Alloy - Logic

Slides has content taken from the Alloy Tutorial by G. Dennis & R. Seater
(see alloy.mit.edu)

Relations

• All relations are first-order (a relation cannot contain another relation)
  – Sets of sets not allowed

sig Book {addr: Name -> lone Addr}

A valid value (a multirelation):

addr = {
    (B0, N0, A0),
    (B0, N1, A1),
    (B1, N1, A2),
    (B1, N2, A2)
}
Domain, Range of Relations

- Like Z we can talk about the domain and range of a relation.
  - The domain is the set of elements in the first column
  - The range is the set of elements in the last column

addr = {(B0, N0, A0), (B1, N1, A0), (B0, N2, A2)}
Domain of addr is {(B0),(B1)}
Range of addr is {(A0),(A2)}

Alloy Operators

- Two types of operators
  - Set operators (union, intersection, etc)
    - The tuple structure has no effect on these operators
  - Relational operators
    - Operators that affect the tuple structure
Set Operators

- + union
- & intersection
- - difference
- in subset
- = equality

Relational Operators

- -> arrow operator: cross product
- . dot operator: relation composition
- [] box – relation composition
- ~ transpose
- ^ transitive closure
- * reflexive transitive closure
- <: domain restriction
- :> range restriction
- ++ override
**Arrow Operator**

\[ \rightarrow \quad \textit{cross product} \]

Name = \{(N0), (N1)\}
Addr = \{(A0), (A1)\}
Book = \{(B0)\}

Name->Addr = \{(N0, A0), (N0, A1), (N1, A0), (N1, A1)\}
Book->Name->Addr = \{(B0, N0, A0), (B0, N0, A1), (B0, N1, A0), (B0, N1, A1)\}

b = \{(B0)\}
b' = \{(B1)\}
address = \{(N0, A0), (N1, A1)\}
address' = \{(N2, A2)\}

b->b' = \{(B0, B1)\}
b->address + b'->address' = \{(B0, N0, A0), (B0, N1, A1), (B1, N2, A2)\}

---

**Composition**

\[ p \cdot q = \]

\[ x.f = \]

Alloy Lecture 1
sig Book {addr: Name -> lone Addr}
pred show [b: Book] {
    #b.addr > 1
    #Name.(b.addr) > 1
}

Book = {(B0), (B1)}
Name = {(N0), (N1), (N2)}
Addr = {(A0), (A1), (A2)}

d = (N0, A0)
addr = {(B0, N0, A0), (B1, N1, A0), (B0, N2, A2)}

{(B0)}.addr = ???
Name.({(B0)}.addr) = ???
Useful composition facts

- When \( x \) is a scalar (e.g., \( \{(N0)\} \)) and \( r \) is a binary relation (e.g., \( \{(N0,A1), (N0,A2), (N1, A1)\} \)) then \( x.r \) is the set of elements that \( r \) maps \( x \) to (i.e., \( \{(A1),(A2)\} \))

- When \( f \) is a function and \( x \) is a scalar in the domain of \( f \), \( x.f \) is the scalar that \( f \) maps \( x \) to
  - That is \( x.f \) is like function application \( f(x) \)
  - Note that if \( x \) is not in the domain of \( f \) then \( x.f \) returns an empty set

\[
\neg = \neg + (\neg \ast \neg) + (\neg \ast \neg) + \ldots
\]

\[
\ast = \text{idem} + \neg
\]

```alloy
def Node = \{(N0), (N1), (N2), (N3)\}
def next = \{(N0, N1), (N1, N2), (N2, N3)\}
def \text{\neg next} = \{(N1, N0), (N2, N1), (N3, N2)\}
def \text{\ast next} = \{(N0, N1), (N0, N2), (N0, N3), (N1, N1), (N1, N2), (N1, N3), (N2, N2), (N2, N3), (N3, N3)\}
def first = \{(N0)\}
def rest = \{(N0), (N2), (N3)\}
def first.\text{\ast next} = \text{rest}
def first.\text{\neg next} = \text{Node}
```
### Constants

- **none** - empty set
- **univ** - universal set
- **iden** - identity relation

```alloy
def Name = { (N0), (N1), (N2) }
def Addr = { (A0), (A1) }
def none = {} 
def univ = { (N0), (N1), (N2), (A0), (A1) } 
def iden = { (N0, N0), (N1, N1), (N2, N2), (A0, A0), (A1, A1) }
```

The domain of a binary relation $r$ is $r.univ$
The range of a binary relation $r$ is $\text{univ}.r$
Boolean Operators

<table>
<thead>
<tr>
<th>!</th>
<th>not</th>
<th>negation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp; &amp;</td>
<td>and</td>
<td>conjunction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>=&gt;</td>
<td>implies</td>
<td>implication</td>
</tr>
<tr>
<td>,</td>
<td>else</td>
<td>alternative</td>
</tr>
<tr>
<td>&lt;=&gt;</td>
<td>iff</td>
<td>bi-implication</td>
</tr>
</tbody>
</table>

*four equivalent constraints:*

\[ F \implies G \iff H \]

\[ F \implies C \lor \neg H \]

\[ (F \& G) || ({\neg F} \& \& H) \]

\[ (F \land G) \lor (\neg F) \land I \]

Quantifiers

| all x: e | F | all F holds for every x in e |
| all x: e1, y: e2 | F | all F holds for at least one x in e |
| all x, y: e | F | all F holds for no x in e |
| all disj x, y: e | F | all F holds for at most one x in e |

| some n: Name, a: Address | a in n.address | some name maps to some address — address book not empty |
| no n: Name | n in n.^address | no name can be reached by lookups from itself — address book acyclic |
| all n: Name | lone a: Address | a in n.address | all F holds for exactly one x in e |
| every name maps to at most one address — address book is functional |

| all n: Name | no disj a, a': Address | (a + a') in n.address | no name maps to two or more distinct addresses — same as above |
| no name maps to two or more distinct addresses — same as above |
Set Declarations

- **set** any number
- **one** exactly one
- **lone** zero or one
- **some** one or more
  - RecentlyUsed: set Name
    - RecentlyUsed is a subset of the set Name
  - senderAddress: Addr
    - senderAddress is a singleton subset of Addr
  - senderName: lone Name
    - senderName is either empty or a singleton subset of Name
  - receiverAddresses: some Addr
    - receiverAddresses is a nonempty subset of Addr
  - **Note that x: e** is equivalent to **x: one e**

Relation Declarations

- workAddress: Name -> lone Addr
  - each alias refers to at most one work address
- homeAddress: Name -> one Addr
  - each alias refers to exactly one home address
- members: Name lone -> some Addr
  - address belongs to at most one group name and group contains at least one address
- **r: A -> B** is equivalent to **r: A set -> set B**
- (r: A m -> n B) is equivalent to ((all a: A | n a.r) and (all b: B | m r.b))
Quantified Expressions

**some** \( e \) - \( e \) has **at least one** tuple

**no** \( e \) - \( e \) has **no** tuples

**lone** \( e \) - \( e \) has **at most one** tuple

**one** \( e \) - \( e \) has **exactly one** tuple

- **some** \( \text{Name} \)
  - set of names is not empty

- **some** \( \text{address} \)
  - address book is not empty – it has a tuple

- **no** \( \text{Addr} - \text{Name} \)
  - nothing is mapped to addresses except names

- **all** \( n: \text{Name} \mid \text{lone} \ n.\text{address} \)
  - every name maps to at most one address

---

Set Comprehension

- \( \{ n: \text{Name} \mid \text{no} \ n.\text{^address} \ & \text{Addr} \} \)
  - set of names that don't resolve to any actual addresses

- \( \{ n: \text{Name}, a: \text{Address} \mid n -> a \ \text{in} \ \text{^address} \} \)
  - binary relation mapping names to reachable addresses
Set cardinality

• \( \#r \) - number of tuples in \( r \)
• \textbf{all} b: Bag | \( \#b\).marbles \(< 3 \)
  – \textit{all bags have 3 or less marbles}
• \( \#\text{Marble} = \textbf{sum} \) b: Bag | \( \#b\).marbles
  – \textit{the sum of the marbles across all bags equals the total number of marbles}
  – \textit{sum is an alloy operator}

Three logics in one language

• “Everybody loves a winner”
• Predicate logic style
  \[ \forall w | \text{Winner}(w) \Rightarrow \forall p | \text{Loves}(p, w) \]
  \[ \text{all} \ p: \text{Person}, \ w: \text{Winner} \mid p \rightarrow w \text{ in loves} \]
• Relational calculus
  \textit{Person} \times \textit{Winner} \subseteq \textit{loves}
  \textit{Person} \rightarrow \textit{Winner} \text{ in loves}
• Navigation expression
  \textbf{all} \ p: \text{Person} | \text{Winner in} \ p.\text{loves}