Lecture 2: A Relational Logic

and analysis with Alloy
declarative modelling of software
Oxford, home of Z

Pittsburgh, home of SMV

The Atlantic Divide

American school of formal methods

European school

- emphasis on verification algorithms (like SMV)
- emphasis on modelling (like Z, VDM, B, Alloy)

Alloy brings together

- automatic analysis (like SMV)
- logical notation (like Z)
First order effects

- Finite interpretation
- Generalized relational join operator
- No scalars or sets; all expressions are relation-valued
- Novel features
- No need to distinguish scalars from singleton sets
- No constructors; compositions by projection
- Design implications
- To allow exhaustive search

Alloy is first order
What's atomic in the real world?

- **Very little** -- a modelling abstraction
- **Immutable** -- don't change over time
- **Uninterpreted** -- can't be broken into smaller parts
- **Indivisible** -- atoms are
- **Atoms & relations** -- structures are built from atoms

```plaintext
Date 0  Date 1
tomorrow

State 0  State 1
next

contains
```

```plaintext
nextState 0
```
Employer, Employee in PERSON
  
Employer = \{BOB, CAROL\}
  
Employer = \{ALICE\}
  
\text{atoms that share properties share a type}

no subtyping, so

\{FILESYSTEM = \{FILESYSTEM0, FILESYSTEM2\}\}

\{STATE = \{STATE0, STATE1, STATE2\}\}

\{PERSON = \{ALICE, BOB, CAROL\}\}

\{DATE = \{JAN1, JAN2, \ldots, DEC31\}\}

partitioned into basic types, each a set

\text{a finite (but perhaps big) set}

\text{contains all atoms}

universe

\text{types}
Relations

definition
A relation is a non-empty set of tuples.

typing
A relation type is a non-empty list of basic types.

relations
- birthday: (PERSON, DATE)
  - Alice: May 1
  - Bob: January 4
  - Carol: December 9

- likes: (PERSON, PERSON)
  - Alice likes Bob
  - Bob likes Carol
  - Carol likes Bob
Relations as tables

- can view relation as table
- atoms as entries, tuples as rows
- order of columns matters, but not order of rows
- can have zero rows, but not zero columns
- no blank entries

Example

\[
\text{birthday} = \{(\text{ALICE, MAY1}), (\text{BOB, JAN4}), (\text{CAROL, DEC9})\}
\]
dimensions

- Elise heterogeneous
- Relation of type \( (T, \ldots, T) \) is homogeneous

- \( \#p \) is an integer expression giving the size of \( p \)
  \( \#p \in \mathbb{N}_0 \) \- number of rows
  \( \text{size} \)

- \( \#p \in \mathbb{N}_0 \) \- number of columns
  \( \text{arity} \)

- \( \#p \in \mathbb{N}_0 \) \- finite, \( \geq 0 \)
  \( \text{arity} \)

- \( \#p \in \mathbb{N}_0 \) \- finite, \( > 0 \)

- \( \#p \in \mathbb{N}_0 \) \- unary, binary, ternary for \( k = 1, 2, 3 \)
  \( \text{arity} \)

- \( \#p \in \mathbb{N}_0 \) \- relation of arity \( k \) is a \( k \)-relation
  \( \text{arity} \)

- \( \#p \in \mathbb{N}_0 \) \- relation of type \( (T, \ldots, T) \) is homogeneous
  \( \text{homogeneity} \)

- \( \#p \in \mathbb{N}_0 \) \- else heterogeneous
Relations as graphs can view 2-relation as graphs. Atoms as nodes, tuples as arcs.

\[ \text{likes} = \{(\text{alice, bob}), (\text{bob, carol}), (\text{carol, bob})\} \]

Example
sets and scalars

sets and scalars represented as relations
set: a unary relation
scalar: a unary, singleton relation

\{(a)\}, \{(a)\}, \{\{a\}\}, \{\{a\}\}, \{\{\{a\}\}\}

no distinction between
unlike standard set theory

\{\{\{a\}\}\}
Alice = \{(ALICE)\}
Employer = \{(ALICE)\}
\{(BOB), (CAROL)\}
Employer = \{(BOB), (CAROL)\}

PERSON = \{(ALICE), (BOB), (CAROL)\}

-- note (a)'s! Employee = \{(BOB), (CAROL)\}

scalars vs. relation
set: a unary relation
represented as relations
sets and scalars

sets and scalars
For associating binary relations with atoms:

\[
\{ (\text{BB0, ALICE, MAY1}), (\text{BB0, BOB, JAN4}), (\text{BB1, CAROL, DEC9}) \}
\]

For relationships involving 3 atoms:

\[
\{ (\text{Alice, Apple, $60k}), (\text{Bob, Biogen, $70k}) \}
\]

\[
\{ (\text{PERSON, COMPANY, SALARY}) \}
\]
left-type(likes) = right-type(likes) = PERSON

right-set(likes) = \{(bob, carol)\}

left-set(likes) = \{(alice, bob, carol)\}

\{likes = \{(alice, bob), (bob, carol), (carol, bob)\}\} examples

left and right types

\langle \text{left (right) type of } p \text{ is the first (last) basic type of } p \rangle \text{ type}

\langle \text{left and right types} \rangle

left and right sets

\langle \text{left (right) set of } p \text{ is set of atoms in left-(right-)most column} \rangle

left and right sets
set operators

standard set operators

\[ p \text{ and } q \text{ contain same set of tuples} \]
\[ p \subset q \text{ if } p \text{ in } q \]
\[ p = q \text{ if } p \text{ and } q \text{ contain same set of tuples} \]

interpretation of +

- for scalars, makes a set
- for sets, makes a new set
- for relations, combines maps

\[ \text{likes} + \text{Alice} \rightarrow \text{Bob} \]

\[ \text{Employer} + \text{Employee} \]

\[ \text{Alice} + \text{Bob} \]

\[ \text{contains tuples in } p \text{ but not in } q \]

\[ \text{contains all tuples in both } p \text{ and } q \]

\[ \text{contains tuples of } p \text{ and tuples of } q \]

difference \( p - q \)

intersection \( p \cap q \)

union \( p + q \)
For scalars \( a \) and \( b \), \( a \rightarrow b \) is a tuple.

For sets \( s \) and \( t \), \( s \rightarrow t \) is the Cartesian product.

Examples:

Birthday = Alice\( \rightarrow \) May1 + Bob\( \rightarrow \) Jan4 + Carol\( \rightarrow \) Dec9

Employee\( \rightarrow \) Employee in likes

If \( p \) contains \((p_1, \ldots, p_n)\) and \( q \) contains \((q_1, \ldots, q_m)\) then \( p \rightarrow q \) contains \((p_1, \ldots, p_n, q_1, \ldots, q_m)\).
join definition

if \( p \) contains \((p_1, \ldots, p_{n-1}, p_n)\) and \( q \) contains \((q_1, \ldots, q_{m})\) and \( p_n = q_1 \) then
\( p \cdot q \) contains \((p_1, \ldots, p_{n-1}, q_2, \ldots, q_m)\)

constraints

\[
\text{arity}(p) + \text{arity}(q) > 2 \\
\text{right-type}(p) = \text{left-type}(q)
\]

definition

\( b \cdot d \) then \( b \cdot d \) contains \((p_1, \ldots, p\_n-1, q_2, \ldots, q\_m)\)

and \( p_n = q_1 \)

and \( q \) contains \((q_1, \ldots, q\_m)\)

If \( p \) contains \((p_1, \ldots, p\_n)\)

join
Given join examples, we have

\[
\text{Alice} \cdot \text{likes} = \{(\text{BOB})\}; \quad \text{likes} \cdot \text{Alice} = \{} \}
\]

\[
\text{likes} \cdot \text{birthday} = \{(\text{ALICE}, \text{MAY1}), (\text{BOB}, \text{JAN4}), (\text{CAROL}, \text{DEC9})\}
\]

\[
\text{birthday} \cdot \text{likes} = \{(\text{ALICE, MARY1}), (\text{BOB, JAN4}), (\text{CAROL, DEC9})\}
\]

\[
\text{Alice} \cdot (\text{bb0.birthdayRecords}) = \{(\text{MAY1})\}
\]

\[
\text{bb0.birthdayRecords} = \{(\text{ALICE, MARY1}), (\text{BOB, JAN4})\}
\]

\[
\text{birthday} \cdot (\text{likes}) = \{(\text{ALICE, MARY1}), (\text{BOB, JAN4}), (\text{CAROL, DEC9})\}
\]

\[
(\text{bb0, Alice}) \cdot (\text{likes, birthday}) = \{(\text{ALICE, MARY1}), (\text{BOB, JAN4}), (\text{CAROL, DEC9})\}
\]
for binary relations \( p \) and \( q \), \( p \cdot d \) is standard join of \( p \) and \( q \) for binary relation \( r \) of type \( (S, T) \).

For set \( s \) and binary relation \( r \), \( s \cdot r \) is image of \( s \) under \( r \).

\[ r \cdot 1 \text{ is left-set of } r \]
\[ s \cdot r \text{ is right-set of } r \]

\( s \cdot r \) is image of \( s \) under \( r \).
Join variants for non-binary relations, join is not associative

Syntactic variants of join

\[ \text{Alice.bbo.birthdayRecords} \]
\[ \text{Alice.bbo::birthdayRecords} \]
\[ \text{Alice.bbo.[birthdayRecords]} \]

Binding power: :: most, then ., then []

\[ (b \cdot d) \cdot r = [r]b \cdot d \]
\[ (r \cdot b) \cdot d = r :: b \cdot d \]

Equivalent expressions

\[ [d]b = b :: d = b \cdot d \]
transpose

for relation \( r: (S,T) \)

\[ \sim r \text{ contains } (b,a) \text{ whenever } r \text{ contains } (a,b) \]

\[ \sim r \text{ has type } (T,S) \]

a theorem

for set \( s \) and binary relation \( r \),

\[ r.s = s.\sim r \]
Given

\( p \) contains \((a, q)\) and \( q \) does not map a

\( q \) contains \((a', q')\), or

\( b ++ d \)

for relations \( \mathcal{P}, \mathcal{B} : (S, T) \)
Closure for relation $r$: $(T,T)$

$^r = r + r.r + r.r.r + r.r.r.r + \ldots$ is smallest transitive relation $p$ containing $r$

$\ast r = \text{idem}(T) + r + r.r + r.r.r + r.r.r.r + \ldots$ is smallest reflexive & transitive relation $p$ containing $r$

Examples

- $\text{precedes} = \sim_{\neq} \text{next}$
- $\text{reaches} = \ast_{\text{connects}}$
- $\text{ancestor} = \text{parent}^\ast$

is smallest reflexive & transitive relation $p$ containing $r$

\[ \cdots + I.T + I.T + I.T + I.T + I.T + I.T + \cdots \]

is smallest transitive relation $p$ containing $r$

\[ \cdots + I.T + I.T + I.T + I.T + I.T + I.T + \cdots \]

for relation $r$: $(T,T)$

Closure
<table>
<thead>
<tr>
<th><strong>Operator</strong></th>
<th><strong>Function</strong></th>
<th><strong>Type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p &amp; q</code></td>
<td>Logical AND</td>
<td><code>(T)</code></td>
</tr>
<tr>
<td><code>\neg p</code></td>
<td>Logical NOT</td>
<td><code>(S)</code></td>
</tr>
<tr>
<td><code>p</code></td>
<td>Identity</td>
<td><code>(T)</code></td>
</tr>
<tr>
<td><code>p</code>, <code>\neg p</code></td>
<td>粘连</td>
<td><code>(T,T)</code></td>
</tr>
<tr>
<td><code>p \lor q</code></td>
<td>Logical OR</td>
<td><code>(S)</code></td>
</tr>
<tr>
<td><code>\exists p</code></td>
<td>Existential</td>
<td><code>(T)</code></td>
</tr>
<tr>
<td><code>\forall p</code></td>
<td>Universal</td>
<td><code>(T)</code></td>
</tr>
<tr>
<td><code>p \rightarrow q</code></td>
<td>Implication</td>
<td><code>(T)</code></td>
</tr>
</tbody>
</table>

**Notes:**
- `\forall p`: For all `p`
- `\exists p`: There exists `p`
- `p \rightarrow q`: If `p` then `q`
- `p \equiv q`: `p` if and only if `q`
navigation expressions from 2-relations and the operators + ^ * ~ interpret as path-sets
cousin = parent.sibling.~parent

daniel.sibling.sibling
= daniel.sibling.siblign

tim

claudia

emily spouse

spouse

example

follow p backwards

d~
d_

dv

follow p zero or more times

follow p once or more

follow p or q

follow p then q

interpret as path-sets

~ * + .

from 2-relations and the operators

navigation expressions
a navigation example

\[ \text{all } n: \text{Node} \mid n.\text{~source-queue-elt}.\text{from} = n \]

or equivalently

\[ \text{all } n: \text{Node} \mid n.\text{~source-queue-elt}.\text{from} \in \text{idem[Node]} \]

we can write

\[ \text{from field of that node emanating from a node have a } \text{all messages queued on links} \]

to say
negated operators

if-then-else expressions

\[ F \implies G \]
\[ F \implies G \text{ else } H \]
\[ F \lor G \]
\[ \{ F; G \} \]
\[ F \land G \]

standard connectives

logical operators

\[ e \in \alpha, e' \in \alpha \]
\[ \neg e \in \alpha, e' \in \alpha \]
\[ F \land G \]
\[ F \lor G \]
\[ F \implies G \]
\[ F \iff G \]
\[ F \leftrightarrow G \]
\[ F \rightarrow G \]
\[ F \rightarrow G, H \]

if-then-else expressions

\[ F \implies G \text{ else } H \]
\[ F \lor G \]
\[ \{ F; G \} \]
\[ F \land G \]

not F
set declarations

form

var : [set | option] setexpr

same meaning as

Employee : Person

not unary, so no scalar constraint

Employee is a subset of Person

p is a scalar in Person

examples

bb : Person -> Date

Employee : set Person

Employee is a subset of Person

p : Person

p is a scalar in Person

v : e in e and #v = 1

v : set e in e and #v <= 1

v : option e in e and #v <= 1

same meaning as

Employee : Person
Relation declarations

var : expr => [mult] expr

form

Examples

r : A -> i B
r is a partial function

r : A ?-> B
r is a total function

r : A i-> B
r is a bijection

multiplicity symbols

? zero or one
i exactly one
+
one or more

meaning

r : e0 -> e1
means

r : e0 m -> e1
r : e0 n <- e1
and n e1's for each e0, and m e0's for each e0,

r : e0 m -> e1
r : e0 n <- e1
for each e0,
What is an object model?

- A set of declarations drawn as a graph.
- Boxes denote sets, arcs represent relations.
- Parentless boxes have implicit type.

Example:

- **Person**: set PERSON
- **Company**: set COMPANY
- **Employee**: set Person
- **worksFor**: Employee →! Company

Diagram:

- `Company` → `Employee` → `Person`

Object models
comprehensions

example

\{ a, b : Person | a.parents = b.parents \& a \neq b \}

sibling = { a, b : Person | a.parents = b.parents \& a \neq b }

and \{ (a0) in e0, (a1) in e1, etc \}

such that \( E \) holds when \( v0 = \{(a0)\}, v1 = \{(a1)\}, \ldots \)

is the relation containing tuples \((ao,a1,\ldots)\)

\{ \{ v0 : e0, v1 : e1, \ldots \} | E \}

meaning

\{ \forall \text{ var : setexpr} | \ldots \text{ formula} \}

General form
Example

\[ \forall a: \text{Person} \mid a \in a.\text{parents} \]

(meaning)

\[ \forall v0: e0, v1: e1, \ldots \mid F \]

universal quantification

universal quantification
quantifiers

other quantifiers

$F \mid \cdots \mid \text{one } v_0: e_0, v_1: e_1, \ldots \mid F$ is equivalent to $F \mid \cdots \mid \text{all } v_0: e_0, v_1: e_1, \ldots \mid F$

$F \mid \cdots \mid \text{one } v_0: e_0, v_1: e_1, \ldots \mid F$ is equivalent to $F \mid \cdots \mid \text{all } v_0: e_0, v_1: e_1, \ldots \mid F$

\begin{align*}
F & \text{ holds for exactly one } x \in e \\
F & \text{ holds for at most one } x \in e \\
F & \text{ holds for no } x \in e \\
F & \text{ holds for some } x \in e \\
F & \text{ holds for all } x \in e 
\end{align*}

\text{note}
quantified expressions

for quantifier Q and expression e, make formula

\[ Q \ e \]

meaning

- **some** \( e \) e is non-empty \( \#e > 0 \)
- **no** \( e \) e is empty \( \#e = 0 \)
- **sole** \( e \) e has at most one tuple \( \#e \leq 1 \)
- **one** \( e \) e has one tuple \( \#e = 1 \)

example

**no** Man & Woman \( no \ person \ is \ both \ a \ man \ and \ a \ woman \)
sample quantifications

biological constraints

one eve: Person | in eve. ~mother

biblical constraints

no p: Person | some p.spouse & p.siblings

no p: Person | sole p.spouse

all p: Person | sole p.spouse

all p: Person | one p.mother

biological constraints

no p: Person | p in p.parents

all p: Person | one p.mother

all p: Person | one p.mother

cultural constraints

no p: Person | eve.*~mother
summary: doing more with less

everything's a relation

(a, b) \in r

and

a \neq b

r

first-order operators

r : A \rightarrow B

means i \in A \Rightarrow B replaces i \in p(A \times B)

dot operator

plays many roles
dot operator

inexpressive

inexpressive

tractable

tractable

expressive

expressive
write a constraint on an undirected graph that says it is acyclic.
a solution
sample graph
higher-order quantifiers

examples

\[
\begin{align*}
\forall \mathcal{F} : \mathcal{R} \mid F \text{ holds for all } x \in \mathcal{R} \\
\forall : \mathcal{S} \mid \mathcal{F} \text{ holds for all } s = \mathcal{S}' \text{ where } \mathcal{S}' \in \mathcal{S} \\
\forall : \text{option} \mathcal{S} \mid \mathcal{F} \\
\forall : \text{set} \mathcal{S} \mid \mathcal{F}
\end{align*}
\]
model checking

- keep counters, discard model or vice versa?
- but in software, essence is incremental modelling
- emphasis is finding showstopper flaws
- culture of model checking

- missing at operation level
- modularity

- fixed topology of processes
- not suited for abstract schemes
- built-in communications
- no transitive closure, etc
- must encode in records, arrays
- only low-level datatypes

model checking