**Set Theory**

* **Set theory – syntax**
* Set – a collection of elements/members
  + A set is unordered can be infinite
  + The elements of a set must be distinct
  + x ∈ S → an infix predicate meaning element x is in the set S, “x belongs to S”
* Set enumeration – defining a set by listing its elements
  + E.g. {A, B, C}
* Set comprehension – defining a set by using a characteristic predicate
  + {term | predicate} – elements are the terms for which the predicate evaluates to T
  + E.g. {x | x ∈ N ∧ x < 6}
* Recall that types in predicate logic are sets
  + **Axioms**:
    - If ∃x . x ∈ B then
    - (∀x : B . P(x)) ⇔ (∀x . x ∈ B ⇒ P(x))
    - (∃x : B . P(x)) ⇔ (∃x . x ∈ B ∧ P(x))
  + Where P(x) is the characteristic predicate for the set B
* Z notation – {term ⋅ signature | predicate}
  + Term – a term in predicate logic
    - Can be omitted if it’s just a variable
  + Signature – list of variables used in the term and their types
    - Can be omitted – but need one of term or signature
  + Predicate – any wff in predicate logic w/ the variables in term used as free variables
    - Can be omitted – just put “true”
  + E.g. people who live in Waterloo
    - {p : Person | lives\_in(p, Waterloo)}
  + E.g. ages of students who attend UW
    - {age(s) . s : Student | attends(s, UW)}
* Set comprehension:
  + x ∈ {y | P(y)} ⇔ P(x)
    - Same information can be expressed as set membership or a unary predicate
  + **Axioms**:
    - x ∈ {y : S | P(y)} ⇔ x ∈ S ∧ P(x)
    - x ∈ {f(y) . y : S | P(y)} ⇔ ∃y . y ∈ S ∧ P(y) ∧ (x = f(y))
* Empty set = ∅
  + **Axiom**: ∀x . ¬(x ∈ ∅)
* Universal set = U
  + All objects of concern; e.g. a type
* Set equality – two sets have exactly the same elements
  + **Axiom**: D = B ⇔ (∀x . x ∈ D ⇔ x ∈ B)
* Subsets – all elements of D are also elements of B
  + **Axiom**: D ⊆ B ⇔ (∀x . x ∈ D ⇒ x ∈ B)
  + **Axiom** – proper subset: D ⊂ B ⇔ D ⊆ B ∧ ¬(D = B)
* Singleton set – contains only one element
  + i.e. |D| = 1
* Power set (of a set) – set of all its subsets
  + **Axiom**: P(D) = {B | B ⊆ D}
  + |P(D))| = 2^|D| for finite sets
  + S : P(D) ⇔ S ⊆ D
* Derived laws:
  + |= ∅ ⊆ B – empty set is a subset of every set
  + |= B ⊆ B – every set is a subset of itself
  + |= D ⊆ B ∧ B ⊆ C ⇒ D ⊆ C – subset is transitive
  + |= D = B ⇔ D ⊆ B ∧ B ⊆ D – set equality
  + |= S ∈ P(Q) ⇔ (∀x . x ∈ S ⇒ x ∈ Q) – power set
* **Set operations**
* Set functions
  + **Axiom** – set union: D ∪ B = {x | x ∈ D ∨ x ∈ B}
    - Union of multiple sets: ∪J, where J = a set of sets
      * E.g. ∪({S} ∪ T) = S ∪ (∪T), S = a set, T = a set of sets
  + **Axiom** – set intersection: D ∩ B = {x | x ∈ D ∧ x ∈ B}
    - Intersection of multiple sets: ∩J, where J = set of sets
      * E.g. ∩({S} ∩ T) = S ∩ (∩T)
  + |D ∪ B| = |D| + |B| – |D ∩ B|
  + D and B are disjoint if D ∩ B = ∅
  + **Axiom** – absolute complement: D’ = {x | x ∈ U ∧ ¬(x ∈ D)}
  + **Axiom** – set difference (relative complement): D – B = {x | x ∈ D ∧ ¬(x ∈ B)}
* Derived laws:
  + Commutativity, associativity, distributivity, De Morgan’s
  + Empty/universal set identities
  + Intersection is subset – D ∩ B ⊆ D
  + Subset of union – D ⊆ D ∪ B
* **Set theory – transformational proof**
* Type 1: to prove ⇔, use ↔
* Type 2: to prove D = B
  + 2.1: prove x ∈ D ↔ x ∈ B
  + 2.2: prove = directly; replace terms directly, as in ND
* Use ND when proving P |− Q
* Russell’s Paradox
  + If a set can contain sets as elements, can it contain itself?
  + Consider S = {X | ¬(X ∈ X)} – contains sets that do not contain themselves
  + S ∈ S ∨ ¬(S ∈ S) by lem

case S ∈ S {

S ∈ {X | X ∈ X)} by defn of S

¬(S ∈ S) by set comprehension

false by not\_e

}

case ¬(S ∈ S) {

¬(S ∈ {X | X ∈ X}) by defn of S

¬(¬(S ∈ S)) by set comprehension

false by not\_e

}

false by cases

* + Thus a set cannot contain itself
  + Russell’s hierarchy – sets can only contain sets at a lower level
    - Level 1 – sets of individual elements; e.g. 1, 2, …
    - Level 2 – sets of sets of individual elements; e.g. {1, 2}, {2, 3}, …
    - Level 3 – e.g. {{1, 2}, {2, 3}}, {{3, 4}, {4, 5}}, …
  + There is no set that contains every set
  + Thus S = {X | X ∉ X} cannot exist
* **Set relations**
* Tuple – contains two or more (ordered) objects
* Cartesian product
  + C × B = {(c, b) | c ∈ C ∧ b ∈ B}
  + E.g. People = {R, J, S}, Cars = {Honda, BMW}
  + People × Cars = {(R, Honda), (R, BMW), (J, Honda), (J, BMW), (S, Honda), (S, BMW)}
* Relations
  + A relation is a subset of the Cartesian product of two or more sets
    - Binary relation – a set of pairs
  + R : C ↔ B or R : P(C × B)
  + Similar to set membership vs. unary predicates, relations and multiple-arity predicates express the same information
    - E.g. north\_of(Toronto, Waterloo) ⇔ (Toronto, Waterloo) ∈ North\_Of
  + E.g. own : People ↔ Cars, fix : People ↔ Cars
    - Own = {(R, Honda), (J, Honda), (S, BMW)}
    - Fix = {(R, Honda), (J, BMW)}
  + The relation from people to cars in which the people fix cars they own
    - Own ∩ fix
  + The relation from people to cars in which the people fix cars they don’t own
    - Own’ ∩ fix or fix – own
  + Everyone who fixes a model of car also owns that model
    - Fix ⊆ own
* Domain & range
  + Domain = all of the first elements in the pairs of the relation
  + Range = all of the second elements in the pairs of the relation
  + For R : A ↔ B:
    - **Axiom**: dom R = {x : A | ∃y : B . (x, y) ∈ R}
    - **Axiom**: ran R = {y : B| ∃x : A . (x, y) ∈ R}
  + E.g. people who own cars
    - Dom(own)
  + E.g. models of cars of people who both own and fix the same model of car
    - Ran(own ∩ fix)
  + E.g. every mechanic fixes at least one kind of car that he or she doesn’t own
    - Dom(fix) ⊆ dom(fix – own)
* Inverse relation – reverses order of pairs
  + **Axiom**: R~ or R-1 = {(b, a) . a : A, b : B | (a, b) ∈ R} for R : A ↔ B
* Identity relation – pairs every element of a set with itself
  + **Axiom**: id B = {(a, a) | a ∈ B}
* Relational composition
  + Consider sets A, B, C, relations R : A ↔ B, S : B ↔ C
  + **Axiom**: R; S = {(a, c) . a : A, c : C | ∃b . (a, b) ∈ R ∧ (b, c) ∈ S}
    - “R followed by S”
  + i.e. pairs (a ∈ dom R, c ∈ ran S) such that they are “connected” by b ∈ ran R and b ∈ dom S
  + E.g. colour: car ↔ colour
    - Colour = {(Honda, tan), (BMW, blue), (Honda, green)}
    - Own; colour = {(R, tan), (T, green), (J, tan), (J, green), (S, blue)}
* Iteration
  + **Axiom**: for set D and relation R : D ↔ D
    - R0 = id D
    - Rn = R; Rn – 1
* Relational image
  + **Axiom**: for sets D, B, S such that S ⊆ D and relation R : D ↔ B
    - R(|S|) = {y : B | ∃x : D . (x, y) ∈ R ∧ x ∈ S}
  + i.e. subset of ran R such that its corresponding domain element x ∈ S
  + i.e. R(|S|) = ran(S <| R)
* Restrictions
  + For R : D ↔ B and S ⊆ D
  + Domain restriction – contains pairs whose 1st element is in S
    - **Axiom**: S <| R = {(a, b) . a : D, b : B | (a, b) ∈ R ∧ a ∈ S}
    - E.g. {R} <| own = {(R, Honda)}
  + Domain subtraction – contains pairs whose 1st element is not in S
    - **Axiom**: S <-| R = {(a, b) . a : D, b : B | (a, b) ∈ R ∧ ¬(a ∈ S)}
    - E.g. {R} <-| own = {(J, Honda), (S, BMW)}
  + Range restriction – contains pairs whose 2nd element is in S
    - **Axiom**: S |> R = {(a, b) . a : D, b : B | (a, b) ∈ R ∧ b ∈ S}
    - E.g. own |> {Honda} = {(R, Honda), (J, Honda)}
  + Range subtraction – contains pairs whose 2nd element is not in S
    - **Axiom**: S |-> R = {(a, b) . a : D, b : B | (a, b) ∈ R ∧ ¬(b ∈ S)}
    - E.g. own |-> {Honda} = {(S, BMW)}
* Relational override
  + For R : D ↔ B and S : D ↔ B
  + “Update” R with S where pairs in R with the same 1st element of pairs in S are replaced by those pairs in S
  + **Axiom:** R ⊕ S = ((dom S) <-| R ) ∪ S
    - = {(x, y) | (x, y) ∈ R ∧ x ∉ (dom S) ∨ (x, y) ∈ S}
  + i.e. remove pairs in R that have the same 1st element as pairs in S, then add all pairs in S
  + E.g. own ⊕ {(R, Ford)} = {(R, Ford), (J, Honda), (S, BMW)}
* When providing an interpretation:
  + Domain – X = {a}, Y = {b, c}
  + Set – syntax = D → meaning = {a, b}
  + Relation – syntax = R → meaning = {(a, b), (a, c)} (i.e. R : X ↔ Y)
  + E.g. [R; R~]

= {(a, b), (a, c)}; {(a, b), (a, c)}~

= {(a, b), (a, c)}; {(b, a), (c, a)}

= {(a, a)}

* Ex:
  + Owns: People ↔ Dwellings
  + Rents: People ↔ Dwellings
  + Students: P(People)
  + Houses: P(Dwellings)
  + All houses that are rented are owned
    - (ran(rents)) ∩ houses ⊆ ran(owns)
  + Not every student owns a house
    - ¬(Students ⊆ dom(owns |> Houses))
  + Students who rent houses do not own any dwellings
    - (dom(Students <| (rents |> Houses)) ∩ dom(owns)) = ∅