t \in q\}

difference \quad p - q \quad \{t \mid t \in p \land t \notin q\}

intersection \quad p \& q \quad \{t \mid t \in p \land t \in q\}

subset \quad p \text{ in } q, p : q \quad \{(p_1, \ldots, p_n) \in p\} \subseteq \{(q_1, \ldots, q_n) \in q\}

equality \quad p = q \quad \{(p_1, \ldots, p_n) \in p\} = \{(q_1, \ldots, q_n) \in q\}

File + Dir, Object - Dir, Open & File
File in Object
Object = File + Dir + Alias
brother + sister
sister in sibling
root: Dir
arrow product

\[ p \rightarrow q \quad \{(p_1, \ldots, p_n, q_1, \ldots, q_m) \mid (p_1, \ldots, p_n) \in p \land (q_1, \ldots, q_m) \in q\} \]

alice -> bob
Person -> Winner
univ -> univ
alice -> bob in loves
f -> root in dir
dir: Object -> Dir
arrow idioms

when $s$ and $t$ are sets
\[ s \rightarrow t \] is their cartesian product
\[ r: s \rightarrow t \] says $r$ maps atoms in $s$ to atoms in $t$

when $x$ and $y$ are scalars
\[ x \rightarrow y \] is a tuple
dot & box join

\[ p \cdot q \{ (p_1, \ldots, p_{n-1}, q_2, \ldots, q_m) \mid (p_1, \ldots, p_n) \in p \land (p_n, q_2, \ldots, q_m) \in q \} \]

- alice \{ (ALICE) \}
- loves \{ (ALICE, BOB), (ALICE, CAROL), (CAROL, ALICE) \}
- alice.\text{loves} \{ (ALICE, CAROL) \}
- \text{loves.\text{alice}} \{ (CAROL) \}
- \text{loves.\text{loves}} \{ (ALICE, ALICE) \}

\[ \text{p.expr} [q] = q. (p.\text{expr}) \]

\[ \text{loves.\text{loves}}[\text{alice}] \{ (ALICE) \} \]
join idioms

when \( p \) and \( q \) are binary relations
\[ p \cdot q \text{ is standard relational composition} \]

when \( r \) is a binary relation and \( s \) is a set
\[ s \cdot r \text{ is relational image of } s \text{ under } r \text{ (‘navigation’) } \]
\[ \text{univ}.r \text{ is the range of } r \]
\[ r \cdot s \text{ is relational image of } s \text{ under } \sim r \text{ (‘backwards navigation’) } \]
\[ r.\text{univ} \text{ is the domain of } r \]

when \( f \) is a function and \( x \) is a scalar
\[ x \cdot f \text{ is application of } f \text{ to } x \]

what is \( x \cdot f \) when \( x \) outside domain of \( f \)?
the partial function tarpit

Romeo’s wife is Juliette or Romeo is unmarried

\( \text{romeo.wife} = \text{juliette} \text{ or } \text{romeo in Unmarried} \)

\( \triangleright \) true if Romeo has no wife?

approaches

\( \triangleright \) 3-valued logic (eg, VDM, OCL) [complex, no congruence]

\[ \text{maybe or true} \]

\( \triangleright \) all functions total (eg, Larch) [function not just a relation]

\[ ? = \text{juliette or true} \]

\( \triangleright \) undefined values (eg, OCL) [strictness]

\[ \text{undefined} = \text{juliette or true} \]

\( \triangleright \) partial semantics (eg, Z) [complex, no congruence]

\[ ? \text{ or true } \Rightarrow ? \]

\( \triangleright \) bad applications are false (eg, Parnas) [\( x \neq y, \neg x = y \text{ differ} \)]

\[ \text{false or true} \]
joins on multirelations

given a relation on books/aliases.Addresses
  addr \{ (B0,A0,D0), (B0,A1,D1), (B1,A1,D2), (B1,A2,D3) \}
  b \{ (B0) \}
  a \{ (A0) \}
  d \{ (D3) \}

we have
  b.addr \{ (A0,D0), (A1,D1) \}
  b.addr[a] \{ (D0) \}
  addr.d.univ \{ (B1) \}
playing with the analyzer
### Other Handy Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transpose</td>
<td>~p</td>
<td>{ (p_n, ... p_1)</td>
</tr>
<tr>
<td>Transitive Closure</td>
<td>^p</td>
<td>Smallest q</td>
</tr>
<tr>
<td>Restriction</td>
<td>s &lt;: p</td>
<td>{ (p_1, ... p_n)</td>
</tr>
<tr>
<td>Domain</td>
<td>dom p</td>
<td>{ (p_1)</td>
</tr>
<tr>
<td>Override</td>
<td>p ++ q</td>
<td>q + (p - dom q \subseteq p)</td>
</tr>
</tbody>
</table>
quantifiers & cardinalities

quantifiers
  \texttt{all, some, no, one, lone}

quantified formulas
  \texttt{all} \ x: \ e \ | \ F \quad \land_{v \in x} \ F \ [\{(v)\}/x]

cardinality expressions
  \#e \quad \text{size of relation e}
  \texttt{no} \ e \quad \#e = 0
  \texttt{some} \ e \quad \#e > 0
  \texttt{lone} \ e \quad \#e \leq 1
  \texttt{one} \ e \quad \#e = 1
declarations & multiplicity

multiplicity keywords: some, one, lone, set

set declarations

\[
s: m \rightarrow s \subseteq e \land m \rightarrow e
\]

\[
s: e \rightarrow s: one \rightarrow e
\]

relation declarations

\[
r: e \rightarrow m \rightarrow n \rightarrow e' \rightarrow r \subseteq e \times e' \land \forall x: e \mid n \rightarrow x.r \land \forall x: e' \mid m \rightarrow r.x
\]

examples

alice: Winner
Winner: set Person
loves: Person some -> some Person
root: Dir
dir: (Object - root) -> one Dir
puns

to support familiar declaration syntax
  › Alloy declaration  \( r : A \rightarrow B \)
  › has traditional reading  \( r \in 2^{(A \times B)} \)
  › has Alloy reading  \( r \subseteq A \times B \)

to support ‘navigation expressions’
  › Alloy expression  \( x.f.g \)
  › has traditional reading  \( g(f(x)) \) unless \( f(x) \) undefined or a set
  › has Alloy reading  image (image({(x)}, f), g)
summary of features

simple syntax
▶ same operators for sets, scalars, relations
▶ no lifting \{x\}
▶ conventional, but puns

simple semantics
▶ no undefined expressions: all operators total
▶ two-valued logic

why does this work?
▶ first order: no sets of sets needed
▶ no boolean expressions
language
structure of an alloy model

signatures & fields
  › introduces sets and relations
  › ‘extends’ hierarchy for classification & subtypes

constraints paragraphs
  › facts: assumed to hold
  › predicates: reusable constraints
  › functions: reusable expressions
  › assertions: conjectures to check

commands
  › run: generate instances of a predicate
  › check: generate counterexamples to an assertion
instances

alloy analyzer is a **model finder**
› finds solutions to constraints
› run predicate: solution is instance
› check assertion: solution is counterexample

solution
› assignment of relational values to variables
› variables are
   sets (signatures)
   relations (fields)
   skolem constants (witnesses, predicate arguments)
module examples/addressBook/addLocal

abstract sig Target {}

sig Addr extends Target {}

sig Name extends Target {}

sig Book {addr: Name -> Target}

fact Acyclic {all b: Book | no n: Name | n in n.^{(b.addr)}}

fun lookup (b: Book, n: Name): set Addr {n.^{(b.addr)} & Addr}

pred add (b, b': Book, n: Name, t: Target) {b'.addr = b.addr + n->t}

run add for 3 but 2 Book

assert addLocal {
  all b,b': Book, n,n': Name, a: Addr |
  add (b,b',n,a) and n != n' => lookup (b,n') = lookup (b',n') }

check addLocal for 3 but 2 Book
declarations

module examples/addressBook/addLocal

abstract sig Target {}
sig Addr extends Target {}
sig Name extends Target {}
abstract: Target in Addr + Name
extends: Addr in Target and Name in Target and no Addr & Name

sig Book {addr: Name -> Target}
addr: Book -> Name -> Target
module examples/addressBook/addLocal

abstract sig Target {}
sig Addr extends Target {}
sig Name extends Target {}
sig Book {addr: Name -> Target}

fact Acyclic {all b: Book | no n: Name | n in n.^{(b.addr)}}
fun lookup (b: Book, n: Name): set Addr {n.^{(b.addr)} & Addr}
pred add (b, b': Book, n: Name, t: Target) {b'.addr = b.addr + n->t}

run add for 3 but 2 Book
negating an assertion

assert addLocal { all b,b': Book, n,n': Name, a: Addr |
  add (b,b',n,a) and n != n' => lookup (b,n') = lookup (b',n') }
check addLocal for 3 but 2 Book

for analysis, equivalent to these:

pred addLocal () { some b,b': Book, n,n': Name, a: Addr |
  add (b,b',n,a) and n != n' and not lookup (b,n') = lookup (b',n') }
run addLocal for 3 but 2 Book

pred addLocal (b,b': Book, n,n': Name, a: Addr) { add (b,b',n,a) and n != n' and not lookup (b,n') = lookup (b',n') }
run addLocal for 3 but 2 Book
counterexample: textual

module examples/addressBook/addLocal

sig Target extends univ = {Addr_0, Name_0, Name_1}
sig Addr extends Target = {Addr_0}
sig Name extends Target = {Name_0, Name_1}
sig Book extends univ = {Book_0, Book_1}

    addr: {Book_0 -> Name_1 -> Name_0,
            Book_1 -> {Name_0 -> Addr_0, Name_1 -> Name_0}}

skolem constants

addLocal_b = {Book_0}
addLocal_b' = {Book_1}
addLocal_n = {Name_0}
addLocal_n' = {Name_1}
addLocal_a = {Addr_0}
counterexample: graphical

without customization
counterexample: graphical

with **Book** projected
how the analysis works
alloy analyzer architecture
relational values

space of values for relation in scope 2

space of values as 4 boolean vars
sample translation

example

\[ b, b': \text{Book} \]
\[ n: \text{Name} \]
\[ \text{names}: \text{Book} \rightarrow \text{Name} \]
\[ b'.\text{names} = b.\text{names} + n \]

compositional translation

\[ b \ [i] : \text{true if ith element of Book is in the set b} \]
\[ \text{names} \ [i, j] : \text{true if names maps ith element of Book to jth element of Name} \]
\[ b.\text{names} + n \ [i] = (\exists k. \ b \ [k] \land \text{names} \ [k, i]) \lor n \ [i] \]
\[ b'.\text{names} = b.\text{names} + n = \]
\[ \forall i. \ (\exists k. \ b' \ [k] \land \text{names} \ [k, i]) \leftrightarrow (\exists k. \ b \ [k] \land \text{names} \ [k, i]) \lor n \ [i] \]
quantification

grounding out

\textbf{all} \ x: \ t \ | \ F

becomes: \( F [x0/x] \ \textbf{and} \ F [x0/x] \ \textbf{and} \ ... \)

skolemization

\textbf{some} \ x: \ t \ | \ F

becomes: \( F [X/x] \) where \( X \) is a fresh free variable

\textbf{all} \ x: \ s \ | \ \textbf{some} \ y: \ t \ | \ F

becomes: \( \textbf{all} \ x: \ s \ | \ F [x.Y/y] \) where \( Y: \ s-> \ t \) is a free (function) var
optimizations

symmetry
› atoms of a signature are interchangeable
› so adds symmetry breaking predicates automatically
› for util/ordering, ordering is fixed $A_1, A_2, A_3, \ldots$

other optimizations
› sharing: detecting shared formulas before they appear during grounding out
› boolean simplifications
› careful conversion to CNF
› ‘atomization’ using subtypes
performance: SAT solvers

size of solvable constraint in #boolean variables

from Sharad Malik
performance: moore’s law

speed of main processor offering in MHz
from intel.com
homework
I'M MY OWN GRANDPA  (Dwight Latham & Moe Jaffe)

Many, many years ago when I was twenty-three
I was married to a widow who was pretty as could be ...

... Now if my wife is my grandmother, then I'm her grandchild
And every time I think of it, it nearly drives me wild
‘Cause now I have become the strangest case you ever saw
As husband of my grandmother, I am my own grandpa

I’m my own grandpa,
I’m my own grandpa,
It sounds funny, I know
But it really is so
I’m my own grandpa
self-grandpa in alloy

module examples/grandpa/grandpa1

abstract sig Person {father: lone Man, mother: lone Woman}
sig Man extends Person {wife: lone Woman}
sig Woman extends Person {husband: lone Man}

fact {
  no p: Person | p in p.^{mother+father}
  wife = ~husband
}

fun grandpas (p: Person): set Person {p.{mother+father}.father}

pred ownGrandpa (p: Person) {p in grandpas (p)}
run ownGrandpa for 4 Person
your task

find a solution to the song by

› changing the expression in the function grandpa
› adding any new constraints that seem necessary
free upgrades available!

new memory sticks for old!

ask me if you’d like:
› yesterday’s address book examples
› today’s updated lecture and examples