3.3.2 Logical conditions

To show many uses of binary variables we will take as an example a set of projects that we may or may not decide to do. We shall call the projects rather unimaginatively A, B, C, D, E, F, G and H and with each of these projects we will associate a decision variable (a binary variable) which is 1 if we decide to do the project and 0 if we decide not to do the project. We call the corresponding variables *a*, *b*, *c*, *d*, *e*, *f*, *g* and *h*. So decision variable *a* taking on the value 1 means that we do project A, whilst *a* taking on the value of 0 means that we do not do project A. We are now going to express some constraints in words and see how they can be modeled using these binary variables.

3.3.2.1 Choice among several possibilities

The first constraint that we might impose is 'we must choose no more than one project to do'. We can easily see that this can be expressed by the constraint

$$a + b + c + d + e + f + g + h \leq 1$$

Why is this true? Think of selecting one project, for instance project C. If *c* is 1 then the constraint immediately says that *a*, *b*, *d*, *e*, *f*, *g* and *h* must be 0 because there is no other setting for the values of these variables that will satisfy that constraint. So if *c* is 1 then all the others are 0 and, as there is nothing special about C, we can see immediately that the constraint means that we can only do one project.

If the constraint was that we could do no more than three projects then obviously all we have to do in the constraint above is to replace the 1 by 3 and then we can easily see that in fact we can have no more than three of the 0-1 variables being equal to 1. It is possible to have just two of them or one of them or even none of them being equal to 1 but we can certainly not have four or more being 1 and satisfy the constraint at the same time.

Now suppose that the constraint is that we must choose exactly two of the projects. Then the constraint we can use to model this is

$$a + b + c + d + e + f + g + h = 2$$

Since the binary variables can take only the values 0 or 1, we must have exactly two of them being 1 and all the rest of them being 0, and that is the only way that we can satisfy that constraint.

3.3.2.2 Simple implications

Now consider an entirely different sort of constraint. Suppose that 'if we do project A then we must do project B'. How do we model this? Like a lot of mathematics, it is a question of learning a trick that has been discovered. Consider the following constraint

$$b \geq a$$

To show that this formulation is correct, we consider all the possible combinations of settings (there are four of them) of *a* and *b*. First, what happens if we do not do project A. If we also do not do project B then the constraint is satisfied ($0 = b \geq a = 0$) and our word constraint is satisfied. If, on the other hand, we do do project B, then $1 = b \geq a = 0$, and the constraint is again satisfied. Now consider what happens if we actually do project A, *i.e.* $a = 1$. Then the constraint is violated if $b = 0$ and is satisfied (0 is not ≥ 1) if $b = 1$, in other words we do project B. The last case to consider is if we do not do project A, *i.e.* $a = 0$ and we do project B, *i.e.* $b = 1$. Then the constraint is satisfied ($1 \geq 0$) and the word constraint is indeed satisfied. We can lay these constraints out in a table. There are only four possible conditions: {do A, do B}, {do A, do not do B}, {do not do A, do not do B}, {do not do A, do B}, and we can see that the illegal one is ruled out by the algebraic constraint.

Table 3.1: Evaluation of a binary implication constraint

$b \geq a$	a=0	a=1
b=0	Yes	No
b=1	Yes	Yes

The next word constraint we consider is 'if we do project A then we must not do project B'. How might we set about modeling this? The way to think of it is to notice that the property of *notdoing* B can be modeled very easily when we already have a binary variable *b* representing doing B. We invent a new variable

$$\overline{b} = 1 - b$$

where $\overline{b}$ represents the doing of project $\overline{B}$ *i.e.*, the project 'not doing B'. If $b = 1$ then $\overline{b} = 0$ (in other words, if we do project B then we do not do 'not B') whereas if $b = 0$ then $\overline{b} = 1$ (if we do not do project B then we do project $\overline{B}$). This is very convenient and $\overline{b}$ is called the **complement** of b. We can use this trick frequently. Just above we learned how to model 'if we do A then we must do B' and now we are trying to model 'if we do A then we must not do B', *i.e.* 'if we do A then we must do $\overline{B}$'. As 'if we do A then we must do B' was modeled by $b \geq a$ we can immediately see that the constraint 'if we do A then we must do $\overline{B}$' can be obtained by replacing b by $\overline{b}$ in the constraint, in other words $\overline{b} \geq a$. Replacing $\overline{b}$ by $1 - b$ we get

$$1 - b \geq a$$

or

$$1 \geq a + b$$

or alternatively

$$a + b \leq 1$$

Now that we have obtained this constraint, it is quite obvious. What it says is that if we do project A ($a = 1$) then b must be 0. This is exactly what we wanted to model. The point of the somewhat long-winded argument we showed above, however, is that we have used the result from the first logical constraint that we wanted to model, plus the fact that we have now introduced the notion of the complement of the project, to build up a newer and more complicated constraint from these two primitive concepts. This is what we will do frequently in what follows.

We see an example of this by trying to model the word constraint 'if we do not do A then we must do B', in other word, 'if not A then B'. We can go back immediately to our first logical constraint 'if we do A then we must do B' which was modeled as $b \geq a$. Now we are actually replacing A by not A, so we can see immediately that our constraint is

$$b \geq 1 - a$$

which is

$$a + b \geq 1$$

Again this constraint is obvious now that we have got to it. If we do not do A then $a = 0$, so $b \geq 1$ and since the maximum value of b is 1 then this immediately means that $b = 1$. Again we have just taken our knowledge of how to model 'if A then B' and the notion of the complement of a variable to be able to build up a more complex constraint.

The next constraint to consider is 'if we do project A we must do project B, and if we do project B we must do project A'. We have seen that the first is modeled as $b \geq a$ and the second as $a \geq b$. Combining these two constraints we get

$$a = b$$

in other words projects A and B are selected or rejected together, which is exactly what we expressed in our word constraint.

3.3.2.3 Implications with three variables

The next constraint we consider is 'if we do project A then we must do project B and project C'. The first thing to note is that this is in fact two constraints. One, the first, is 'if we do A then we must do project B' and the second constraint is 'if we do A then we must do project C'. With this observation we can see that the word constraint can be modeled by the two inequalities

$$b \geq a \text{ and } c \geq a$$

so that if $a = 1$, then both $b = 1$ and $c = 1$.

Another constraint might be 'if we do project A then we must do project B or project C'. It is like the previous constraint, except we now have an 'or' in place of the 'and'. The constraint

$$b + c \geq a$$

models this correctly. To see this, consider the following situation. If a is 0 then $b + c$ can be anything, and so b and c are not constrained. If $a = 1$, then one or both of b and c must be 1.

We may also try to model the inverse situation: 'if we do Project B or project C then we must do A'. This is again a case that may be formulated as two separate constraints: 'if we do B then we must do A' and 'if we do C then we must do A', giving rise to the following two inequalities

$$a \geq b \text{ and } a \geq c$$

so that if either $b = 1$ or $c = 1$, then we necessarily have $a = 1$.

A harder constraint to model is the following 'if we do both B and C then we must do A'. How might we model this? One way to think about it is to express it in the following way: 'if we do both B and C then we must not do not-A', or, 'we can do at most two of B, C or not-A' which we would model as

$$b + c + (1 - a) \leq 2$$

or in other words

$$b + c - a \leq 1$$

or perhaps more conventionally

$$a \geq b + c - 1$$

Looking at this last inequality, we can see that there is no effect on a when b and c are 0, or when just one of b and c is 1, but a does have to be ≥ 1 when both b and c are 1. A binary variable having to be greater than or equal to 1 means that the binary variable has to be precisely 1.

3.3.2.4 Generalized implications

Generalizing the previous we now might try to model 'if we do two or more of B, C, D or E then we must do A', and our first guess to this might be the constraint

$$a \geq b + c + d + e - 1$$

Certainly if, say, b and c are both equal to 1 then we have $a \geq 1$ and so a must equal 1, but the problem comes when three of the variables are equal to 1, say b, c and d. Then the constraint says that $a \geq 3 - 1$, *i.e.* $a \geq 2$, which is impossible as a is a binary variable. So we have to modify the constraint as follows

$$a \geq \frac{1}{3} \cdot (b + c + d + e - 1)$$

The biggest value that the expression inside the parentheses can take is 3, if $b = c = d = e = 1$. The $\frac{1}{3}$ in front of the parenthesis means that in the worst case a must be ≥ 1 (so a is equal to 1). But we must verify that the constraint is true if, say, just $b = c = 1$ (and d and e are equal to 0). In this case we have that $a \geq \frac{1}{3} \cdot (1 + 1 + 0 + 0 - 1)$ *i.e.* $a \geq \frac{1}{3}$. But since a can only take on the values 0 or 1 then the constraint $a \geq \frac{1}{3}$ means that a must be 1. This is exactly what we want.

We can generalize this to modeling the statement 'if we do M or more of N projects (B, C, D, ...) then we must do project A' by the constraint

$$a \geq \frac{b + c + d + \ldots - M + 1}{N - M + 1}$$

So far we have only given one way of modeling each of these constraints. We return to the constraint 'if we do B or C then we must do A' which we modeled as two constraints $a \geq b$ and $a \geq c$. It is possible to model this with just one constraint if we take this as a special case of the 'M or more from N' constraint we have just modeled. 'If we do B or C then we must do A' is the same as 'if we do M or more of N projects (B, C, D, ...) then we must do project A' with $M = 1$ and $N = 2$. So the constraint is

$$a \geq \frac{b + c - M + 1}{N - M + 1} \quad \text{i.e.}$$
$$a \geq \frac{b + c - 1 + 1}{2 - 1 + 1} \quad \text{i.e.}$$
$$a \geq \frac{1}{2} \cdot (b + c)$$

So this single constraint is exactly the same in terms of binary variables as the two constraints which we produced before. Which of these two representations is better? In fact the representation in terms of two constraints is better. But both of the two are correct and both will give a correct answer if put into an Integer Programming system. It is just that the first pair of constraints will in general give a solution more rapidly.

More and more complicated constraints can be built up from the primitive ideas we have explored so far. Since these more complicated constraints do not occur very frequently in actual practical modeling we shall not explore them further. Table 3.2 summarizes the formulations of logical conditions we have seen in the different paragraphs of Section 3.3.2.

Table 3.2: Formulation of logical conditions using binary variables

At most one of A, B,...,H	$a+b+c+d+e+f+g+h \leq 1$
Exactly two of A, B,...,H	$a+b+c+d+e+f+g+h = 2$
If A then B	$b \geq a$
Not B	$\overline{b} = 1-b$
If A then not B	$a+b \leq 1$
If not A then B	$a+b \geq 1$
If A then B, and if B then A	$a = b$
If A then B and C	$b \geq a$ and $c \geq a$
If A then B or C	$b+c \geq a$
If B or C then A	$a \geq b$ and $a \geq c$
	or alternatively: $a \geq \frac{1}{2} \cdot (b+c)$
If B and C then A	$a \geq b+c-1$
If two or more of B, C, D or E then A	$a \geq \frac{1}{3} \cdot (b+c+d+e-1)$
If M or more of N projects (B, C, D, ...) then A	$a \geq \frac{b+c+d+\ldots-M+1}{N-M+1}$

3.3.3 Products of binary variables

We move on to modeling the product of binary variables. Suppose we have three binary variables b_1, b_2 and b_3 and we want to model the equation

$$b_3 = b_1 \cdot b_2$$

This is not a linear equation since it involves the product of two variables, so we have to express it in some linear form. There is a trick and it is another one of these tricks that one just has to learn. Consider the following set of three inequalities

$$b_3 \leq b_1$$
$$b_3 \leq b_2$$
$$b_3 \geq b_1 + b_2 - 1$$

Then we claim that this represents the product expression that we wish to model. To see we construct the following Table 3.3.

Table 3.3: Product of two binaries

b_1	b_2	b_3	$b_3 = b_1 \cdot b_2$?	$b_3 \leq b_1$?	$b_3 \leq b_2$?	$b_3 \geq b_1 + b_2 - 1$?
0	0	0	Yes	Yes	Yes	Yes
0	0	1	No	No	No	Yes
0	1	0	Yes	Yes	Yes	Yes
0	1	1	No	No	Yes	Yes
1	0	0	Yes	Yes	Yes	Yes
1	0	1	No	Yes	No	Yes
1	1	0	No	Yes	Yes	No
1	1	1	Yes	Yes	Yes	Yes

We can see that the column headed $b_3 = b_1 \cdot b_2$? is true if and only if we have a 'Yes' in the three columns $b_3 \leq b_1$?, $b_3 \leq b_2$? and $b_3 \geq b_1 + b_2 - 1$?, so the three linear equations do exactly represent and are true at exactly the same time as the product is true.

This is a particularly long winded way of demonstrating the equivalence of the product term and the three linear equations and in fact now we have got it it is actually quite easy to see why these three inequalities are correct. Since b_3 is b_1 multiplied by something that is less than or equal to 1, b_3 will always be less than or equal to b_1 and by a similar argument b_3 will always be less than or equal to b_2. The only further case we have to consider is when both b_1 and b_2 are equal to 1 and then we have to force b_3 to be equal to 1. This is done by the constraint $b_3 \geq b_1 + b_2 - 1$ which is non restrictive if only one or none of b_1 and b_2 are 1 but forces b_3 to be 1 when $b_1 = b_2 = 1$.

Looking at the constraint this way immediately enables us to model for instance

$$b_4 = b_1 \cdot b_2 \cdot b_3,$$

in other words the product of three variables, as the four constraints

$$b_4 \leq b_1$$

$$b_4 \leq b_2$$
$$b_4 \leq b_3$$
$$b_4 \geq b_1 + b_2 + b_3 - 2$$

If any of b_1, b_2 and b_3 are 0 then b_4 must be 0 but if b_1, b_2 and b_3 are 1 then b_4 must be greater than or equal to $3 - 2$, *i.e.* b_4 must be greater than or equal to 1 so b_4 must be 1.

3.3.4 Dichotomies: either/or constraints

All the constraints we have seen so far have had to be satisfied simultaneously, but sometimes we need to model that either Constraint 1 or Constraint 2 has to be satisfied, not necessarily both of them. Consider the problem:

minimize	$Z = x_1 + x_2$
subject to	$x_1 \geq 0, x_2 \geq 0$ and
	Either $2 \cdot x_1 + x_2 \geq 6$ (Constraint 1) or $x_1 + 2 \cdot x_2 \geq 7$ (Constraint 2)

Since we have just two variables, we can graph the feasible region, and we have done this in the figure below where the grey shaded area is the feasible region.

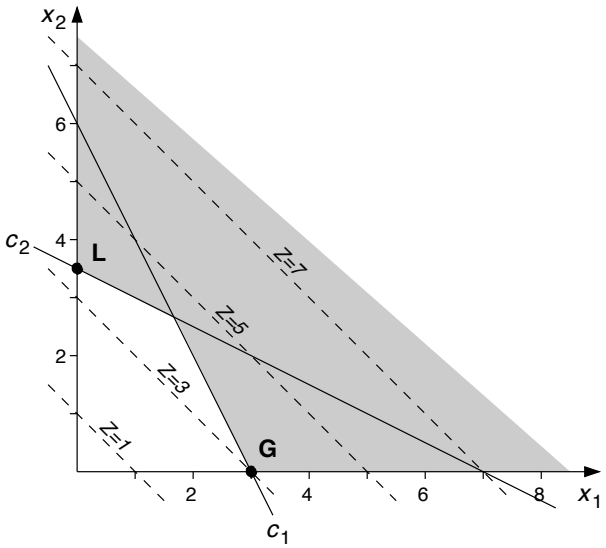

Figure 3.2: Example of either/or constraints

Point L ($x_1 = 0, x_2 = 3.5, Z = 3.5$) is a local optimum. Point G ($x_1 = 3, x_2 = 3, Z = 3$) is the global minimum.

We can model this with one additional binary variable b.

$$2 \cdot x_1 + x_2 \geq 6 \cdot b$$
$$x_1 + 2 \cdot x_2 \geq 7 \cdot (1 - b)$$

To show that this is true, we just have to consider the two cases:

$b = 0$:	$2 \cdot x_1 + x_2 + x_3 \geq 0$	$x_1 + 2 \cdot x_2 + 3 \cdot x_3 \geq 7$	Constraint 2 is satisfied
$b = 1$:	$2 \cdot x_1 + x_2 + x_3 \geq 6$	$x_1 + 2 \cdot x_2 + 3 \cdot x_3 \geq 0$	Constraint 1 is satisfied

3.4 Binary variables 'do everything'

All global entities (general integers, partial integers, semi-continuous variables, and both sorts of Special Ordered Sets) can be expressed in terms of binary variables. However, as shown by the examples in this section, it is usually preferable to use the specific global entities.