CIS 771:
Software Specifications

Lecture: Alloy Logic (part B)

Outline

- Atoms and Relations
- Expressing structure with relations
- Terminology for relations
- Snapshots

Alloy Atoms

Atoms are Alloy’s primitive entities

- *indivisible*
  - it can’t be broken down into smaller parts
- *immutable*
  - its properties don’t change over time
- *uninterpreted*
  - it doesn’t have any built-in properties, the way that numbers do, for example.

Very few things in the world are truly atomic. An Alloy atom is a modeling abstraction -- it represents an entity whose details are irrelevant or that we simply don’t want to expose in our modeling and reasoning.

…if you want to expose some properties of atoms, you introduce relations to capture these properties as additional structure.

Related Concepts

A map is a model with atoms as abstractions

major cities

minor cities

towns

mountains

Related Concepts

A map is a model with atoms as abstractions

major cities

minor cities

towns

mountains

...details are hidden

Relations

A relation is a structure that relates atoms

- A relation (table)...
  - consists of a set of tuples (rows)
    - number of tuples is called *size*
    - any size is possible including size = 0
    - order of rows doesn't matter
  - each tuple is a sequence of atoms
    - all tuples must have the same length (arity)
    - order of atoms *does* matter

Set view

atoms

```
addr = {(B0,N0,A0),
        (B0,N1,A1),
        (B1,N2,A2),
        (B1,N3,A2)}
```

Table view

B0	N0	A0
B0	N1	A1
B1	N1	A2
B1	N2	A2

size = 4

arity = 3

In Alloy, all relations are first-order -- relations cannot contain relations, no sets of sets

Relations

Terminology

- Unary relation
  - arity = 1 (1 column)
- Binary relation
  - arity = 2 (2 columns)
- Ternary relation
  - arity = 3 (3 columns)
- Multirelation
  - arity >= 3
- Scalar
  - unary relation with one tuple

Every value in Alloy logic is a relation!

Examples

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

names = {(B0,N0),
         (B0,N1),
         (B1,N2)}

addr = {(B0,N0,A0),
        (B1,N3,A2)}

myName = {(N0)}
yourBook = {(B1)}
(all a:Addr | … a …)
```

All quantified variables in Alloy are bound to scalars

Representing Singletons

Since every value in Alloy logic is a relation, consider how "single values" are represented…

```
(all a:Addr | … a …)
```

All quantified variables in Alloy are bound to scalars, e.g., {(A0)}

{(N0)} {(A0)}

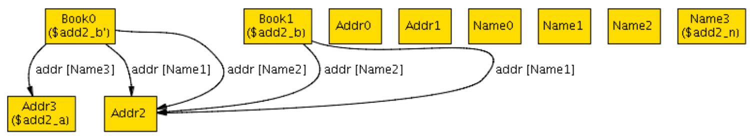
```
pred add (b,b':Book, n:Name, a:Addr)
  {b'.addr = b.addr + n->a}
```

{(N0,A0)}

Tuple expressions correspond to relations containing exactly one tuple (singleton relation)

For You To Do

```
pred add2(b, b': Book, n:Name, a:Addr) {
  no n.(b.addr)
  n.(b'.addr) = a
  all n1: (Name - n) | n1.(b.addr) = n1.(b'.addr)
}
```

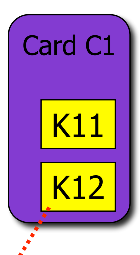


- For the operation and associated snap shot above, write (following the example on the previous slide) the sets/relations associated with *all* of the Alloy identifiers in the example above. *Be sure that each identifier is associate with a set of relations.*

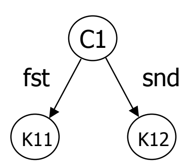
Expressing Structure

Although the only objects in the logic are indivisible atoms, you can model a composite object with atoms for the components and a relation(s) to bind them together.

A hotel key card with two cryptographic keys



Cryptographic keys



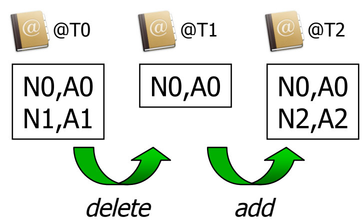

```
fst = {(C1,K11),(C2,K21)}
snd = {(C1,K12),(C2,K22)}
```

Expressing Structure

Although atoms are immutable, you can model mutation, in which the value of an object changes over time, by separating the identity of the object and its values into separate atoms, and relating identities, values, and times.

```
addrT = {(N0,A0,T0), (N1,A1,T0),
         (N0,A0,T1)
         (N0,A0,T2), (N2,A2,T2)}}
```

Consider a simplified version of the Address Book example in which the `addr` *map is extended with an additional column modeling the time at which each tuple is present in the address book.*



Expressing Structure

Although atoms are uninterpreted, you can give them properties by introducing relations between them.


```
abstract sig ThreatLevel {
  nextUp: ThreatLevel}
one sig Severe, High, Elevated, Guarded,
        Low extends ThreatLevel {}
```

```
fact threatOrder {
   Low.nextUp = Guarded
   Guarded.nextUp = Elevated
   Elevated.nextUp = High
   High.nextUp = Severe
   no Severe.nextUp
}
```

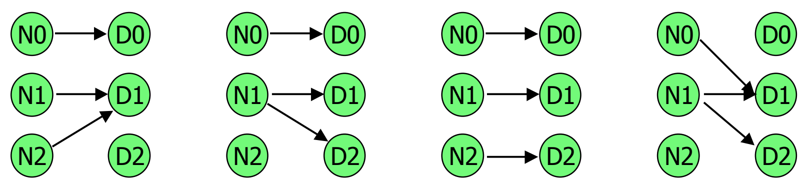
Consider a modeling of the homeland security threat levels using the Alloy "enumerated type" idiom. Even though we make a collection of signatures with appropriate names, the names are uninterpreted in the sense that there is no semantics associated with the ordering on threat levels

We can add a field `nextUp` *to realize the semantics of ordering, and then explicitly define the ordering.*

Functions and Injections

- A binary relation that maps each atom to at most one other atom is said to be *functional*, and is called a *function.*
- A binary relation that maps at most one atom to each atom is *injective*.

Examples



functional, but not injective | *injective, but not functional* | *functional, and injective* | *neither functional, nor injective*

Note: an empty relation is trivially functional and injective.

Domain and Range

- The *domain* of a relation is the set of atoms in its first column
- The *range* of a relation is the set of atoms in its last column

Example 1

```
address
  = {(N0,D0),
     (N1,D1),
     (N2,D1)}
domain (address)
  = {(N0),(N1),(N2)}
range (address)
  = {(D0),(D1)}
```

...binary relation

Example 2

```
addr
  = {(B0,N0,D0),
     (B0,N1,D1),
     (B1,N2,D2)}
domain (addr)
  = {(B0),(B1)}
range (address)
  = {(D0),(D1),(D2)}
```

...higher arity relation

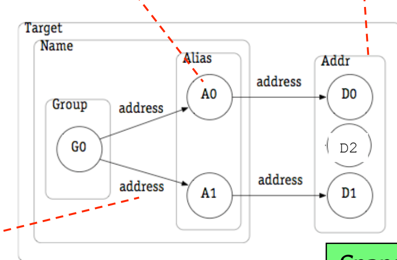
Alloy State Snapshot

Particular values of sets and *binary* relations can be shown in a ***snapshot***

```
address = {(G0, A0), (G0, A1), (A0, D0), (A1, D1)}
Target = {(G0), (A0), (A1), (D0), (D1), (D2)}
Name = {(G0), (A0), (A1)}
Alias = {(A0), (A1)}
Group = {(G0)}
Addr = {(D0), (D1), (D2)}
```

atom

set



relation

Snapshot

Snapshot Projections

Multirelations can be visualized as graphs via projections

```
addr = {(B0, G0, A0), (B0, G0, A1),
        (B0, A0, D0), (B0, A1, D1), (B1, A0, D1)}
Book = {(B0), (B1)}
```

Project over `Book`

First projection (B0): *select only those tuples that contain B0, then drop the book column for all tuples.*

```
B0.addr = {(G0, A0), (G0, A1), (A0, D0), (A1, D1)}
```

Second projection (B1): *select only those tuples that contain B1, then drop the book column for all tuples.*

```
B0.addr = {(A0, D1)}
```

relational join

For You To Do

- For each of the relations in the previous "For You To Do" exercise (dealing with operation add2),
  - state if the relation is injective and justify your answer
  - state if the relation is functional and justify your answer
  - state the domain of the relation
  - state the range of the relation

Acknowledgements

- The material in this lecture is based on Sections 3.1-3.2 from…
  - *Software Abstractions: Logic, Language, and Analysis,* Daniel Jackson, MIT Press, 2006.

CIS 771 --- Alloy Logic (part B)