CIS 771: Software Specifications

Lecture: Alloy Logic (part E)

Copyright 2007, John Hatcliff, and Robby. The syllabus and all lectures for this course are copyrighted materials and may not be used in other course settings outside of Kansas State University in their current form or modified form without the express written permission of one of the copyright holders. During this course, students are prohibited from selling notes to or being paid for taking notes by any person or commercial firm without the express written permission of one of the copyright holders.
Outline

- Declarations
- Set multiplicities
- Relational multiplicities
- Declaration constraints
- Nested multiplicities
- Cardinality constraints
A declaration introduces a name for a relation whose value is a subset of the value of the bounding expression appearing after the “:”

```
relation-name : expression
```

**Examples**

- `address : Name -> Addr`
  - maps names to addresses (representing a single address book)
- `addr: Book -> Name -> Addr`
  - maps books to names to addresses (representing a collection of address books)
- `address: Name -> (Name + Addr)`
  - maps names to names and addresses (representing a multilevel address book)

*Note: the bounding expression is usually formed with unary relations and the arrow operator, but any expression can be used.*
A *declaration* introduces a name for a relation whose value is a subset of the value of the *bounding expression* appearing after the “:”

\[
relation-name : expression
\]

**Examples -- with more complicated expressions**

- `address : (Alias + Group)->(Addr + Alias + Group)`
  - maps aliases and groups to addresses, aliases, and groups
- `address : (Alias->Group) + (Group->(Addr + Alias + Group)`
  - has the same “type” as the declaration above
    - aliases, groups in domain; addresses, aliases, groups in co-domain
  - more precise than the version above because it constrains aliases to only map to groups
  - illustrates how declaration expressions can combine relations (e.g., via the union operator) as well as sets
Set Multiplicities

If the *bounding expression* of a declaration denotes a set (is unary), it can be prefixed by a multiplicity keyword $m$ which constrains the size of set $x$ according to $m$.

```
x : [m e]
```

*Bounding expression with multiplicity m.*

**Multiplicy Keywords** -- similar to those used for quantification

- **set**
  - any number
- **one**
  - exactly one
- **lone**
  - zero or one
- **some**
  - one or more

For set-value bounded expressions, omitting the multiplicity keyword is the same as writing **one**.
Set Multiplicities

If the *bounding expression* of a declaration denotes a set (is unary), it can be prefixed by a multiplicity keyword $m$ which constrains the size of set $x$ according to $m$.

\[ x : m e \]

Examples

- RecentlyUsed: `set Name`
  - says that `RecentlyUsed` is a subset of the set `Name`

- `senderAddress: Addr`
  - says that `senderAddress` is a scaler in the set `Addr`

- `senderName: lone Name`
  - says that `senderName` is an option: either a scaler in the set `Name`, or empty

- `receiverAddresses: some Addr`
  - says that `receiverAddresses` is a non-empty subset of `Addr`. 
Examples of relational multiplicities in UML...

- Each class has one associated professor
- Each professor has one or more associated classes
- Each magazine has 0 or more subscribers
- Each person subscribes to 0 or more magazines
Relational Multiplicities

Examples of relational multiplicities in Entity-Relation models
Relational Multiplicities in Alloy

Multiplicities in Alloy can be placed on the elements of a relation declaration...

\[ r : S \, m \rightarrow \, n \, T \]

- A multiplicity \( m \) on the left (domain) tells you the size of the set associated with each range set element.
- A multiplicity \( n \) on the right (range) tells you the size of the set associated with each domain set element.
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.
Common Relations via Multiplicities

Examples

- \( r: A \rightarrow \text{one} \ B \)
  - A (total) function with domain \( A \), range \( B \)

- \( r: A \text{ one} \rightarrow B \)
  - An injective relation

- \( r: A \rightarrow \text{lone} \ B \)
  - A (partial) function

- \( r: A \text{ one} \rightarrow \text{one} \ B \)
  - An injective (total) function

- \( r: A \text{ some} \rightarrow \text{some} \ B \)
  - A surjective relation
For You To Do

- For each of the relations, give the most precise declaration with multiplicities.

(A)  
```
S0 → T0
S1 → T1
S2 → T2
S3 → T3
```

(B)  
```
S0 → T0
S1 → T1
S2 → T2
S3 → T3
```

(C)  
```
S0 → T0
S1 → T1
S2 → T2
S3 → T3
```

(D)  
```
S0 → T0
S1 → T1
S2 → T2
S3 → T3
```

- For each of the declarations below, draw two relations with the same domain / range as the relations above such that the first relation satisfies the multiplicities in the declaration while the second one violates it.
  - r: S some -> lone T
  - r: S set -> one T
  - r: S lone -> lone T
  - r: S some -> some T
Multiplicities are a Shorthand

Multiplicities are just a shorthand, and can be replaced by standard constraints...

\[ r : A \overset{m}{\rightarrow} n B \]

*can be written as...*

\[
\text{all } a : A \mid n a.r \\
\text{all } b : B \mid m r.b
\]

Example

members: Group \text{ lone } \rightarrow \text{ some } Addr

*can be written as...*

\[
\text{all } g : \text{ Group } \mid \text{ some } g.\text{members} \\
\text{all } a : \text{ Addr } \mid \text{ lone } \text{members.a}
\]
Generalizing to Tuples

In the schema below, $A$ and $B$ can be arbitrary expression, and don’t have to be relation names.

$$r : A^m \rightarrow nB$$

...in such a case, this says that $r$ maps $m$ tuples in $A$ to each tuple in $B$, and maps each tuple in $A$ to $n$ tuples in $B$.

Example

$$\text{addr: } (\text{Book} \rightarrow \text{Name}) \rightarrow \text{lone Addr}$$

...says that that relation $\text{addr}$ associates at most one address with each address book / name pair.

$\{ (B0, N0, A1) \\
(B0, N1, A2) \\
(B1, N0, A3) \} \quad \checkmark$

$\{ (B0, N0, A1) \\
(B0, N0, A2) \\
(B1, N0, A3) \} \quad \times$
Declaration Constraints

Declaration syntax can also be used to impose constraints on relations that have already be declared, or on arbitrary expressions...

*Original declaration...*

\[
\text{address}: \ (\text{Group} \ + \ \text{Alias}) \rightarrow \text{Addr}
\]

...imposing additional constraints somewhere later in the model...

\[
(\text{Alias} <: \text{address}) : \ \text{Alias} \rightarrow \text{lone} \ \text{Addr}
\]

...says that each alias maps to at most one address

Declaration constraints, like any other formula, can be combined with logical operators, placed inside the body of quantifications, etc.

\[
\text{all } b: \text{Book} \mid b.\text{addr}: \text{Name} \ \text{lone} \rightarrow \text{Addr}
\]

...says that each address book is injective (maps at most one name to an address)
Nested Multiplicities

Multiplicities can be nested

A declaration of the form...

$$r: A \to (B \ m \to n \ C)$$

...means that for each tuple in A, the corresponding tuples in $B \to C$ form a relation with the given multiplicity. In the case that A is a set, the multiplicity constraint is equivalent to...

$$\text{all } a: A \mid a.r : B \ m \to n \ C$$

Example

$$\text{addr: Book} \to (\text{Name lone} \to \text{Addr})$$

...says that, for any book, each address is associated with at most one name, and is equivalent to...

$$\text{all } b: \text{Book} \mid b.\text{addr}: \text{Name lone} \to \text{Addr}$$

...whereas...

$$\text{addr: (Book} \to \text{Name)}\text{lone} \to \text{Addr}$$

...says that each address is associated with at most book/name combination. The first (but not the second) allows an address to appear in more than one book

CIS 771 --- Alloy Logic (part E)
For You To Do

For each of the relation declarations below, give two relation instances such that the first relation satisfies the declaration while the second relation violates the declaration...

- \( \text{addr}: (\text{Book} \rightarrow \text{Name}) \rightarrow \text{some Addr} \)
- \( \text{addr}: \text{lone Book} \rightarrow (\text{Name} \rightarrow \text{Addr}) \)
- \( \text{addr}: \text{one Book} \rightarrow (\text{one Name} \rightarrow \text{some Addr}) \)

For each of the relation declarations above, write a declaration that does not include multiplicities but instead is accompanied by explicit constraint that achieve the same effect as the declarations above

Given the declaration below, write a declaration constraint that enforces the additional property that each group maps to one or more addresses

- \( \text{address}: (\text{Group} + \text{Alias}) \rightarrow \text{Addr} \)
Cardinality Constraints

The operator \# applied to a relation gives the number of tuples it contains, as an integer value.

Address: (Group + Alias) -> Addr

... 
=all g: Group | #g.address > 1

...says that every group has more than one address associated with it

addr: Book -> Name -> Addr

... 
=all a: Addr | #addr.a <= 5

...says that the number of book/name pairs associated with each address is less than or equal to 5
Sum Expression

The expression below denotes the integer obtained by summing the values of the integer expression \( ie \) for all values of the scalar drawn from the set \( e \).

\[
\text{sum } x : e \mid ie
\]

Example

\[
\text{addr : Book -> Name -> Addr}
\]

\[
\text{sum b : \{ b:Book \mid \#b.addr > 5 \} \mid \#b.addr}
\]

...gives the total number of entries across all books that have great than five entries
Acknowledgements

- The material in this lecture is based on Sections 3.6 and 3.7 from...