Lesson 7: Logical If/Then and Either/Or Constraints

1 A Binary IP

USNA is considering purchasing some new generators to ensure power in the case of storms. The table below gives the fixed cost and MW of power generated by each potential generator.

	Fixed cost ($ million)	Power Generated (MW)
Generator 1	1.20	3,000
Generator 2	0.75	4,000
Generator 3	0.50	5,000
Generator 4	1.00	3,500
Generator 5	1.10	3,800

USNA wants to ensure that at least 8500 MW of power are available if there is an outage.

1. Formulate a concrete IP that would tell USNA the optimal generators to purchase while satisfying power demands at minimum cost.

2 Types of Binary Variable Constraints

In general, there are several types of logic constraints we'd like to enforce:

- Fixed Charge (Lesson 7)
- Mutually exclusive and multiple choice constraints
- If/then constraints (binary and continuous variables)
- Either/or constraints

3 Mutually Exclusive and Multiple Choice Constraints

The two simplest type of logical constraints deal with mutual exclusion or multiple choice.

- **Mutual Exclusive:** You are only allowed to choose a specific subset of variables to be 1.
    - Suppose in the generator problem above, you are only allowed to choose at most 1 of generators 2, 3, and 4. Write a logical constraint that enforces this.

- **Multiple Choice:** You must choose a certain subset of variables to be equal to 1.
    - Suppose that USNA knows that facilities will only approve the purchase of exactly 2 of generators 1, 2, 3, 4, and 5. Write a logical constraint that enforces this.

4 If/Then Constraints on Binary Variables

Often, it is the case that we want to enforce if/then logic on binary variables. For example, perhaps Generator 1 and 3 are made by competing companies, so if Generator 1 is purchased, we can not purchase Generator 3.

2. Try to write a constraint to capture this logic. That is write an inequality which says if $x_1 = 1$ then $x_3 = 0$.

It can seem like a lot of guess and check in order to get a constraint that actually works. Fortunately, there's a process you can follow to make this significantly easier. Namely, the following steps can make writing these logical constraints much easier.

- Write the constraint as a conditional statement and convert all clauses to 1s (why?)

- Write the constraint from left to right.
  - The left side of the **if** statement is the LHS of the constraint
  - The inequality is always $\leq$
  - If it is converted to 1s, everything following **then** is the RHS of the constraint. That is the high value on the left "pushes" the RHS to its high value.
- **ALWAYS** check every logical constraint for **explicit** and **implicit** satisfaction.
  - **Explicit** satisfaction:

  - **Implicit** satisfaction:

Using the steps above, capture the following if/then constraint logic:

3. If we purchase generator 2 then we must purchase generator 4.

4. If we don't purchase generator 1 then we must purchase generator 5.

The same idea applies with 3 or more variables, but it does get more complicated and you have to think through the logic. We'll try two more:

5. If we purchase generator 2 or generator 1 then we can't purchase generator 5.

6. If we don't purchase generator 1 and we do purchase generator 3 then we must also purchase generator 2.

5 Either/Or Constraints: Motivating Example

Now, we switch gears to another application of binary variables.

Quality Cabinets used an integer-program to determine how many of each type of cabinet to make in order to maximize profit. They used decision variables x_s, x_d, and x_e to represent the number of standard, deluxe, and enhanced cabinets, respectively, to produce each week. Each cabinet requires a certain number of hours of painting time and there is a limit on the number of painting hours available. A small part of the model is:

Maximize	$25x_a + 45x_d + 60x_e$	
subject to	$2x_a + 4x_d + 5x_e \leq 700$	(painting time)

6 Model Update Requested

Now Quality Cabinets is considering renting better painting equipment and has asked us to update our model to help with this decision. The equipment will cost \$300 per week, but will reduce the time required to to do the painting by 15 minutes for a standard cabinet, by 30 minutes for a deluxe cabinet, and by 1 hour for an enhanced cabinet.

We decide to add a binary variable z. In the solution, if $z = 1$, then Quality Cabinets should rent the equipment. If $z = 0$, they should not.

7. Should the objective function change? If not, explain. If so, write the updated objective function.

8. If the equipment is not rented, what should the painting constraint be? Label it (A).

9. If the equipment is rented, what should the painting constraint be? Label it (B).

7 Logical (Either/Or) Constraints

There is no place for "if – then" statements in a math programming model, so we have to enforce this logic indirectly using linear constraints and binary variables.

> A constraint is *relaxed* if it has no impact. We can relax a "$\leq$" constraint by making the value on the right so large that it will never restrict the values of the variables on the left.

10. Rewrite constraint (A) above *using* z so that it is enforced if $z = 0$ (the equipment is not rented), but *relaxed* if $z = 1$ (the equipment is rented).

11. Rewrite constraint (B) above *using* z so that it is enforced if $z = 1$ (the equipment is rented), but *relaxed* if $z = 0$ (the equipment is not rented).

12. Write the updated version of the partial Quality Cabinets model here.

8 Logical Constraints Summary

Suppose $a^T x \leq b$ is a linear constraint, M is a number that is bigger than $a^T x$ could ever be (but not TOO big!–choose M wisely), and z is a binary variable.

A constraint that is enforces $a^T x \leq b$ if $z = 0$ and relaxes $a^T x \leq b$ if $z = 1$:

A constraint that is enforces $a^T x \leq b$ if $z = 1$ and relaxes $a^T x \leq b$ if $z = 0$:

9 Many more possibilities

There are many ways to creatively use linear constraints to enforce modeling requirements; this lesson contains only a few examples. Often this process takes some trial and error. **Always be sure to test the logic with various values of the decision variables to make sure the constraint is doing what you want it to do.**