CHAPTER 6

Relational Logic

6.1 Introduction

Propositional Logic allows us to talk about relationships among individual propositions, and it gives us the machinery to derive logical conclusions based on these relationships. Suppose, for example, we believe that, if Jack knows Jill, then Jill knows Jack. Suppose we also believe that Jack knows Jill. From these two facts, we can conclude that Jill knows Jack using a simple application of Implication Elimination.

Unfortunately, when we want to say things more generally, we find that Propositional Logic is inadequate. Suppose, for example, that we wanted to say that, in general, if one person knows a second person, then the second person knows the first. Suppose, as before, that we believe that Jack knows Jill. How do we express the general fact in a way that allows us to conclude that Jill knows Jack? Here, Propositional Logic is inadequate; it gives us no way of succinctly encoding this more general belief in a form that captures its full meaning and allows us to derive such conclusions.

Relational Logic is an alternative to Propositional Logic that solves this problem. The trick is to augment our language with two new linguistic features, viz. *variables* and *quantifiers*. With these new features, we can express information about multiple objects without enumerating those objects; and we can express the existence of objects that satisfy specified conditions without saying which objects they are.

In this chapter, we proceed through the same stages as in the introduction to Propositional Logic. We start with syntax and semantics. We then discuss evaluation and satisfaction. We look at some examples. Then, we talk about properties of Relational Logic sentences and logical entailment for Relational Logic. Finally, we say a few words about the equivalence of Relational Logic and Propositional Logic and its decidability.

6.2 Syntax

In Propositional Logic, sentences are constructed from a basic vocabulary of propositional constants. In Relational Logic, there are no propositional constants; instead we have *object constants*, *relation constants*, and *variables*.

In our examples here, we write both variables and constants as strings of letters, digits, and a few non-alphanumeric characters (e.g. "_"). By convention, variables begin with letters from the end of the alphabet (viz. *u*, *v*, *w*, *x*, *y*, *z*). Examples include *x*, *ya*, and *z_2*. By convention, all constants begin with either alphabetic letters (other than *u*, *v*, *w*, *x*, *y*, *z*) or digits. Examples include *a*, *b*, 123, *comp*225, and *barack_obama*.

Note that there is no distinction in spelling between object constants and relation constants. The type of each such word is determined by its usage or, in some cases, in an explicit specification.

As we shall see, relation constants are used in forming complex expressions by combining them with an appropriate number of arguments. Accordingly, each relation constant has an associated *arity*, i.e. the number of arguments with which that relation constant can be combined. A relation constant that can combined with a single argument is said to be *unary*; one that can be combined with two arguments is said to be *binary*; one that can be combined with three arguments is said to be *ternary*; more generally, a relation constant that can be combined with n arguments is said to be n-ary.

A *vocabulary* consists of a set of object constants, a set of relation constants, and an assignment of arities for each of the relation constants in the vocabulary. (Note that this definition here is slightly non-traditional. In many textbooks, a vocabulary (sometimes called a *signature*) includes a specification of relation constants but not object constants, whereas our definition here includes both types of constants.)

A *term* is defined to be a variable or an object constant. Terms typically denote objects presumed or hypothesized to exist in the world; and, as such, they are analogous to noun phrases in natural language, e.g. *Joe* or *someone*.

There are three types of *sentences* in Relational Logic, viz. relational sentences (the analog of propositions in Propositional Logic), logical sentences (analogous to the logical sentences in Propositional Logic), and quantified sentences (which have no analog in Propositional Logic).

A *relational sentence* is an expression formed from an n-ary relation constant and n terms. For example, if q is a relation constant with arity 2 and if a and y are terms, then the expression shown below is a syntactically legal relational sentence. Relational sentences are sometimes called *atoms* to distinguish them from logical and quantified sentences.

$$q(a, y)$$

Logical sentences are defined as in Propositional Logic. There are negations, conjunctions, disjunctions, implications, and equivalences. See below for examples.

Negation:	$(\neg p(a))$
Conjunction:	$(p(a) \wedge q(b, c))$
Disjunction:	$(p(a) \vee q(b, c))$
Implication:	$(p(a) \Rightarrow q(b, c))$
Biconditional:	$(p(a) \Leftrightarrow q(b, c))$

Note that the syntax here is exactly the same as in Propositional Logic except that the elementary components are relational sentences rather than proposition constants.

Quantified sentences are formed from a *quantifier*, a variable, and an embedded sentence. The embedded sentence is called the *scope* of the quantifier. There are two types of quantified sentences in Relational Logic, viz. universally quantified sentences and existentially quantified sentences.

A *universally quantified sentence* is used to assert that all objects have a certain property. For example, the following expression is a universally quantified sentence asserting that, if p holds of an object, then q holds of that object and itself.

$$(\forall x.(p(x) \Rightarrow q(x,x)))$$

An *existentially quantified sentence* is used to assert that some object has a certain property. For example, the following expression is an existentially quantified sentence asserting that there is an object that satisfies p and, when paired with itself, satisfies q as well.

$$(\exists x.(p(x) \wedge q(x,x)))$$

Note that quantified sentences can be nested within other sentences. For example, in the first sentence below, we have quantified sentences inside of a disjunction. In the second sentence, we have a quantified sentence nested inside of another quantified sentence.

$$((\forall x.p(x)) \vee (\exists x.q(x,x)))$$
$$(\forall x.(\exists y.q(x,y)))$$

As with Propositional Logic, we can drop unneeded parentheses in Relational Logic, relying on precedence to disambiguate the structure of unparenthesized sentences. In Relational Logic, the precedence relations of the logical operators are the same as in Propositional Logic, and quantifiers have higher precedence than logical operators.

The following examples show how to parenthesize sentences with both quantifiers and logical operators. The sentences on the right are partially parenthesized versions of the sentences on the left. (To be fully parenthesized, we would need to add parentheses around each of the sentences as a whole.)

$$\forall x.p(x) \Rightarrow q(x) \qquad (\forall x.p(x)) \Rightarrow q(x)$$
$$\exists x.p(x) \wedge q(x) \qquad (\exists x.p(x)) \wedge q(x)$$

Notice that, in each of these examples, the quantifier does *not* apply to the second relational sentence, even though, in each case, that sentence contains an occurrence of the variable being quantified. If we want to apply the quantifier to a logical sentence, we must enclose that sentence in parentheses, as in the following examples.

$$\forall x.(p(x) \Rightarrow q(x))$$
$$\exists x.(p(x) \wedge q(x))$$

An expression in Relational Logic is *ground* if and only if it contains no variables. For example, the sentence $p(a)$ is ground, whereas the sentence $\forall x.p(x)$ is not.

An occurrence of a variable is *free* if and only if it is not in the scope of a quantifier of that variable. Otherwise, it is *bound*. For example, y is free and x is bound in the following sentence.

$$\exists x.q(x,y)$$

A sentence is *open* if and only if it has free variables. Otherwise, it is *closed*. For example, the first sentence below is open and the second is closed.

$$p(y) \Rightarrow \exists x.q(x,y)$$
$$\forall y.(p(y) \Rightarrow \exists x.q(x,y))$$

6.3 Semantics

The semantics of Relational Logic presented here is termed *Herbrand semantics*. It is named after the logician Herbrand, who developed some of its key concepts. As Herbrand is French, it should properly be pronounced "air-brahn". However, most people resort to the Anglicization of this, instead pronouncing it "her-brand". (One exception is Stanley Peters, who has been known at times to pronounce it "hare-brained".)

The *Herbrand base* for a vocabulary is the set of all ground relational sentences that can be formed from the constants of the language. Said another way, it is the set of all sentences of the form $r(t_1,...,t_n)$, where r is an n-ary relation constant and t_1, ... , t_n are object constants.

For a vocabulary with object constants a and b and relation constants p and q where p has arity 1 and q has arity 2, the Herbrand base is shown below.

$$\{p(a), p(b), q(a,a), q(a,b), q(b,a), q(b,b)\}$$

It is worthwhile to note that, for a given relation constant and a finite set of object constants, there is an upper bound on the number of ground relational sentences that can be formed using that relation constant. In particular, for a set of object constants of size b, there are b^n distinct n-tuples of object constants; and hence there are b^n ground relational sentences for each n-ary relation constant. Since the number of relation constants in a vocabulary is finite, this means that the Herbrand base is also finite.

A *truth assignment* for a Relational Logic language is a function that maps each ground relational sentence in the Herbrand base to a truth value. As in Propositional Logic, we use the digit 1 as a synonym for true and 0 as a synonym for false; and we refer to the value assigned to a ground relational sentence by writing the relational sentence with the name of the truth assignment as a superscript. For example, the truth assignment shown below is an example for the case of the language mentioned a few paragraphs above.

$$\begin{aligned} p(a) &\rightarrow 1 \\ p(b) &\rightarrow 0 \\ q(a,a) &\rightarrow 1 \\ q(a,b) &\rightarrow 0 \\ q(b,a) &\rightarrow 1 \\ q(b,b) &\rightarrow 0 \end{aligned}$$

As with Propositional Logic, once we have a truth assignment for the ground relational sentences of a language, the semantics of our operators prescribes a unique extension of that assignment to the complex sentences of the language.

The rules for logical sentences in Relational Logic are the same as those for logical sentences in Propositional Logic. A truth assignment satisfies a negation $\neg\phi$ if and only if it does not satisfy ϕ. A truth assignment satisfies a conjunction $(\phi_1 \wedge ... \wedge \phi_n)$ if and only if it satisfies every ϕ_i. A truth assignment satisfies a disjunction $(\phi_1 \vee ... \vee \phi_n)$ if and only if it satisfies at least one ϕ_i. A truth assignment satisfies an implication $(\phi \Rightarrow \psi)$ if and only if it does not satisfy ϕ or does satisfy ψ. A truth assignment satisfies an equivalence $(\phi \Leftrightarrow \psi)$ if and only if it satisfies both ϕ and ψ or it satisfies neither ϕ nor ψ.

In order to define satisfaction of quantified sentences, we need the notion of instances. An *instance* of an expression is an expression in which all free variables have been consistently replaced by ground terms. Consistent replacement here means that, if one occurrence of a variable is replaced by a ground term, then all occurrences of that variable are replaced by the same ground term.

A universally quantified sentence is true for a truth assignment if and only if *every* instance of the scope of the quantified sentence is true for that assignment. An existentially quantified sentence is true for a truth assignment if and only if *some* instance of the scope of the quantified sentence is true for that assignment.

A truth assignment *satisfies* a sentence with free variables if and only if it satisfies every instance of that sentence. A truth assignment *satisfies* a set of sentences if and only if it satisfies every sentence in the set.

6.4 Evaluation

Evaluation for Relational Logic is similar to evaluation for Propositional Logic. The only difference is that we need to deal with quantifiers. In order to evaluate a universally quantified sentence, we check that all instances of the scope are true. (We are in effect treating it as the conjunction of all those instances.) In order to evaluate an existentially quantified sentence, we check that at least one instance of the scope. (We are in effect treating it as the disjunction of those instances.)

Once again, assume we have a language with a unary relation constant p, a binary relation constant q, and object constants a and b; and consider our truth assignment from the last section.

What is the truth value of the sentence $\forall x.(p(x) \Rightarrow q(x,x))$ under this assignment? There are two instances of the scope of this sentence. See below.

$$p(a) \Rightarrow q(a,a)$$
$$p(b) \Rightarrow q(b,b)$$

We know that $p(a)$ is true and $q(a,a)$ is true, so the first instance is true. $q(b,b)$ is false, but so is $p(b)$ so the second instance is true as well.

$$(p(a) \Rightarrow q(a,a)) \rightarrow 1$$
$$(p(b) \Rightarrow q(b,b)) \rightarrow 1$$

Since both instances are true, the original quantified sentence is true as well.

$$\forall x.(p(x) \Rightarrow q(x,x)) \rightarrow 1$$

Now let's consider a case with nested quantifiers. Is $\forall x.\exists y.q(x,y)$ true or false for the truth assignment shown above? As before, we know that this sentence is true if every instance of its scope is true. The two possible instances are shown below.

$$\exists y.q(a,y)$$
$$\exists y.q(b,y)$$

To determine the truth of $\exists y.q(a,y)$, we must find at least one instance of $q(a,y)$ that is true. The possibilities are shown below.

$$q(a,a)$$
$$q(a,b)$$

Looking at our truth assignment, we see that the first of these is true and the second is false.

$$q(a,a) \rightarrow 1$$
$$q(a,b) \rightarrow 0$$

Since one of these instances is true, the existential sentence as a whole is true.

$$\exists y.q(a,y) \rightarrow 1$$

Now, we do the same for the second existentially quantified. The possible instances in this case are shown below.

$$q(b,a)$$
$$q(b,b)$$

Of these, the first is true, and the second is false.

$$q(b,a) \rightarrow 1$$
$$q(b,b) \rightarrow 0$$

Again, since one of these instances is true, the existential sentence as a whole is true.

$$\exists y.q(b,y) \rightarrow 1$$

At this point, we have truth values for our two existential sentences. Since both instances of the scope of our original universal sentence are true, the sentence as a whole must be true as well.

$$\forall x.\exists y.q(x,y) \rightarrow 1$$

6.5 Satisfaction

As in Propositional Logic, it is in principle possible to build a truth table for any set of sentences in Relational Logic. This truth table can then be used to determine which truth assignments satisfy a given set of sentences.

As an example, let us assume we have a language with just two object constants a and b and two unary relation constants p and q. Now consider the sentences shown below, and assume our job is to find a truth assignment that satisfies these sentences.

$$p(a) \vee p(b)$$
$$\forall x.(p(x) \Rightarrow q(x))$$
$$\exists x.q(x)$$

A truth table for this problem is shown below. Each of the four columns on the left represents one of the elements of the Herbrand base for this language. The three columns on the right represent our sentences.

$p(a)$	$p(b)$	$q(a)$	$q(b)$	$p(a) \vee p(b)$	$\forall x.(p(x) \Rightarrow q(x))$	$\exists x.q(x)$
1	1	1	1	1	1	1
1	1	1	0	1	0	1
1	1	0	1	1	0	1
1	1	0	0	1	0	0
1	0	1	1	1	1	1
1	0	1	0	1	1	1
1	0	0	1	1	0	1
1	0	0	0	1	0	0
0	1	1	1	1	1	1
0	1	1	0	1	0	1
0	1	0	1	1	1	1
0	1	0	0	1	0	0
0	0	1	1	0	1	1
0	0	1	0	0	1	1
0	0	0	1	0	1	1
0	0	0	0	0	1	0

Looking at the table, we see that there are twelve truth assignments that make the first sentence true, nine that make the second sentence true, twelve that make the third sentence true, and five that make them all true (rows 1, 5, 6, 9, and 11).

6.6 Example - Sorority World

Consider once again the Sorority World example introduced in Chapter 1. Recall that this world focusses on the interpersonal relations of a small sorority. There are just four members - Abby, Bess, Cody, and Dana. Our goal is to represent information about who likes whom.

In order to encode this information in Relational Logic, we adopt a vocabulary with four object constants (*abby*, *bess*, *cody*, *dana*) and one binary relation constant (*likes*).

If we had complete information about the likes and dislikes of the girls, we could completely characterize the state of affairs as a set of ground relational sentences or negations of ground relational sentences, like the ones shown below, with one sentence for each member of the Herbrand base. (In our example here, we have written the positive literals in black and the negative literals in grey in order to distinguish the two more easily.)

¬*likes*(*abby*,*abby*)	¬*likes*(*abby*,*bess*)	*likes*(*abby*,*cody*)	¬*likes*(*abby*,*dana*)
¬*likes*(*bess*,*abby*)	¬*likes*(*bess*,*bess*)	*likes*(*bess*,*cody*)	¬*likes*(*bess*,*dana*)
likes(*cody*,*abby*)	*likes*(*cody*,*bess*)	¬*likes*(*cody*,*cody*)	*likes*(*cody*,*dana*)
¬*likes*(*dana*,*abby*)	¬*likes*(*dana*,*bess*)	*likes*(*dana*,*cody*)	¬*likes*(*dana*,*dana*)

To make things more interesting, let's assume that we do *not* have complete information, only fragments of information about the girls' likes and dislikes. Let's see how we can encode such fragments in Relational Logic.

Let's start with a simple disjunction. *Bess likes Cody or Dana*. Encoding a sentence with a disjunctive noun phrase (such as *Cody or Dana*) is facilitated by first rewriting the sentence as a disjunction of simple sentences. *Bess likes Cody or Bess likes Dana*. In Relational Logic, we can express this fact as a simple disjunction with the two possibilities as disjuncts.

$$likes(bess,cody) \vee likes(bess,dana)$$

Abby likes everyone Bess likes. Again, paraphrasing helps translate. *If Bess likes a girl, then Abby also likes her.* Since this is a fact about everyone, we use a universal quantifier.

$$\forall y.(likes(bess,y) \Rightarrow likes(abby,y))$$

Cody likes everyone who likes her. In other words, *if some girl likes Cody, then Cody likes that girl*. Again, we use a universal quantifier.

$$\forall y.(likes(y,cody) \Rightarrow likes(cody,y))$$

Bess likes somebody who likes her. The word *somebody* here is a tip-off that we need to use an existential quantifier.

$$\exists y.(likes(bess,y) \wedge likes(y,bess))$$

Nobody likes herself. The use of the word *nobody* here suggests a negation. A good technique in such cases is to rewrite the English sentence as the negation of a positive version of the sentence before translating to Relational Logic.

$$\neg\exists x.likes(x,x)$$

Everybody likes somebody. Here we have a case requiring two quantifiers, one universal and one existential. The key to this case is getting the quantifiers in the right order. Reversing them leads to a very different statement.

$$\forall x.\exists y.likes(x,y)$$

There is someone everyone likes. The preceding sentence tells us that everyone likes someone, but that someone can be different for different people. This sentence tells us that everybody likes the same person.

$$\exists y.\forall x.likes(x,y)$$

6.7 Example - Blocks World

The Blocks World is a popular application area for illustrating ideas in the field of Artificial Intelligence. A typical Blocks World scene is shown in Figure 1.

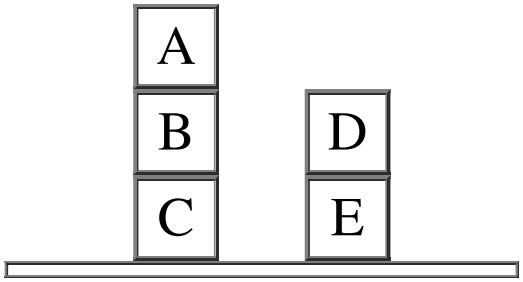

Figure 1 - One state of Blocks World.

Most people looking at this figure interpret it as a configuration of five toy blocks. Some people conceptualize the table on which the blocks are resting as an object as well; but, for simplicity, we ignore it here.

In order to describe this scene, we adopt a vocabulary with five object constants, as shown below, with one object constant for each of the five blocks in the scene. The intent here is for each of these object constants to represent the block marked with the corresponding capital letter in the scene.

$$\{a, b, c, d, e\}$$

In a spatial conceptualization of the Blocks World, there are numerous meaningful relations. For example, it makes sense to think about the relation that holds between two blocks if and only if one is resting on the other. In what follows, we use the relation constant *on* to refer to this relation. We might also think about the relation that holds between two blocks if and only if one is anywhere above the other, i.e. the first is resting on the second or is resting on a block that is resting on the second, and so forth. In what follows, we use the relation constant *above* to talk about this relation. There is the relation that holds of three blocks that are stacked one on top of the other. We use the relation constant *stack* as a name for this relation. We use the relation constant *clear* to denote the relation that holds of a block if and only if there is no block on top of it. We use the relation constant *table* to denote the relation that holds of a block if and only if that block is resting on the table. The set of relation constants corresponding to this conceptualization is shown below.

$$\{on, above, stack, clear, table\}$$

The arities of these relation constants are determined by their intended use. Since *on* is intended to denote a relation between two blocks, it has arity 2. Similarly, *above* has arity 2. The *stack* relation constant has arity 3. Relation constants *clear* and *table* each have arity 1.

Given this vocabulary, we can describe the scene in Figure 1 by writing ground literals that state which relations hold of which objects or groups of objects. Let's start with *on*. The following sentences tell us directly for each ground relational sentence whether it is true or false. (Once again, we have written the positive literals in black and the negative literals in grey in order to distinguish the two more easily.)

$\neg on(a,a)$	$on(a,b)$	$\neg on(a,c)$	$\neg on(a,d)$	$\neg on(a,e)$
$\neg on(b,a)$	$\neg on(b,b)$	$on(b,c)$	$\neg on(b,d)$	$\neg on(b,e)$
$\neg on(c,a)$	$\neg on(c,b)$	$\neg on(c,c)$	$\neg on(c,d)$	$\neg on(c,e)$
$\neg on(d,a)$	$\neg on(d,b)$	$\neg on(d,c)$	$\neg on(d,d)$	$on(d,e)$
$\neg on(e,a)$	$\neg on(e,b)$	$\neg on(e,c)$	$\neg on(e,d)$	$\neg on(e,e)$

We can do the same for the other relations. However, there is an easier way. Each of the remaining relations can be defined in terms of *on*. These definitions together with our facts about the *on* relation logically entail every other ground relational sentence or its negation. Hence, given these definitions, we do not need to write out any additional data.

A block satisfies the *clear* relation if and only if there is nothing on it.

$$\forall y.(clear(y) \Leftrightarrow \neg\exists x.on(x,y))$$

A block satisfies the *table* relation if and only if it is not on some block.

$$\forall x.(table(x) \Leftrightarrow \neg\exists y.on(x,y))$$

Three blocks satisfy the *stack* relation if and only if the first is on the second and the second is on the third.

$$\forall x.\forall y.\forall z.(stack(x,y,z) \Leftrightarrow on(x,y) \wedge on(y,z))$$

The *above* relation is a bit tricky to define correctly. One block is above another block if and only if the first block is on the second block or it is on another block that is above the second block. Also, no block can be above itself. Given a complete definition for the *on* relation, these two axioms determine a unique *above* relation.

$$\forall x.\forall z.(above(x,z) \Leftrightarrow on(x,z) \vee \exists y.(on(x,y) \wedge above(y,z)))$$

$$\neg\exists x.above(x,x)$$

One advantage to defining relations in terms of other relations is economy. If we record *on* information for every object and encode the relationship between the *on* relation and these other relations, there is no need to record any ground relational sentences for those relations.

Another advantage is that these general sentences apply to Blocks World scenes other than the one pictured here. It is possible to create a Blocks World scene in which none of the *on* sentences we have listed is true, but these general definitions are still correct.

6.8 Example - Modular Arithmetic

In this example, we show how to characterize Modular Arithmetic in Relational Logic. In Modular Arithmetic, there are only finitely many objects. For example, in Modular Arithmetic with modulus 4, we would have just four integers - 0, 1, 2, 3 - and that's all. Our goal here to define the addition relation. Admittedly, this is a modest goal; but, once we see how to do this; we can use the same approach to define other arithmetic relations.

Let's start with the *same* relation, which is true of every number and itself and is false for numbers that are different. We can completely characterize the *same* relation by writing ground relational sentences, one positive sentence for each number and itself and negative sentences for all of the other cases.

***same*(0,0)**	$\neg$*same*(0,1)	$\neg$*same*(0,2)	$\neg$*same*(0,3)
$\neg$*same*(1,0)	***same*(1,1)**	$\neg$*same*(1,2)	$\neg$*same*(1,3)
$\neg$*same*(2,0)	$\neg$*same*(2,1)	***same*(2,2)**	$\neg$*same*(2,3)
$\neg$*same*(3,0)	$\neg$*same*(3,1)	$\neg$*same*(3,2)	***same*(3,3)**

Now, let's axiomatize the *next* relation, which, for each number, gives the next larger number, wrapping back to 0 after we reach 3.

next(0,1)
next(1,2)
next(2,3)
next(3,0)

Properly, we should write out the negative literals as well. However, we can save that work by writing a single axiom asserting that *next* is a functional relation, i.e., for each member of the Herbrand base, there is just one successor.

$$\forall x.\forall y.\forall z.(next(x,y) \wedge next(x,z) \Rightarrow same(y,z))$$

In order to see why this saves us the work of writing out the negative literals, we can write this axiom in the equivalent form shown below.

$$\forall x.\forall y.\forall z.(next(x,y) \wedge \neg same(y,z) \Rightarrow \neg next(x,z))$$

The addition table for Modular Arithmetic is the usual addition table for arbitrary numbers except that we wrap around whenever we get past 3. For such a small arithmetic, it is easy to write out the ground relational sentences for addition, as shown below.

plus(0,0,0)	*plus*(1,0,1)	*plus*(2,0,2)	*plus*(3,0,3)
plus(0,1,1)	*plus*(1,1,2)	*plus*(2,1,3)	*plus*(3,1,0)
plus(0,2,2)	*plus*(1,2,3)	*plus*(2,2,0)	*plus*(3,2,1)
plus(0,3,3)	*plus*(1,3,0)	*plus*(2,3,1)	*plus*(3,3,2)

As with *next*, we avoid writing out the negative literals by writing a suitable functionality axiom for *plus*.

$$\forall x.\forall y.\forall z.\forall w.(plus(x,y,z) \wedge \neg same(z,w) \Rightarrow \neg plus(x,y,w))$$

That's one way to do things, but we can do better. Rather than writing out all of those relational sentences, we can use Relational Logic to define *plus* in terms of *same* and *next* and use that axiomatization to deduce the ground relational sentences. The definition is shown below. First, we have an identity axiom. Adding 0 to any number results in the same number. Second we have a successor axiom. If z is the sum of x and y, then the sum of the successor of x and y is the successor of z. Finally, we have our functionality axiom once again.

$\forall y.plus(0,y,y)$
$\forall x.\forall y.\forall z.\forall x2.\forall z2.(plus(x,y,z) \wedge next(x,x2) \wedge next(z,z2) \Rightarrow plus(x2,y,z2))$
$\forall x.\forall y.\forall z.\forall w.(plus(x,y,z) \wedge \neg same(z,w) \Rightarrow \neg plus(x,y,w))$

One advantage of doing things this way is economy. With these sentences, we do not need the ground relational sentences about *plus* given above. They are all logically entailed by our sentences about *next* and the definitional sentences. A second advantage is versatility. Our sentences define *plus* in terms of *next* for arithmetic with any modulus, not just modulus 4.

6.9 Logical Properties

As we have seen, some sentences are true in some truth assignments and false in others. However, this is not always the case. There are sentences that are always true and sentences that are always false as well as sentences that are sometimes true and sometimes false.

As with Propositional Logic, this leads to a partition of sentences into three disjoint categories. A sentence is *valid* if and only if it is satisfied by *every* truth assignment. A sentence is *unsatisfiable* if and only if it is not satisfied by any truth assignment. A sentence is *contingent* if and only if there is some truth assignment that satisfies it and some truth assignment that falsifies it.

Alternatively, we can classify sentences into two overlapping categories. A sentence is *satisfiable* if and only if it is satisfied by at least one truth assignment, i.e. it is either valid or contingent. A sentence is *falsifiable* if and only if there is at least one truth assignment that makes it false, i.e. it is either contingent or unsatisfiable.

Note that these definitions are the same as in Propositional Logic. Moreover, some of our results are the same as well. If we think of ground relational sentences as propositions, we get similar results for the two logics - a ground sentence in Relational Logic is valid / contingent / unsatisfiable if and only if the corresponding sentence in Propositional Logic is valid / contingent / unsatisfiable.

Here, for example, are Relational Logic versions of common Propositional Logic validities - the Law of the Excluded Middle, Double Negation, and deMorgan's laws for distributing negation over conjunction and disjunction.

$$p(a) \vee \neg p(a)$$
$$p(a) \Leftrightarrow \neg\neg p(a)$$
$$\neg(p(a) \wedge q(a,b)) \Leftrightarrow (\neg p(a) \vee \neg q(a,b))$$
$$\neg(p(a) \vee q(a,b)) \Leftrightarrow (\neg p(a) \wedge \neg q(a,b))$$

Of course, not all sentences in Relational Logic are ground. There are valid sentences of Relational Logic for which there are no corresponding sentences in Propositional Logic.

The *Common Quantifier Reversal* tells us that reversing quantifiers of the same type has no effect on truth assignment.

$$\forall x.\forall y.q(x,y) \Leftrightarrow \forall y.\forall x.q(x,y)$$
$$\exists x.\exists y.q(x,y) \Leftrightarrow \exists y.\exists x.q(x,y))$$

Existential Distribution tells us that it is okay to move an existential quantifier inside of a universal quantifier. (Note that the reverse is not valid, as we shall see later.)

$$\exists y.\forall x.q(x,y) \Rightarrow \forall x.\exists y.q(x,y)$$

Finally, *Negation Distribution* tells us that it is okay to distribute negation over quantifiers of either type by flipping the quantifier and negating the scope of the quantified sentence.

$$\neg\forall x.p(x) \Leftrightarrow \exists x.\neg p(x)$$
$$\neg\exists x.p(x) \Leftrightarrow \forall x.\neg p(x)$$

6.10 Logical Entailment

A set of Relational Logic sentences Δ *logically entails* a sentence ϕ (written $\Delta \models \phi$) if and only if every truth assignment that satisfies Δ also satisfies ϕ.

As with validity and contingency and satisfiability, this definition is the same for Relational Logic as for Propositional Logic. As before, if we treat ground relational sentences as propositions, we get similar results. In particular, a set of ground premises in Relational Logic logically entails a ground conclusion in Relational Logic if and only if the corresponding set of Propositional Logic premises logically entails the corresponding Propositional Logic conclusion.

For sentences without variables, we have the following results. The sentence $p(a)$ logically entails $(p(a) \vee p(b))$. The sentence $p(a)$ does *not* logically entail $(p(a) \wedge p(b))$. However, any set of sentences containing both $p(a)$ and $p(b)$ does logically entail $(p(a) \wedge p(b))$.

The presence of variables allows for additional logical entailments. For example, the premise $\exists y.\forall x.q(x,y)$ logically entails the conclusion $\forall x.\exists y.q(x,y)$. If there is *some* object y that is paired with every x, then every x has some object that it pairs with, viz. y.

$$\exists y.\forall x.q(x,y) \models \forall x.\exists y.q(x,y)$$

Here is another example. The premise $\forall x.\forall y.q(x,y)$ logically entails the conclusion $\forall x.\forall y.q(y,x)$. The first sentence says that q is true for all pairs of objects, and the second sentence says the exact same thing. In cases like this, we can interchange variables.

$$\forall x.\forall y.q(x,y) \models \forall x.\forall y.q(y,x)$$

Understanding logical entailment for Relational Logic is complicated by the fact that it is possible to have free variables in Relational Logic sentences. Consider, for example, the premise $q(x,y)$ and the conclusion $q(y,x)$. Does $q(x,y)$ logically entail $q(y,x)$ or not?

Our definition for logical entailment and the semantics of Relational Logic give a clear answer to this question. Logical entailment holds if and only if every truth assignment that satisfies the premise satisfies the conclusion. A truth assignment satisfies a sentence with free variables if and only if it satisfies every instance. In other words, a sentence with free variables is equivalent to the sentence in which all of the free variables are universally quantified. In other words, $q(x,y)$ is satisfied if and only if $\forall x.\forall y.q(x,y)$ is satisfied, and similarly for $q(y,x)$. So, the first sentence here logically entails the second if and only if $\forall x.\forall y.q(x,y)$ logically entails $\forall x.\forall y.q(y,x)$; and, as we just saw, this is, in fact, the case.

6.11 Relational Logic and Propositional Logic

One interesting feature of *Relational Logic* (RL) is that it is expressively equivalent to Propositional Logic (PL). For any RL language, we can produce a pairing between the ground relational sentences in that language and the proposition constants in a PL language. Given this correspondence, for any set of *arbitrary* sentences in our RL language, there is a corresponding set of sentences in the language of PL such that any RL truth assignment that satisfies our RL sentences agrees with the corresponding PL truth assignment applied to the PL sentences.

The procedure for transforming our RL sentences to PL has multiple steps, but each step is easy. We first convert our sentences to prenex form, then we ground the result, and we rewrite in PL. Let's look at these steps in turn.

A sentence is in *prenex form* if and only if it is closed and all of the quantifiers are on the outside of all logical operators.

Converting a set of RL sentences to a logically equivalent set in prenex form is simple. First, we rename variables in different quantified sentences to eliminate any duplicates. We then apply quantifier distribution rules in reverse to move quantifiers outside of logical operators. Finally, we universally quantify any free variables in our sentences.

For example, to convert the sentence $\forall y.p(x,y) \vee \exists y.q(y)$ to prenex form, we first rename one of the variables. In this case, let's rename the y in the second disjunct to z. This results in the sentence $\forall y.p(x,y) \vee \exists z.q(z)$. We then apply the distribution rules in reverse to produce $\forall y.\exists z.(p(x,y) \vee q(z))$. Finally, we universally quantify the free variable x to produce the prenex form of our original sentence, viz. $\forall x.\forall y.\exists z.(p(x,y) \vee q(z))$

Once we have a set of sentences in prenex form, we compute the grounding. We start with our initial set Δ of sentences and we incrementally build up our grounding Γ. On each step we process a sentence in Δ, using the procedure described below. The procedure terminates when Δ becomes empty. The set Γ at that point is the grounding of the input.

(1) The first rule covers the case when the sentence ϕ being processed is ground. In this case, we remove the sentence from Delta and add it to Gamma.

$$\Delta_{i+1} = \Delta_i - \{\phi\}$$
$$\Gamma_{i+1} = \Gamma_i \cup \{\phi\}$$

(2) If our sentence is of the form $\forall \nu.\phi[\nu]$, we eliminate the sentence from Δ_i and replace it with copies of the scope, one copy for each object constant τ in our language.

$$\Delta_{i+1} = \Delta_i - \{\forall \nu.\phi[\nu]\} \cup \{\phi[\tau] \mid \tau \text{ an object constant}\}$$
$$\Gamma_{i+1} = \Gamma_i$$

(3) If our sentence of the form $\exists \nu.\phi[\nu]$, we eliminate the sentence from Δ_i and replace it with a disjunction, where each disjunct is a copy of the scope in which the quantified variable is replaced by an object constant in our language.

$$\Delta_{i+1} = \Delta_i - \{\exists \nu.\phi[\nu]\} \cup \{\phi[\tau_1] \vee \ldots \vee \phi[\tau_n]\}$$
$$\Gamma_{i+1} = \Gamma_i$$

The procedure halts when Δ_i becomes empty. The set Γ_i is the grounding of the input. It is easy to see that Γ_i is logically equivalent to the input set.

Here is an example. Suppose we have a language with just two object constants a and b. And suppose we have the set of sentences shown below. We have one ground sentence, one universally quantified sentence, and one existentially quantified sentence. All are in prenex form.

$$\{p(a), \forall x.(p(x) \Rightarrow q(x)), \exists x.\neg q(x)\}$$

A trace of the procedure is shown below. The first sentence is ground, so we remove it from Δ add it to Γ. The second sentence is universally quantified, so we replace it with a copy for each of our two object constants. The resulting sentences are ground, and so we move them one by one from Δ to Γ. Finally, we ground the existential sentence and add the result to Δ and then move the ground sentence to Γ. At this point, since Δ is empty, Γ is our grounding.

$$\Delta_0 = \{\{p(a), \forall x.(p(x) \Rightarrow q(x)), \exists x.\neg q(x)\}$$
$$\Gamma_0 = \{\}$$

$$\Delta_1 = \{\forall x.(p(x) \Rightarrow q(x)), \exists x.\neg q(x)\}$$
$$\Gamma_1 = \{p(a)\}$$

$$\Delta_2 = \{p(a) \Rightarrow q(a), p(b) \Rightarrow q(b), \exists x.\neg q(x)\}$$
$$\Gamma_2 = \{p(a)\}$$

$$\Delta_3 = \{p(b) \Rightarrow q(b), \exists x.\neg q(x)\}$$
$$\Gamma_3 = \{p(a), p(a) \Rightarrow q(a)\}$$

$$\Delta_4 = \{\exists x.\neg q(x)\}$$
$$\Gamma_4 = \{p(a), p(a) \Rightarrow q(a), p(b) \Rightarrow q(b)\}$$

$$\Delta_5 = \{\neg q(a) \vee \neg q(b)\}$$
$$\Gamma_5 = \{p(a), p(a) \Rightarrow q(a), p(b) \Rightarrow q(b)\}$$

$$\Delta_6 = \{\}$$
$$\Gamma_6 = \{p(a), p(a) \Rightarrow q(a), p(b) \Rightarrow q(b), \neg q(a) \vee \neg q(b)\}$$

Once we have a grounding Γ, we replace each ground relational sentence in Γ by a proposition constant. The resulting sentences are all in Propositional Logic; and the set is equivalent to the sentences in Δ in that any RL truth assignment that satisfies our RL sentences agrees with the corresponding Propositional Logic truth assignment applied to the Propositional Logic sentences.

For example, let's represent the RL sentence $p(a)$ with the proposition pa; let's represent $p(b)$ with pb; let's represent $q(a)$ with qa; and let's represent $q(b)$ with qb. With this correspondence, we can represent the sentences in our grounding with the Propositional Logic sentences shown below.

$$\{pa, pa \Rightarrow qa, pb \Rightarrow qb, \neg qa \vee \neg qb\}$$

Since the question of unsatisfiability for PL is decidable, then the question of unsatisfiability for RL is also decidable. Since logical entailment and unsatisfiability are correlated, we also know that the question of logical entailment for RL is decidable.

Another consequence of this correspondence between RL and PL is that, like PL, RL is *compact* - every unsatisfiable set of sentences contains a finite subset that is unsatisfiable. This is important as it assures us that we can demonstrate the unsatisfiability by analyzing just a finite set of sentences; and, as we shall see in the next chapter, logical entailment can be demonstrated with finite proofs.

Recap

Relational Logic is an alternative to Propositional Logic that includes some linguistic features, viz. constants and variables and quantifiers. In Relational Logic, simple sentences have more structure than in Propositional Logic. Furthermore, using variables and quantifiers, we can express information about multiple objects without enumerating those objects; and we can express the existence of objects that satisfy specified conditions without saying which objects they are. The syntax of Relational Logic begins with object constants and relation constants. *Relational*

sentences are the atomic elements from which more complex sentences are built. *Logical sentences* are formed by combining simpler sentences with logical operators. In the version of Relational Logic used here, there are five types of logical sentences - negations, conjunctions, disjunctions, implications, and equivalences. There are two types of *quantified sentences*, viz. *universal sentences* and *existential sentences*. The *Herbrand base* for a Relational Logic language is the set of all ground relational sentences in the language. A *truth assignment* for a Relational Logic language is a mapping that assigns a truth value to each element of it Herbrand base. The truth or falsity of compound sentences is determined from a truth assignment using rules based on the five logical operators of the language. A truth assignment *satisfies* a sentence if and only if the sentences is *true* under that truth assignment. A sentence is *valid* if and only if it is satisfied by *every* truth assignment. A sentence is *satisfiable* if and only if it is satisfied by at least one truth assignment. A sentence is *falsifiable* if and only if there is at least one truth assignment that makes it false. A sentence is *unsatisfiable* if and only if it is not satisfied by any truth assignment. A sentence is *contingent* if and only if it is both satisfiable and falsifiable, i.e. it is neither valid nor unsatisfiable. A set of sentences Δ *logically entails* a sentence ϕ (written $\Delta \models \phi$) if and only if every truth assignment that satisfies Δ also satisfies ϕ.

Exercises

Exercise 6.1: Say whether each of the following expressions is a syntactically legal sentence of Relational Logic. Assume that *jim* and *molly* are object constants; assume that *person* is a unary relation constant; and assume that *parent* is a binary relation constant.

(*a*) $parent(jim, molly)$
(*b*) $parent(molly, molly)$
(*c*) $\neg person(jim)$
(*d*) $person(jim, molly)$
(*e*) $parent(molly, z)$
(*f*) $\exists x.parent(molly, x)$
(*g*) $\exists y.parent(molly, jim)$
(*h*) $\forall z.(z(jim, molly) \Rightarrow z(molly, jim))$

Exercise 6.2: Consider a language with n object constants and a single binary relation constant.

(*a*) How many ground terms are there in this language - $n, n^2, 2^n, 2^{n^2}, 2^{2^n}$?

(*b*) How many ground atomic sentences are there in this language - $n, n^2, 2^n, 2^{n^2}, 2^{2^n}$?

(*c*) How many distinct truth assignments are possible for this language - $n, n^2, 2^n, 2^{n^2}, 2^{2^n}$?

Exercise 6.3: Consider a language with object constants a and b and relation constants p and q where p has arity 1 and q has arity 2. Imagine a truth assignment that makes $p(a)$, $q(a,b)$, $q(b,a)$ true and all other ground atoms false. Say whether each of the following sentences is true or false under this truth assignment.

(*a*) $\forall x.(p(x) \Rightarrow q(x,x))$
(*b*) $\forall x.\exists y.q(x,y)$
(*c*) $\exists y.\forall x.q(x,y)$
(*d*) $\forall x.(p(x) \Rightarrow \exists y.q(x,y))$
(*e*) $\forall x.p(x) \Rightarrow \exists y.q(y,y)$

Exercise 6.4: Consider a state of the Sorority World that satisfies the following sentences.

$\neg likes(abby,abby)$	$likes(abby,bess)$	$\neg likes(abby,cody)$	$likes(abby,dana)$
$likes(bess,abby)$	$\neg likes(bess,bess)$	$likes(bess,cody)$	$\neg likes(bess,dana)$
$\neg likes(cody,abby)$	$likes(cody,bess)$	$\neg likes(cody,cody)$	$likes(cody,dana)$
$likes(dana,abby)$	$\neg likes(dana,bess)$	$likes(dana,cody)$	$\neg likes(dana,dana)$

Say which of the following sentences is satisfied by this state of the world.

(*a*) $likes(dana,cody)$
(*b*) $\neg likes(abby,dana)$
(*c*) $likes(bess,cody) \vee likes(bess,dana)$
(*d*) $\forall y.(likes(bess,y) => likes(abby,y))$
(*e*) $\forall y.(likes(y,cody) \Rightarrow likes(cody,y))$
(*f*) $\forall x.\neg likes(x,x)$

Exercise 6.5: Consider a version of the Blocks World with just three blocks - *a*, *b*, and *c*. The *on* relation is axiomatized below.

$\neg on(a,a)$	$on(a,b)$	$\neg on(a,c)$
$\neg on(b,a)$	$\neg on(b,b)$	$on(b,c)$
$\neg on(c,a)$	$\neg on(c,b)$	$\neg on(c,c)$

Let's suppose that the *above* relation is defined as follows.

$$\forall x.\forall z.(above(x,z) \Leftrightarrow on(x,z) \vee \exists y.(above(x,y) \wedge above(y,z)))$$

A sentence ϕ is consistent with a set Δ of sentences if and only if there is a truth assignment that satisfies all of the sentences in $\Delta \cup \{\phi\}$. Say whether each of the following sentences is consistent with the sentences about *on* and *above* shown above.

(*a*) $above(a,c)$
(*b*) $above(a,a)$
(*c*) $above(c,a)$

Exercise 6.6: Say whether each of the following sentences is valid, contingent, or unsatisfiable.

(*a*) $\forall x.p(x) \Rightarrow \exists x.p(x)$
(*b*) $\exists x.p(x) \Rightarrow \forall x.p(x)$
(*c*) $\forall x.p(x) \Rightarrow p(x)$
(*d*) $\exists x.p(x) \Rightarrow p(x)$
(*e*) $p(x) \Rightarrow \forall x.p(x)$
(*f*) $p(x) \Rightarrow \exists x.p(x)$
(*g*) $\forall x.\exists y.p(x,y) \Rightarrow \exists y.\forall x.p(x,y)$
(*h*) $\forall x.(p(x) \Rightarrow q(x)) \Rightarrow \exists x.(p(x) \wedge q(x))$
(*i*) $\forall x.(p(x) \Rightarrow q(x)) \wedge \exists x.(p(x) \wedge \neg q(x))$
(*j*) $(\exists x.p(x) \Rightarrow \forall x.q(x)) \vee (\forall x.q(x) \Rightarrow \exists x.r(x))$

Exercise 6.7: Let Γ be a set of Relational Logic sentences, and let ϕ and ψ be individual Relational Logic sentences. For each of the following claims, state whether it is true or false.

(*a*) $\forall x.\phi \vDash \phi$

(*b*) $\phi \vDash \forall x.\phi$

(*c*) If $\Gamma \vDash \neg\phi[\tau]$ for some ground term τ, then $\Gamma \nvDash \forall x.\phi[x]$

(*d*) If $\Gamma \vDash \phi[\tau]$ for some ground term τ, then $\Gamma \vDash \exists x.\phi[x]$

(*e*) If $\Gamma \vDash \phi[\tau]$ for every ground term τ, then $\Gamma \vDash \forall x.\phi[x]$