0. Sets

Sets are a simple yet surprisingly important mathematical notion. In fact, there was a time when it was believed that all of mathematics could be formalised using the Theory of Sets (Set Theory). Even though that dream was never realised (we shall see later why), sets have proven themselves extraordinarily useful in computer science.

We shall not worry too much about defining sets here; let's assume that you already have a good idea about sets. Instead we shall confine ourselves to reminding ourselves about certain basic concepts and properties about sets, and later how to specify and then implement them in an algorithmic framework. In the following, we do not assume a universe from which elements are drawn — those restrictions will come later.

0.1 **Set Membership and Equality**

The primary notion that we shall assume is the idea of an element x being a member of a set S, which we shall write as $x \in S$. If an element y is not a member of a set S, we write $y \notin S$. A commonly used notation for sets uses *braces* to delimit the set, and *commas* to separate the members. For example, $\{0,1,2\}$ is a set.

The first notion we encounter about sets is that two sets are equal if they have the same members. More precisely:

Definition 0.1 (Set Equality) Two sets S_1, S_2 are equal, written $S_1 = S_2$, exactly when $x \in S_1$ *if and only if* $x \in S_2$ for any element x.

From the mathematical definition of set equality, one can show the following.

Exercise 0.0 (*Reflexivity of set equality*). Show that $S = S$, for any set S.

Exercise 0.1 (*Symmetry of set equality*). Show that if $S_1 = S_2$ then $S_2 = S_1$ for any sets S_1, S_2.

> **Proof**. Suppose $S_1 = S_2$. So (by 0.1), $x \in S_1$ *if and only if* $x \in S_2$.
> Therefore, $x \in S_2$ *if and only if* $x \in S_1$. That is, $S_2 = S_1$. (By definition 0.1) □

Exercise 0.2 (*Transitivity of set equality*). Show that if $S_1 = S_2$ and $S_2 = S_3$, then $S_1 = S_3$ for any sets S_1, S_2, S_3.

> [We will soon find that the properties of reflexivity, symmetry and transitivity apply in many contexts; together these properties characterise a notion of *equivalence*. When an equivalence is maintained within every construction, it is called a *congruence*.]

This notion of "*extensionality*" implies that members of a set are not listed multiple times (there is a different notion, called a *multiset*, where the number of times an element appears does indeed matter) and that the order in which the members are listed is also not important (as is the case in a *sequence* or *list*).

0.2 **The Empty Set**

We shall assert that *sets exist*. If we were to ask some to prove the existence of sets, they could (in a constructive sense) prove this by producing a set as a witness. What is the simplest set to produce? The *empty set!* The empty set, written as $\{\}$, has no elements as members. That is, $x \notin \{\}$ for any element x. Since the number of elements in the empty set is zero, we will sometimes playfully also write $\mathbf{0}$ to denote the empty set.

0.3 Specifying sets and membership

Given the importance of sets in most mathematics, most programming languages (oddly enough) do not directly support sets. Many languages support lists, where multiple occurrences and the order of elements does matter.

Let us start by specifying sets (and programming) in a declarative language, PROLOG. This language is based on first-order logic. We will learn more about PROLOG later. The focus in this section will be on specifying membership properties of sets, and constructions over sets. You can get started on using PROLOG by visiting the URL `http://tau-prolog.org` in a browser.

We will *represent* sets using PROLOG lists, with the empty list, written `[]`, representing the empty set. (Note that this will only allow us to represent finite sets.) The variable `X` denotes any element (note the capital/upper case letter; a word beginning with a capital letter in PROLOG indicates it is a logical *variable*).

Note however that we are only ***specifying*** sets in terms of their ***abstract*** properties related to membership, not ***implementing*** them, as we would in a Data Structures course.

The binary predicate `mem(X,S)` encodes $x \in S$.

```
/* membership of X in S */

/* no element is a member of the empty set */
mem(X,[]) :- fail.

/* X is a member of a set containing X, of course! */
mem(X,[X|_]) :- !.
/* if X is not the first chosen element of S,
then it may be a later element */
mem(X,[_|R]) :- mem(X,R).
```

Checking if any element (denoted by the logical variable `X`) is a member of the empty set results in failure (which in PROLOG represents the boolean notion of "false"). And if `X` appears in the list encoding the set S, at the first position, then (of course, denoted by !) $x \in S$; otherwise `X` may appear later in the list, which can be checked recursively by `mem(X,R)`, where `R` denotes the tail of the list representing set S. The symbol `:-` should be read as "if", and the last line of code above can be read as "(if `X` isn't the first element of the list representing set S, then) $x \in S$ **if** `X` appears in the tail (represented by variable `R`) of the list."

0.4 Subsets

The next important notion is that of subsets.

Definition 0.2 (Subset). A set S_1 is a subset of another set S_2, written $S_1 \subseteq S_2$, if for any element x, whenever $x \in S_1$ then $x \in S_2$.

[Note that we will be liberal in considering a set to be a subset of itself. When we need to be pedantic, and want to exclude this case, we will use the terminology "proper subset"]

Using just this specification of the notion of subset in terms of membership, one can show the following:

Exercise 0.3 (Reflexivity of subset). Show that $S \subseteq S$, for any set S.
Exercise 0.4 (Transitivity of subset). Show that if $S_1 \subseteq S_2$ and $S_2 \subseteq S_3$, then $S_1 \subseteq S_3$ for any sets S_1, S_2, S_3.
Exercise 0.5 (Antisymmetry of subset). Show that if $S_1 \subseteq S_2$ and $S_2 \subseteq S_1$, then $S_1 = S_2$ for any sets S_1, S_2.
Proof: Let $S_1 \subseteq S_2$**.** Therefore if $x \in S_1$ then $x \in S_2$.
Now if also $S_2 \subseteq S_1$, then if $x \in S_2$ then $x \in S_1$. Thus $x \in S_1$ if and only if $x \in S_2$.
So $S_1 = S_2$.

[We will soon find that the properties of reflexivity and transitivity apply in many contexts; together these properties characterise a notion of *pre-ordering* or a *quasi-ordering*. When antisymmetry also applies, the notion is called a *partial ordering*. When a partial ordering is preserved under every construction, it is called a *pre-congruence*.]

Definition 0.3 (Least and greatest elements in a partially ordered set). Let $(S, \sqsubseteq)$ be any set with a partially ordering. An element $x \in S$ is called a *least* element of S if for any element $y \in S$, $x \sqsubseteq y$. Dually, an element $z \in S$ is called a *greatest* element of S if for any element $y \in S$, $y \sqsubseteq z$.

[Note that in a partially ordered set, least and greatest elements are unique (due to antisymmetry). In a quasi-ordered set, however, we may have multiple minimal or maximal elements.]

A specification of the subset relationship $S_1 \subseteq S_2$ is easy to program recursively, based the predicate `mem(X,R)`.

```
/* subset */
subset([],L)  :- !.
subset([X|R],L)  :- mem(X,L), subset(R,L).
```

The empty set $\mathbf{0}$ is (of course!) a subset of any set S_2 (represented here by list `L`). The set S_1 with (first) element `X` is a subset of set S_2 (represented by `L`) if `X` is a member of set S_2 **and** if the rest of set S_1 (represented by list R) is a subset of set S_2. (The conjunction **and** is represented by a comma between `mem(X,L)` and the recursive call `subset(R,L)`.

```
/* equal sets */
eqset(L1, L2) :- subset(L1,L2), subset(L2,L1).
```

Antisymmetry allows us to define a set equality predicate using `subset(L1,L2)`.

0.5 Union and Intersection of sets

Definition 0.4 (Union) The union of two sets S_1, S_2, written $S_1 \cup S_2$, is defined as the least set S (under the subset ordering) such that $x \in S$ **if** ($x \in S_1$ **or** $x \in S_2$ (or both)). In "set builder notation", $S_1 \cup S_2 = \{x \;\; x \in S_1 \vee x \in S_2\}$.

```
/* Union */
mem(X,union(S1,S2)) :- mem(X,S1).
mem(X,union(S1,S2)) :- mem(X,S2).
```

Union can be represented as a construction, denoted `union(S1,S2)`, and can be defined using the disjunction of two *clauses*, the first checking if $x \in S_1$ and the second checking if $x \in S_2$.

[Note that the construction `union(S1,S2)` is just that — a "construction" that is as yet uninterpreted, and used only to say that we can perform such an operation on sets. So if you try, as an enterprising student in this class did to check if `eqset(union([1,2],[3]), [1,2,3]),` you will be disappointed to find that the answer is not what mathematics will have you expect. That is because the only thing we have specified is membership of the union construction on two sets. This construction does <u>not</u> simplify automatically to the set theoretic (mathematical) union of the two sets. In other words, `union(_,_)` is "only syntax"! We will wait for an implementation of union of sets (represented as lists) later. Hopefully, the implementation will satisfy the membership properties that the specification says it should]

The next interesting concept is set theory is that of intersection of two sets.
Definition 0.5 (<u>Intersection</u>) The intersection of two sets S_1, S_2, written $S_1 \cap S_2$, is defined as the <u>greatest</u> set S less than S_1, S_2 under the subset ordering such that $x \in S$ **if** ($x \in S_1$ **and** $x \in S_2$). In "set builder notation",
$S_1 \cap S_2 = \{x \ \ x \in S_1 \wedge x \in S_2\}$.

```
/* Intersection */
mem(X,inter(S1,S2)) :- mem(X,S1),mem(X,S2).
```

We can represent intersection as a construction, denoted `inter(S1,S2)`, where there is a single clause that takes a conjunction of the membership checks.

From the <u>mathematical specifications</u> of empty set, union and intersection, prove the following (from first principles).

Exercise 0.6 (Subset of union): Show that $S_1 \subseteq S_1 \cup S_2$ and $S_2 \subseteq S_1 \cup S_2$.
Exercise 0.7 (Intersection-subset): Show that $S_1 \cap S_2 \subseteq S_1$ and $S_1 \cap S_2 \subseteq S_2$.

It is only slightly harder to show the following.
Exercise 0.8 (Subset-union): Show that $S_1 \subseteq S_2$ if and only if $S_1 \cup S_2 = S_2$.
Exercise 0.9 (Subset-intersection): Show that $S_1 \subseteq S_2$ if and only if $S_1 \cap S_2 = S_1$.

Exercise 0.10 (<u>Commutativity</u>): Show that $S_1 \cup S_2 = S_2 \cup S_1$.
Exercise 0.11 (<u>Associativity</u>): Show that $(S_1 \cup S_2) \cup S_3 = S_1 \cup (S_2 \cup S_3)$.
Exercise 0.12 (<u>Idempotence</u>**):** $S \cup S = S$.
Exercise 0.13 (<u>Identity</u>): Show that $S \cup \mathbf{0} = S = \mathbf{0} \cup S$.

Union and Intersection exhibit a nice duality — see the relationship between the statements in Exercises 0.10-1.12 with those in 0.14-0.16; 0.18-0.19 with 0.20-0.21; and 0.22 with 0.23.

Exercise 0.14 (<u>Commutativity</u>): Show that $S_1 \cap S_2 = S_2 \cap S_1$.
Exercise 0.15 (<u>Associativity</u>): Show that $(S_1 \cap S_2) \cap S_3 = S_1 \cap (S_2 \cap S_3)$.
Exercise 0.16 (<u>Idempotence</u>): $S \cap S = S$.
Exercise 0.17 (<u>Annihilation</u>): Show that $S \cap \mathbf{0} = \mathbf{0} = \mathbf{0} \cap S$.

In fact, union and intersection exhibit (nice) dual distributive properties as well:

Exercise 0.18 (<u>Left Distributivity</u> of union over intersection): Show that
$S_1 \cup (S_2 \cap S_3) = (S_1 \cup S_2) \cap (S_1 \cup S_3)$.
Exercise 0.19 (<u>Right Distributivity</u> of union over intersection): Show that
$(S_1 \cap S_2) \cup S_3 = (S_1 \cup S_3) \cap (S_2 \cup S_3)$.

Exercise 0.20 (Left Distributivity of intersection over union): Show that $S_1 \cap (S_2 \cup S_3) = (S_1 \cap S_2) \cup (S_1 \cap S_3)$.
Exercise 0.21 (Right Distributivity of intersection over union): Show that $(S_1 \cup S_2) \cap S_3 = (S_1 \cap S_3) \cup (S_2 \cap S_3)$.

Exercise 0.22 (Absorption intersection-union): Show that $S_1 \cap (S_1 \cup S_2) = S_1$.
Exercise 0.23 (Absorption union-intersection): Show that $S_1 \cup (S_1 \cap S_2) = S_1$.

0.6 **Programs as objects about which we can reason**.

Programming is not coding. Programming is the representation of *abstractions*, usually mathematical abstractions, in terms of the constructs of a programming language, such that we can reason about the correctness of the representation.

Programming Activity 0.0: Design tests for checking that the programs written above indeed behave the way they are expect to. You may wish to check that membership behaves the way it should on empty and non-empty sets, where a particular element is and is not a member. You should check that membership works irrespective of order of elements in a list representation of a set, and irrespective of whether duplicates are present in the list representation or not.

For each of the exercises, you should design concrete tests and check that membership (and non-membership) behave as indicated by the property.

0.7 **Set Difference and Complement**

Definition 0.6 (Set Difference) The difference of two sets S_1, S_2, written $S_1 - S_2$, is defined as the greatest set S such that $x \in S$ **if** ($x \in S_1$ **and** $x \notin S_2$). In "set builder notation", $S_1 - S_2 = \{x \ \ x \in S_1 \wedge x \notin S_2\}$.

```
/* Set Difference */
mem(X, diff(S1, S2)) :- mem(X, S1),
                        \+ mem(X,S2).
```

We can represent set difference as a construction, denoted `diff(S1,S2)`, where there is a single clause that checks the membership in `S1` and non-membership in `S2`. The PROLOG operator `\+` denotes negation of the predicate that follows.

Definition 0.7 (Complement) The complement of a set S, written $\overline{S}$, is defined as the set S comprising all elements not in S, that is $x \in \overline{S}$ **if** $x \notin S$. In "set builder notation", $\overline{S} = \{x \ \ x \notin S\}$.

```
/* Complement */
mem(X, compl(S)) :- \+ mem(X, S).
```

Note that complementation of a set is a special case of set difference. Assume that we have a universal set $\mathcal{U}$. Then $\overline{S} = \mathcal{U} - S$. Alternatively, one can think of set difference as a "relative complement".

The notion of a universal set allows us to state the *identity* for intersection, dual to Exercise 0.13:
Exercise 0.24 (Identity of intersection): Show that $S \cap \mathcal{U} = S = \mathcal{U} \cap S$.

as well as the *annihilator* for union, dual to Exercise 0.17
Exercise 0.25 (Annihilation for union): Show that $S \cup \mathscr{U} = \mathscr{U} = \mathscr{U} \cup S$.

Further, once we have a complementation operation, we can complete the story for the boolean algebraic laws, by presenting the so-called De Morgan Laws:
Exercise 0.26 (Complement of union): Show that $\overline{S_1 \cup S_2} = \overline{S_1} \cap \overline{S_2}$.
Exercise 0.27 (Complement of intersection): Show that $\overline{S_1 \cap S_2} = \overline{S_1} \cup \overline{S_2}$.

Exercise 0.28 (Complements: empty/universal set): Show that $\mathbf{0} = \overline{\mathscr{U}}$ and $\overline{\mathbf{0}} = \mathscr{U}$.
Exercise 0.29 (Involution of complement): Show that $\overline{\overline{S}} = S$.
Exercise 0.30 (Union with complement): Show that $S \cup \overline{S} = \mathscr{U}$.
Exercise 0.30 (Intersection with complement): Show that $S \cap \overline{S} = \mathbf{0}$.

Exercise 0.31 (Difference-Complement-Union): Show that $S_1 - S_2 = S_1 \cap \overline{S_2}$.

Definition 0.8 (Symmetric Difference) The symmetric difference of two sets S_1, S_2, written $S_1 \ominus S_2$, is defined as the set S such that $x \in S$ **if** $((x \in S_1$ **and** $x \notin S_2)$ **or** $(x \in S_2$ **and** $x \notin S_1))$.
Exercise 0.32: Show that $S_1 \ominus S_2 = (S_1 - S_2) \cup (S_2 - S_1)$.

0.8 Powerset.

Definition 0.9 (Powerset of S) The *powerset* of a given set S, written $\mathscr{P}(S)$ or more punnily as $\mathbf{2}^S$, is defined as the *set of all subsets* of S. In "set builder notation", $\mathscr{P}(S) = \{S' \ \ S' \subseteq S\}$.
Proposition 0.0: The powerset $\mathscr{P}(S)$ of a given set S forms a partially ordered set under the subset ordering. We will write this as $(\mathscr{P}(S), \subseteq)$.
Exercise 0.33: Prove Proposition 0.0.
Notation: The powerset of the empty set $\mathbf{0}$ is a *singleton* set with a single element, namely the empty set. That is $\mathbf{2}^{\mathbf{0}} = \{\mathbf{0}\}$. So it makes sense to call this set $\mathbf{1}$, and we can punnily write the equation $\mathbf{2}^{\mathbf{0}} = \mathbf{1}$.

Since the empty set is a subset of any set S, it can be considered the least subset of any given set S, and hence the least element in $(\mathscr{P}(S), \subseteq)$ Likewise, since all subsets of set S are contained in it, S is the greatest element in $(\mathscr{P}(S), \subseteq)$.

```
/* Powerset */
mem(X, power(S)) :- subset(X,S).
```

Every subset of S is a member of the powerset $\mathscr{P}(S)$.

0.9 Cartesian Product

Definition 0.10 (Cartesian Product) The (Cartesian) *product* of two sets S_1, S_2, written $S_1 \times S_2$, is defined as the set S such that the *ordered pair* $(x, y) \in S$ **if** $(x \in S_1$ **and** $y \in S_2)$. In "set builder notation", $S_1 \times S_2 = \{(x, y) \ \ x \in S_1 \wedge y \in S_2\}$.

```
/* Cartesian product */
mem((X,Y), cartesian(S1,S2)) :- mem(X,S1),
                                mem(Y,S2).
```

Cartesian product is a canonical construction on sets. Here we specify it using the constructor `cartesian(S1,S2)` where membership is defined in terms of membership of the respective elements of the ordered pair in the corresponding sets.

What is the Cartesian product of a set S with the empty set $\mathbf{0}$? You should be able to see that since (for any y) $y \notin \mathbf{0}$, $S \times \mathbf{0} = \mathbf{0}$, and since (for any x) $x \notin \mathbf{0}$, $\mathbf{0} \times S = \mathbf{0}$. Which is a nice pun!

And what is the Cartesian product of a set S with the singleton set $\mathbf{1}$? You should be able to see that $S \times \mathbf{1} = \{(x, \mathbf{0}) \ \ x \in S\}$, and $\mathbf{1} \times S = \{(\mathbf{0}, y) \ \ y \in S\}$. While these sets are not technically equal to the set S, we shall see (after we introduce the concept of isomorphism $\cong$ between sets) that $S \times \mathbf{1} \cong S \cong \mathbf{1} \times S$. Another nice pun!

0.10 Binary Relations

Definition 0.11 (Binary Relation) A binary relation R between two sets S_1, S_2 is *any* subset of $S_1 \times S_2$. That is, $R \subseteq S_1 \times S_2$.
Note that the empty set $\mathbf{0}$ and $S_1 \times S_2$ are both binary relations. Since the Cartesian product of sets is itself a set, binary relations can be *partially ordered* using the usual notion of subset — in fact, the collection of binary relations over sets S_1, S_2 form a partially ordered set under $\subseteq$, with $\mathbf{0}$ and $S_1 \times S_2$ being the least and greatest elements respectively.

Definition 0.12 (Total Relation) A relation $R \subseteq S_1 \times S_2$ is called *total* if for *each* $x \in S_1$ there exists a $y \in S_2$ such that $(x, y) \in R$. [Otherwise a relation is called *partial*.]

Definition 0.13 (Onto Relation) A relation $R \subseteq S_1 \times S_2$ is called *onto* (*surjective*, *epi*-) if for *each* $y \in S_2$ there exists a $x \in S_1$ such that $(x, y) \in R$.

Definition 0.14 (1-1 Relation) A relation $R \subseteq S_1 \times S_2$ is called *1-1* (*injective*, *mono*-) if whenever $(x_1, y) \in R$ and $(x_2, y) \in R$ then $x_1 = x_2$. That is, any $y \in S_2$ is the "target" under $R \subseteq S_1 \times S_2$ of at most one $x \in S_1$

Definition 0.15 (Functional Relation) A relation $R \subseteq S_1 \times S_2$ is called *functional* (*a function*) if whenever $(x, y_1) \in R$ and $(x, y_2) \in R$ then $y_1 = y_2$. That is, any $x \in S_1$ is related under $R \subseteq S_1 \times S_2$ to at most one $y \in S_2$

When S_1, S_2 are the same, i.e., $R \subseteq S \times S$, then we say "the relation R is over set S."

Definition 0.16 (Identity Relation) Given any set S, the identity relation on S, written $id(S)$, is the binary relation comprising *all* pairs (x, x) for $x \in S$. In "set builder notation", $id(S) = \{(x, x) \ \ x \in S\}$.

```
/* Identity Relation on a set S */
mem((X,X), id(S)) :- mem(X,S), !.
```

The constructor `id(S)` is used to denote the identity relation over set S.

Definition 0.17 (Relational Composition) Given any two binary relations $R_1 \subseteq S_1 \times S_2$, and $R_2 \subseteq S_2 \times S_3$, their relational composition, written

$R_1 \circ R_2 \subseteq S_1 \times S_3$, is the binary relation comprising *all* pairs $(x,z) \in S_1 \times S_3$ such that for some $y \in S_2 : (x,y) \in R_1$ and $(y,z) \in R_2$. In "set builder notation", $R_1 \circ R_2 = \{(x,z) \in S_1 \times S_3 \ \ \exists y \in S_2 : (x,y) \in R_1 \wedge (y,z) \in R_2\}$.

```
/* Relational Composition */
mem((X,Z), comp(R1, R2)) :- mem((X,Y),R1),
                            mem((Y,Z),R2).
```

The constructor `comp(R1, R2)` is used to specify relational composition.

Proposition 0.1: For any binary relation $R \subseteq S_1 \times S_2$, the relations $id(S_1)$ and $id(S_2)$ are the left- and right- identities of relational composition. That is, $id(S_1) \circ R = R = R \circ id(S_2)$.
Exercise 0.34: Prove Proposition 0.1.

Proposition 0.2: Relational composition is <u>associative</u>. That is, for any binary relations $R_1 \subseteq S_1 \times S_2$, $R_2 \subseteq S_2 \times S_3$ and $R_3 \subseteq S_3 \times S_4$: $(R_1 \circ R_2) \circ R_3 = R_1 \circ (R_2 \circ R_3)$
Exercise 0.35: Prove Proposition 0.2.

Proposition 0.3: Relational composition is <u>monotone</u>. That is, for any binary relations $R_1, R_1' \subseteq S_1 \times S_2$, and $R_2, R_2' \subseteq S_2 \times S_3$ such that $R_1 \subseteq R_1'$ and $R_2 \subseteq R_2'$. Then $R_1 \circ R_2 \subseteq R_1' \circ R_2'$.
Proof. Let $(x,y) \in R_1$ and $(y,z) \in R_2$. Then $(x,z) \in R_1 \circ R_2$. But from the definition of subset, $(x,y) \in R_1'$ and $(y,z) \in R_2'$. Therefore $(x,z) \in R_1' \circ R_2'$. $\square$

Definition 0.18 (Relational Inverse) For any binary relation $R \subseteq S_1 \times S_2$, its *<u>inverse</u>*, written $R^{-} \subseteq S_2 \times S_1$, is the binary relation comprising *all* pairs (y,x) for $(x,y) \in R$. In "set builder notation", $R^{-} = \{(y,x) \in S_2 \times S_1 \ \ (x,y) \in R\}$.

```
/* Inverse Relation */
mem( (Y,X), inv(R)) :- mem( (X,Y), R).
```

The constructor `inv(R)` is used to denote the inverse of the relation `R`.

Proposition 0.4: Relational inverse leaves the identity relations unchanged. For all sets S: $(id(S))^{-} = id(S)$
Exercise 0.36: Prove Proposition 0.4.

Proposition 0.5: Relational inverse is involutive. For all relations $R \subseteq S_1 \times S_2$: $(R^{-})^{-} = R$.
Exercise 0.37: Prove Proposition 0.5.

Proposition 0.6: . For all relations $R_1 \subseteq S_1 \times S_2$ and $R_2 \subseteq S_2 \times S_3$: $(R_1 \circ R_2)^{-} = R_2^{-} \circ R_1^{-}$
Exercise 0.38: Prove Proposition 0.6.

0.11 Properties of Binary Relations over a set

Definition 0.19 (<u>Reflexivity</u>) A binary relation $R \subseteq S \times S$, is *<u>reflexive</u>* if for *all* $x \in S$, $(x,x) \in R$.
Proposition 0.7: (i) The identity relation $id(S)$ is reflexive.
(ii) The relation $S \times S$ is reflexive.
Exercise 0.39: Prove Proposition 0.7.

Definition 0.20 (Symmetry) A binary relation $R \subseteq S \times S$, is *symmetric* if whenever $(x, y) \in R$, then $(y, x) \in R$.
Proposition 0.8: (i) The identity relation $id(S)$ is symmetric.
(ii) The relation $S \times S$ is symmetric.
Exercise 0.40: Prove Proposition 0.8.

Definition 0.21 (Transitivity) A binary relation $R \subseteq S \times S$ is *transitive* if whenever $(x, y) \in R$ and $(y, z) \in R$, then $(x, z) \in R$.
Proposition 0.9: (i) The identity relation $id(S)$ is transitive.
(ii) The relation $S \times S$ is transitive.
Exercise 0.41: Prove Proposition 0.9.

Proposition 0.10: A relation $R \subseteq S \times S$ is transitive if and only if $R \circ R \subseteq R$
Exercise 0.41: Prove Proposition 0.10.

0.12 Closure Operations on Binary Relations over a set.

Given a relation $R \subseteq S \times S$, a *closure* is a "constructed" relation R' that contains R, i,e., $R \subseteq R'$, such R' also has certain properties.

Definition 0.22 (Reflexive Closure) For any binary relation $R \subseteq S \times S$, its *reflexive closure*, written $R^{=} \subseteq S \times S$, is the *least* *reflexive* relation containing R. That is, (i) $R \subseteq R^{=}$, (ii) $R^{=}$ is reflexive, and (iii) for any other $R' \subseteq S \times S$ such that $R \subseteq R'$ and R' is reflexive, $R^{=} \subseteq R'$.

Proposition 0.11: $R^{=} = R \cup id(S)$.

```
/* Reflexive Closure */
mem( (X,X), refclos(R,S)) :- mem(X,S), !.
mem( (X,Y), refclos(R,S)) :- mem((X,Y), R), !.
```

We use Proposition 0.11 to specify membership of the reflexive closure of `R`, denoted using the construction `refclos(R,S)`.

Exercise 0.42: Prove Proposition 0.11.

Definition 0.23 (Symmetric Closure) For any binary relation $R \subseteq S \times S$, its *symmetric closure*, written $R^{\leftrightarrow} \subseteq S \times S$, is the *least* *symmetric* relation containing R. That is, (i) $R \subseteq R^{\leftrightarrow}$, (ii) $R^{\leftrightarrow}$ is symmetric, and (iii) for any other $R' \subseteq S \times S$ such that $R \subseteq R'$ and R' is symmetric, $R^{\leftrightarrow} \subseteq R'$.

Proposition 0.12: $R^{\leftrightarrow} = R \cup R^{-}$

```
/* Symmetric Closure */
mem((X,Y), symclos(R)) :- mem((X,Y), R), !.
mem((X,Y), symclos(R)) :- mem((X,Y), inv(R)).
```

We use Proposition 0.12 to specify membership of the symmetric closure of `R`, denoted using the construction `symclos(R)`.

Exercise 0.43: Prove Proposition 0.12.

Definition 0.24 (Transitive Closure) For any binary relation $R \subseteq S \times S$, its *transitive closure*, written $R^+ \subseteq S \times S$, is the *least transitive* relation containing R. That is, (i) $R \subseteq R^+$, (ii) R^+ is transitive, and (iii) for any other $R' \subseteq S \times S$ such that $R \subseteq R'$ and R' is transitive, $R^+ \subseteq R'$.

Definition 0.25 (i-fold composition) The i-fold composition of a binary relation is defined (inductively) as follows:

$R^1 = R$

$R^{1+i} = R \circ R^i$ for $i \geq 1$.

[Note that we could have alternatively have defined $R^{i+1} = R^i \circ R$ for $i \geq 1$.]

Proposition 0.13 $R^{j+k} = R^j \circ R^k$

Proof: By induction on $j \geq 1$.

Base Case ($j = 1$). $R^{1+k} = R \circ R^k = R^1 \circ R^k$

Induction Hypothesis (IH): Suppose we have shown for $j = n$ that $R^{n+k} = R^n \circ R^k$

Induction Step: $R^{(1+n)+k} = R^{1+(n+k)} = R \circ R^{n+k}$ (Assoc of + & Definition 0.25)

$= R \circ (R^n \circ R^k)$ (by IH)

$= (R \circ R^n) \circ R^k$ (by associativity of composition)

$= R^{1+n} \circ R^k = R^j \circ R^k$

Proposition 0.14 $R^+ = \bigcup_{i \geq 1} R^i$

Proof. We first show that $\bigcup_{i \geq 1} R^i$ is a transitive relation containing R

(i) Since $R^1 = R,$ from the definition of (generalised) union, $R \subseteq \bigcup_{i \geq 1} R^i$

(ii) Let $(x, y) \in \bigcup_{i \geq 1} R^i$ and $(y, z) \in \bigcup_{i \geq 1} R^i$. So, from the definition of union, $(x, y) \in R^j$ and $(y, z) \in R^k$ for some $j, k \geq 1$. Therefore $(x, z) \in R^{j+k}$ (Proposition 0.13). But since $R^{j+k} \subseteq \bigcup_{i \geq 1} R^i$ by the definition of generalised union (for $j + k = i$), $(x, z) \in \bigcup_{i \geq 1} R^i$. So $\bigcup_{i \geq 1} R^i$ is a transitive relation.

(iii) Let $R' \subseteq S \times S$ be such that $R \subseteq R'$ and R' is transitive. (So by Proposition 0.10, $R' \circ R' \subseteq R'$)

We show by induction that for each $i \geq 1, R^i \subseteq R'$.

Base case ($i = 1$). By assumption: $R^1 = R \subseteq R'$.

Induction hypothesis: Suppose we have shown that for $i = k$ that $R^i \subseteq R'$.

Induction step: Consider $R^{1+k} = R \circ R^k$.

But $R \subseteq R'$ (by assumption) and $R^k \subseteq R'$ (by IH).

So by Proposition 0.3 and 0.10, $R^{1+k} = R \circ R^k \subseteq R' \circ R' \subseteq R'$.

Hence by induction, for each $i \geq 1, R^i \subseteq R'$.

Now by the definition of generalised union, since for each $i \geq 1$: $R^i \subseteq R'$, $\bigcup_{i \geq 1} R^i \subseteq R'$. □

```
/* Transitive Closure */
mem((X,Y), transclos(R)) :- mem((X,Y), R), !.
```

```
    mem((X,Z), transclos(R)) :- mem((X,Y), R),
                                     mem((Y,Z), transclos(R)), !.
```

Proposition 0.14 allows us to provide an inductive specification for the transitive closure of a relation, denoted by construction `transclos(R)`.

The closure operations can be combined.

Definition 0.26 (Reflexive Symmetric Closure) For any binary relation $R \subseteq S \times S$, its *reflexive-symmetric closure*, written $R^{\Leftrightarrow} \subseteq S \times S$, is the *least reflexive* and *symmetric* relation containing R. That is, (i) $R \subseteq R^{\Leftrightarrow}$, (ii) $R^{\Leftrightarrow}$ is reflexive, (iii) $R^{\Leftrightarrow}$ is symmetric, and (iv) for any other $R' \subseteq S \times S$ such that $R \subseteq R'$ and R' is reflexive and symmetric, $R^{\Leftrightarrow} \subseteq R'$.
Proposition 0.15: $R^{\Leftrightarrow} = id(S) \cup R \cup R^{-}$
Exercise 0.44: Prove Proposition 0.15.

Definition 0.27 (Reflexive Transitive Closure) For any binary relation $R \subseteq S \times S$, its *reflexive-transitive closure*, written $R^{*} \subseteq S \times S$, is the *least reflexive* and *transitive* relation containing R. That is, (i) $R \subseteq R^{*}$, (ii) R^{*} is reflexive, (iii) R^{*} is transitive, and (iv) for any other $R' \subseteq S \times S$ such that $R \subseteq R'$ and R' is reflexive and transitive, $R^{*} \subseteq R'$.

Let us posit $R^{0} = id(S)$
Proposition 0.16: $R^{*} = \bigcup_{i \geq 0} R^{i}$
Exercise 0.44: Prove Proposition 0.16.

Definition 0.28 (Reflexive Symmetric Transitive Closure) For any binary relation $R \subseteq S \times S$, its *reflexive-symmetric-transitive closure*, written $R^{\equiv} \subseteq S \times S$, is the *least reflexive, symmetric* and *transitive* relation containing R. That is,
(i) $R \subseteq R^{\equiv}$, (ii) $R^{\equiv}$ is reflexive, (ii) $R^{\equiv}$ is symmetric, (iv) $R^{\equiv}$ is transitive, and (v) for any other $R' \subseteq S \times S$ such that $R \subseteq R'$ and R' is reflexive, symmetric and transitive, $R^{\equiv} \subseteq R'$.

Proposition 0.17: $R^{\equiv} = (\bigcup_{i \geq 0} (R^{\Leftrightarrow})^{i})$, which is the *least equivalence* relation containing R.
Exercise 0.45: Prove Proposition 0.17.