SWEN-220
Mathematical Models of Software

Alloy Logic
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
logic: everything's a relation

- sets are unary (1 column) relations
  
  $\text{Name} = \{(N0), (N1), (N2)\}$
  $\text{Addr} = \{(A0), (A1), (A2)\}$
  $\text{Book} = \{(B0), (B1)\}$

- scalars are singleton sets
  
  $\text{myName} = \{(N1)\}$
  $\text{yourName} = \{(N2)\}$
  $\text{myBook} = \{(B0)\}$

- binary relation
  
  $\text{names} = \{(B0, N0), (B0, N1), (B1, N2)\}$

- ternary relation
  
  $\text{addrs} = \{(B0, N0, A0), (B0, N1, A1), (B1, N1, A2), (B1, N2, A2)\}$
logic: relations

addr = {(B0, N0, A0), (B0, N1, A1),
        (B1, N1, A2), (B1, N2, A2)}

<table>
<thead>
<tr>
<th>row</th>
<th>column</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0</td>
<td>N0</td>
<td>A0</td>
</tr>
<tr>
<td>B0</td>
<td>N1</td>
<td>A1</td>
</tr>
<tr>
<td>B1</td>
<td>N1</td>
<td>A2</td>
</tr>
<tr>
<td>B1</td>
<td>N2</td>
<td>A2</td>
</tr>
</tbody>
</table>

arity = 3
size = 4

- rows are unordered
- columns are ordered but unnamed
- all relations are first-order
  - relations cannot contain relations, no sets of sets
logic: address book example

Name = \{ (N0), (N1), (N2) \}
Addr = \{ (A0), (A1), (A2) \}
Target = \{ (N0), (N1), (N2), (A0), (A1), (A2) \}
address = \{ (N0, A1), (N1, N2), (N2, A1), (N2, A0) \}
logic: constants

<table>
<thead>
<tr>
<th>none</th>
<th>empty set</th>
</tr>
</thead>
<tbody>
<tr>
<td>univ</td>
<td>universal set</td>
</tr>
<tr>
<td>iden</td>
<td>identity relation</td>
</tr>
</tbody>
</table>

Name = {(N0), (N1), (N2)}
Addr = {(A0), (A1)}

none = {}
univ = {(N0), (N1), (N2), (A0), (A1)}
iden = {(N0, N0), (N1, N1), (N2, N2), (A0, A0), (A1, A1)}
logic: set operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>union</td>
</tr>
<tr>
<td>&amp;</td>
<td>intersection</td>
</tr>
<tr>
<td>-</td>
<td>difference</td>
</tr>
<tr>
<td>in</td>
<td>subset</td>
</tr>
<tr>
<td>=</td>
<td>equality</td>
</tr>
</tbody>
</table>

**Examples:**

- greg = {N0}
- rob = {N1}

- greg + rob = {N0, N1}
- greg = rob = false
- rob in none = false

- Name = {(N0), (N1), (N2)}
- Alias = {(N1), (N2)}
- Group = {(N0)}
- RecentlyUsed = {(N0), (N2)}

- Alias + Group = {(N0), (N1), (N2)}
- Alias & RecentlyUsed = {(N2)}
- Name - RecentlyUsed = {(N1)}
- RecentlyUsed in Alias = false
- RecentlyUsed in Name = true
- Name = Group + Alias = true

- cacheAddr = {(N0, A0), (N1, A1)}
- diskAddr = {(N0, A0), (N1, A2)}

- cacheAddr + diskAddr = {(N0, A0), (N1, A1), (N1, A2)}
- cacheAddr & diskAddr = {(N0, A0)}
- cacheAddr = diskAddr = false
logic: product operator

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

\[ \text{Name} \rightarrow \text{Addr} = \{(N0, A0), (N0, A1), (N1, A0), (N1, A1)\} \]
\[ \text{Book} \rightarrow \text{Name} \rightarrow \text{Addr} = \{(B0, N0, A0), (B0, N0, A1), (B0, N1, A0), (B0, N1, A1)\} \]

\begin{align*}
b &= \{(B0)\} \\
b' &= \{(B1)\} \\
\text{address} &= \{(N0, A0), (N1, A1)\} \\
\text{address}' &= \{(N2, A2)\} \\
\text{b} \rightarrow \text{b}' &= \{(B0, B1)\} \\
\text{b} \rightarrow \text{address} + \text{b}' \rightarrow \text{address}' &= \{(B0, N0, A0), (B0, N1, A1), (B1, N2, A2)\} \end{align*}
logic: relational join

\[ p \cdot q \equiv \]

\[
\begin{array}{c}
\text{p} \\
(a, b) \\
(a, c) \\
(b, d)
\end{array}
\quad \sqrt{.} \\
\begin{array}{c}
\text{q} \\
(a, d, c) \\
(b, c, c) \\
(c, c, c) \\
(b, a, d)
\end{array}
\equiv
\begin{array}{c}
(a, c, c) \\
(a, a, d)
\end{array}
\]

\[ x.f \equiv \]

\[
\begin{array}{c}
\text{x} \\
(c)
\end{array}
\quad \sqrt{.} \\
\begin{array}{c}
\text{f} \\
(a, b) \\
(b, d) \\
(c, a) \\
(d, a)
\end{array}
\equiv
\begin{array}{c}
(a)
\end{array}
\]
## logic: join operators

<table>
<thead>
<tr>
<th>dot join</th>
<th>box join</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{Book} &= \{(B0)\} \\
\text{Name} &= \{(N0), (N1), (N2)\} \\
\text{Addr} &= \{(A0), (A1), (A2)\} \\
\text{Host} &= \{(H0), (H1)\} \\
\text{myName} &= \{(N1)\} \\
\text{myAddr} &= \{(A0)\} \\
\text{address} &= \{(B0, N0, A0), (B0, N1, A0), (B0, N2, A2)\} \\
\text{host} &= \{(A0, H0), (A1, H1), (A2, H1)\} \\
\text{Book.address} &= \{(N0, A0), (N1, A0), (N2, A2)\} \\
\text{Book.address[myName]} &= \{(A0)\} \\
\text{Book.address.myName} &= \{} \\
\text{host[myAddr]} &= \{(H0)\} \\
\text{address.host} &= \{(B0, N0, H0), (B0, N1, H0), (B0, N2, H1)\}
\end{align*}
\]

\[
\begin{align*}
e_{1}[e_{2}] &= e_{2}.e_{1} \\
a.b.c[d] &= d.(a.b.c)
\end{align*}
\]
logic: unary operators

\[ \sim \quad \text{transpose} \]
\[ ^\wedge \quad \text{transitive closure} \]
\[ \ast \quad \text{reflexive transitive closure} \]

(apply only to binary relations)

\[ ^\wedge r = r + r.r + r.r.r + \ldots \]
\[ \ast r = \text{idem} + ^\wedge r \]

Node = \{ (N0), (N1), (N2), (N3) \}
next = \{ (N0, N1), (N1, N2), (N2, N3) \}

\~next = \{ (N1, N0), (N2, N1), (N3, N2) \}
^\wedge next = \{ (N0, N1), (N0, N2), (N0, N3),
(N1, N2), (N1, N3),
(N2, N3) \}
\ast next = \{ (N0, N0), (N0, N1), (N0, N2), (N0, N3),
(N1, N1), (N1, N2), (N1, N3),
(N2, N2), (N2, N3), (N3, N3) \}

first = \{ (N0) \}
rest = \{ (N1), (N2), (N3) \}

first.^next = rest
first.\ast next = Node
logic: boolean operators

| ! | negation   |
| & & | conjunction |
| |   | disjunction |
| => | implication |
| <= > | bi-implication |

four equivalent constraints:

\[ \text{F} \implies \text{G} \text{ else } \text{H} \]

\[ \text{F implies G else H} \]

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

\[ (\text{F and G}) \; \text{or} \; ((\text{not F}) \; \text{and} \; \text{H}) \]
## Logic: Quantifiers

<table>
<thead>
<tr>
<th>Quantifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>all</strong></td>
<td>F holds for <em>every</em> ( x ) in ( e )</td>
</tr>
<tr>
<td><strong>some</strong></td>
<td>F holds for <em>at least one</em> ( x ) in ( e )</td>
</tr>
<tr>
<td><strong>no</strong></td>
<td>F holds for <em>no</em> ( x ) in ( e )</td>
</tr>
<tr>
<td><strong>lone</strong></td>
<td>F holds for <em>at most one</em> ( x ) in ( e )</td>
</tr>
<tr>
<td><strong>one</strong></td>
<td>F holds for <em>exactly one</em> ( x ) in ( e )</td>
</tr>
</tbody>
</table>

### Examples

- **all** \( x: e \mid F \)  
- **all** \( x: e_1, y: e_2 \mid F \)  
- **all** \( x, y: e \mid F \)  
- **all disj** \( x, y: e \mid F \)  

- **some** \( n: Name, a: Address \mid a \in n.\text{address} \)  
  Some name maps to some address — address book not empty

- **no** \( n: Name \mid n \in n.^\text{address} \)  
  No name can be reached by lookups from itself — address book acyclic

- **all** \( n: Name \mid \text{lone} a: Address \mid a \in n.\text{address} \)  
  Every name maps to at most one address — address book is functional

- **all** \( n: Name \mid \text{no disj} a, a': Address \mid (a + a') \in n.\text{address} \)  
  No name maps to two or more distinct addresses — same as above
logic: quantified expressions

\[
\begin{align*}
\text{some } e & \quad e \text{ has at least one tuple} \\
\text{no } e & \quad e \text{ has no tuples} \\
\text{lone } e & \quad e \text{ has at most one tuple} \\
\text{one } e & \quad e \text{ has exactly one tuple}
\end{align*}
\]

\[
\begin{align*}
\text{some } \text{Name} & \quad \text{set of names is not empty} \\
\text{some } \text{address} & \quad \text{address book is not empty} - \text{it has a tuple} \\
\text{no } (\text{address.Addr} - \text{Name}) & \quad \text{nothing is mapped to addresses except names} \\
\text{all } n: \text{Name} \mid \text{lone } n.\text{address} & \quad \text{every name maps to at most one address}
\end{align*}
\]
logic: comprehensions

\{x_1: e_1, x_2: e_2, \ldots, x_n: e_n \mid F\}

\{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 \rightarrow a \text{ in } \text{^address}\}
binary relation mapping names to reachable addresses
logic: if and let

```plaintext
f \text{ implies } e_1 \text{ else } e_2
let x = e | \text{ formula}
let x = e | \text{ expression}
```

four equivalent constraints:

```plaintext
all n: Name |
(some n.\text{workAddress} \\
    \text{ implies } n.\text{address} = n.\text{workAddress} \\
    \text{ else } n.\text{address} = n.\text{homeAddress})

all n: Name |
let w = n.\text{workAddress}, a = n.\text{address} |
(some w \text{ implies } a = w \text{ else } a = n.\text{homeAddress})

all n: Name |
let w = n.\text{workAddress} |
n.\text{address} = (some w \text{ implies } w \text{ else } n.\text{homeAddress})

all n: Name |
n.\text{address} = (let w = n.\text{workAddress} |
(some w \text{ implies } w \text{ else } n.\text{homeAddress}))
```
logic: cardinalities

#r  number of tuples in r
0,1,... integer literal
+  plus
-  minus

=  equals
<  less than
>  greater than
<=  less than or equal to
>=  greater than or equal to

\[ \text{sum } x: e \mid ie \]
sum of integer expression ie for all singletons x drawn from e

all b: Bag \mid \#b.marbles <= 3
all bags have 3 or less marbles

\#Marble = \text{sum } b: Bag \mid \#b.marbles
the sum of the marbles across all bags
equals the total number of marbles