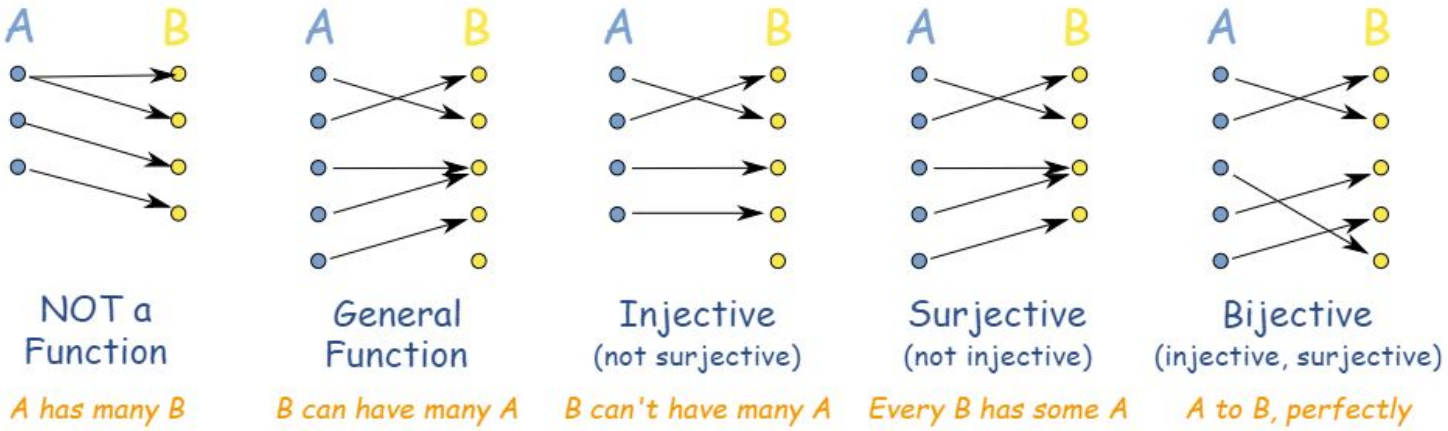
FOL (Chapter 2)

- Partial Function
  - For some value in x, there is a corresponding y
  - every element of the function's codomain is the image of *at most* one element of its domain
- Total Function
  - For all value in x, there is a corresponding y
- Partial Injective Function
  - Every element in y is the image of at most one element of x
- Total Injective Function
  - Every element in y is the image of at most one element of x and there is not a element in x that does not have an y
- Partial Surjective Function
  - For all value in y, there is a corresponding x
- Total Surjective Function
  - For all value in y, there is a corresponding x (x ois partial)
- Bijective Function
  - For all value in x, there is a corresponding distinct y



Alloy Logic (Chapter 3)

- Predicate calculus / First Order Logic (FOL)
    - terms 't' are variables
    - atomic formulas are either term comparisons 't1 = t2' and
    - 't1 != t2' or n-ary predicate tests 't1 -> ... -> tn in P' and
    - 't1 -> ... -> tn not in P'
    - Can use #
    - formulas are composed by
        - boolean connectives 'not', 'and', 'or' and 'implies'
        - quantifications 'all' and 'some' over unary predicates
- Relational calculus / Relational logic
    - Extends FOL with
        - expression comparisons 'e1 in e2' and 'e1 = e2'
        - expression multiplicity tests 'some e', 'lone e', 'no e' and 'one e'
        - binary relational operators '.', '->', '&', '+', '-', ':>' and '<:'
        - unary relational operators '~', '^' and '*'
        - definition of relations by comprehension ({x1: e1, x2: e2, ..., xn: en | F})
        - dot join operation between 2 sets/signatures
- Navigational expression
    - Dot join to access object fields

Alloy Language (Chapter 4)

- Signature
    - Extends: Disjoints subsets
    - Abstract: Sig is union of its extensions
    - May have relations
    - May be declared as a union of sets
- Constraints Paragraphs
    - fact: Always true for any generated instance
    - Function (fun): Reusable expression with 0 or more arguments
    - predicate: Reusable constraint with 0 or more arguments
- Assertions and commands
    - Assert: constraint that should follow from the facts
    - Commands
        - check: tries to find counter-example for an assertion
        - run: attempts to find a solution for a predicate

Operators

- A = { (a), (b), (c), (d), (e) }
- B = { (a), (b) }
- C = { (a, b), (b, c), (e, a) }
- D = { (a, b), (a, c), (c, a), (d, e) }
    - .
        - C.D = {(a, b), (b, c), (e, a)} . {(a, b), (a, c), (c, a), (d, e)} = {(b, a), (e, b), (e, c)}
        - Tip: match second element of C with first element of D and keep the ones on the outside
    - []
        - C[B] = B.C = {(a), (b)} . {(a, b), (b, c), (e, a)} {(b), (c)}
    - + union
    - & intersection
        - C & D = {(a, b)}
    - - difference
    - ->
    - ~ transpose
        - For a binary relation:
            - ~R = {x, y | y -> x in R}
            - i.e. ~R = inverse of R (often written R-1)
    - ^ transitive closure
        - ^C = {(a, b), (a, c), (b, c), (e, a), (e, b), (e, c)}
        - Tip: draw a directed graph and see for each node where you can reach. In this example draw edges from C: a->b, b->c, e->a
    - * reflexive and transitive closure
        - N.edge is the set of all nodes that N connects to. N.edge.edge, …
        - If we want every node that is connected to N: N.^edge
        - ^ does not include the original atom unless it's transitively reachable
        - If we want to also include N, use N.*edge instead
    - <: domain restriction
        - B <: D = {(a, b), (a, c)}
    - :> range restriction
        - C :> A = {(a, b), (b, c), (e, a)}
        - C :> B = {(a, b), (e, a) }
    - ++ override
        - D ++ C = {(c, a), (d, e), (a, b), (b, c), (e, a)}
        - {(a, b), (a, c)} ++ {(a, a)} = {(a, a)}
        - Tip: rel1 ++ rel2 = union but if a tuple from rel1 shares a "key" with a tuple of rel2, that tuple (of rel1) is dropped (merging dictionaries)
    - Mix
        - C - (A -> B) = {(a, b), (b, c), (e, a)} - {(a, a), (a, b), (b, a), (b, b), ...} = {(b, c)}
        - (^D & iden).univ =
            - ({(a, b), (a, c), (a, a), (c, a), (c, c)} & iden).univ = {(a, a), (c, c)}.univ = {(a), (c)}

- Common operators:

Multiplicity constraints in declarations

Set declarations with multiplicities 76	
e is a expression producing a set (arity 1)	
x: set e	x a subset of e
x: lone e	x empty or a singleton subset of e
x: some e	x a nonempty subset of e
x: one e	x a singleton subset of e (i.e. a scalar)
x: e	x a singleton subset of e (equivalent to one)

Set operators 52		
Symbol	Name	Result
+	Union	A set
&	Intersection	
-	Difference	
in	Subset	T or F
=	Equality	

Relation operators 55		
Symbol	Name	Syntax
->	(Arrow) product	R1 -> R2
.	Join	R1 . R2
[]	Join (a second notation for it)	R2 [R1]
~	Transpose	~ R
^	Transitive closure	^ R
*	Reflexive transitive closure	* R
<:	Domain restriction	Set <: R
:>	Range restriction	R :> Set
++	Override	R1 ++ R2

Logical operators 69		
Symbol	Keyword	Name or result
!	not	negation
&&	and	conjunction
||	or	disjunction
=>	implies	implication
<=>	iff	logical equivalence
	else	A=>B else C ≡ (A&&B)||(!A&&C)

Cardinality constraints

80 The prefix operator # (cardinality) on a relation produces the relation's size. expressions.

let 73	
let x = e | A	A with every occurrence of x replaced by expression e

Quantifiers/predicates 70		
	Quantification Q var:set | formula	Predicate on relations Q e
all	universal	—
some	existential	size is 1 or greater
no	¬∃	size is 0
lone	zero or one exists	size is 0 or 1
one	exactly one exists	singleton

Disjointness

71 disj before a list of variables restricts their bindings to be disjoint.

.	dot (join)	none	empty set
[]	box (join)	univ	universal set
		iden	identity relation
f[x]	same as x.f		

Signatures

Signatures 91	
sig A {*fields*}	Declares a set A of atoms
sig A extends B {*fields*}	Declares a subset A of set B, disjoint from all other extends subsets of B
sig A in B {*fields*}	Declares a subset A of B
sig A in B + C {*fields*}	Declares a subset A of the union (+) of sets B and C
abstract sig A {*fields*}	Declares a set A that contains no atoms other than the ones in its subsets (if any)
one sig A {*fields*}	Declares a singleton set A
lone sig A {*fields*}	Declares a set A of 0 or 1 atom
some sig A {*fields*}	Declares a nonempty set A
sig A, B {*fields*}	Declares two sets A and B of atoms Wherever A appeared above, a list of names can appear

Fields (in a signature for set A) 95	
f: e	Declares a relation f that's a subset of A->e. e can be any expression that produces a set — union, intersection, ... , any combination.
f: lone e	Each A is related to no e or one e.
f: one e	Each A is related to exactly one e.
f: some e	Each A is related to at least one e.
f: g->h	Each A is related to a relation from g to h.
f: one g lone -> some h	The multiplicities have their usual meanings. Here, each A is related to exactly one relation relating each g to 1 or more h's, and each h is related to 0 or 1 g.