Logic Modeling of Requirements

BY

Eng: Basma Elsharkawy

Logic Modeling

- Data flow diagrams do not show the logic inside the processes
  - what occurs within a process?
  - How input data is converted into output information
- Logic modeling involves representing internal structure and functionality of processes depicted on a DFD.
- Processes must be clearly described before translating them into programming language.
- Logic modeling can also be used to show when processes on a DFD occur.
- Logic modeling will be generic without taking syntax of a particular programming language

Logic Modeling Deliverables and Outcomes

Each process on the lowest level DFD will be represented by one or more of the following:

- Structured English
- Decision Tables
- Decision Trees
- State-transition diagrams
- Sequence diagrams
- Activity diagrams

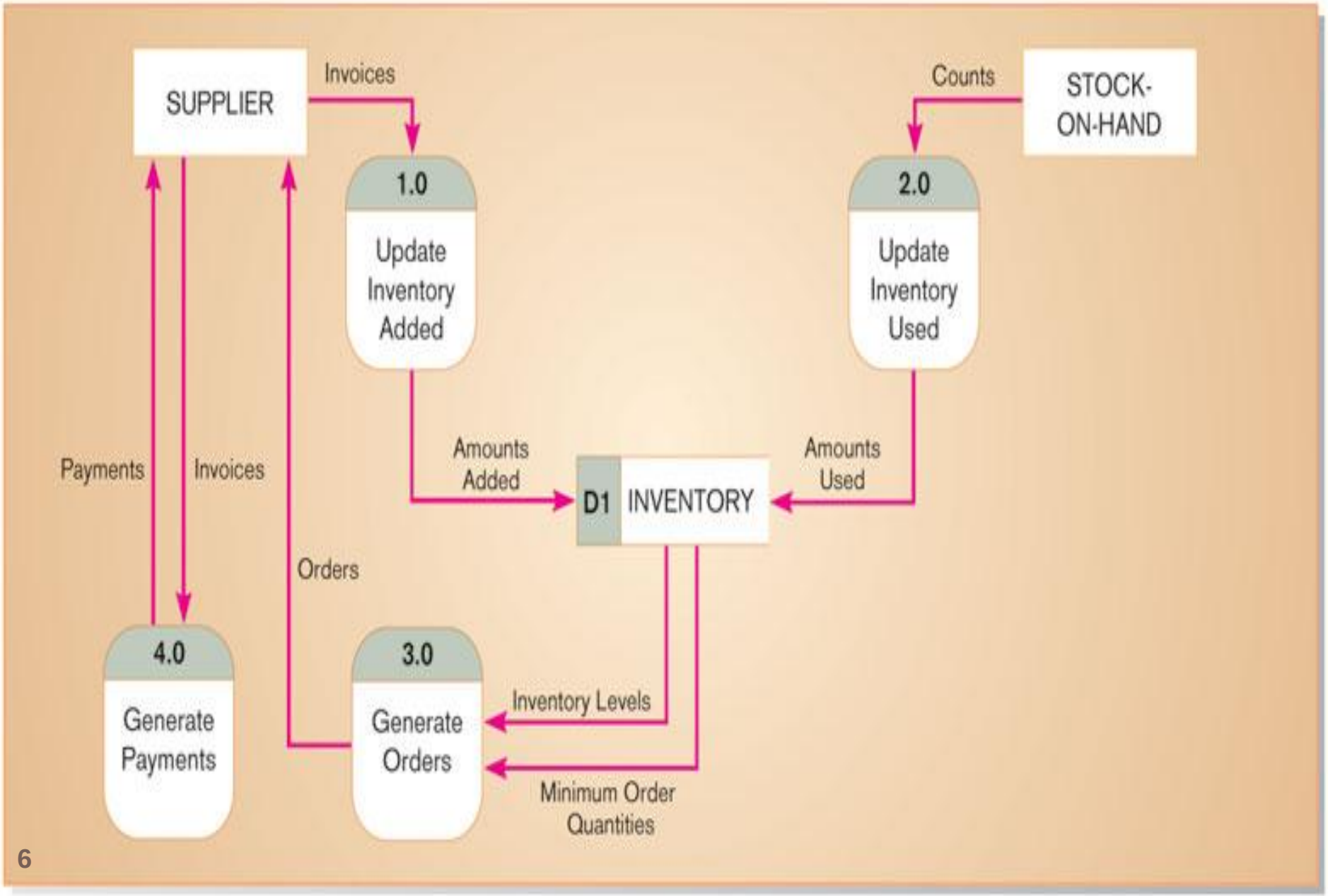
Modeling Logic with Structured English

- **Structured English** is a modified form of English used to specify the **logic** of ***information processes***
- Uses a subset of English vocabulary to express process procedures
  - *Action verbs* – read, write, print, move, merge, add, sort
  - *Noun phrases* – name, address
  - No adjectives or adverbs
- No specific standards – each analyst will have his own way
- File and variable names are **CAPITALIZED**
- **Logical comparisons** are *spelled out* and not used symbols
- Similar to programming language
  - **If conditions**
  - **Case statement**

Modeling Logic with Structured English

- It is possible to represent all three processes used in structured programming: sequence, conditional, repetition
  - **Sequence** – no special structure but one statement following another
  - **Conditional** – **IF** THEN ELSE statement; **CASE** statement
  - **Repetition** – **DO-UNTIL** loops or **DO-WHILE** loops

Current logical DFD for Hoosier Burger's inventory control system



Structured English representations of the four processes depicted in Figure 8-2

```
Process 1.0: Update Inventory Added
DO
    READ next Invoice-item-record
    FIND matching Inventory-record
    ADD Quantity-added from Invoice-item-record to Quantity-in-stock on
      Inventory-record
UNTIL End-of-file
```

```
Process 2.0: Update Inventory Used
DO
    READ next Stock-item-record
    FIND matching Inventory-record
    SUBTRACT Quantity-used on Stock-item-record from Quantity-in-stock on
      Inventory-record
UNTIL End-of-file
```

```
Process 3.0: Generate Orders
DO
    READ next Inventory-record
    BEGIN IF
        If Quantity-in-stock is less than Minimum-order-quantity
        THEN GENERATE Order
    END IF
UNTIL End-of-file
```

```
Process 4.0: Generate Payments
READ Today's-date
DO
    SORT Invoice-records by Date
    READ next Invoice-record
    BEGIN IF
        IF Date is 30 days or greater than Today's-date
        THEN GENERATE Payments
    END IF
UNTIL End-of-file
```

Structured English is used here to describe **input and output**.

Structured English representations of the four processes depicted in Figure 8-2

```
Process 1.0: Update Inventory Added
DO
    READ next Invoice-item-record
    FIND matching Inventory-record
    ADD Quantity-added from Invoice-item-record to Quantity-in-stock on
      Inventory-record
UNTIL End-of-file
```

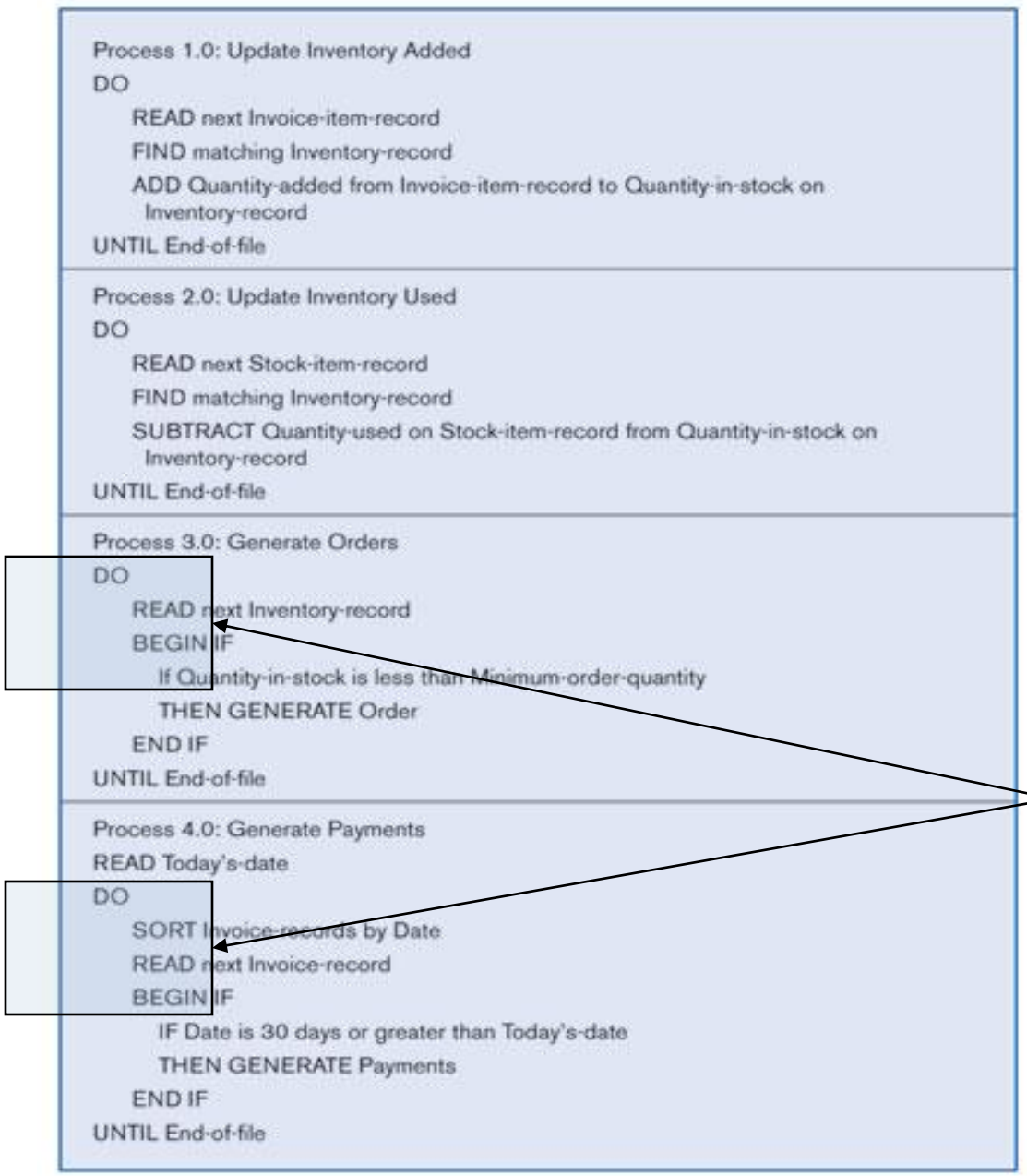
```
Process 2.0: Update Inventory Used
DO
    READ next Stock-item-record
    FIND matching Inventory-record
    SUBTRACT Quantity-used on Stock-item-record from Quantity-in-st
      Inventory-record
UNTIL End-of-file
```

```
Process 3.0: Generate Orders
DO
    READ next Inventory-record
    BEGIN IF
        If Quantity-in-stock is less than Minimum-order-quantity
        THEN GENERATE Order
    END IF
UNTIL End-of-file
```

```
Process 4.0: Generate Payments
READ Today's-date
DO
    SORT Invoice-records by Date
    READ next Invoice-record
    BEGIN IF
        IF Date is 30 days or greater than Today's-date
        THEN GENERATE Payments
    END IF
UNTIL End-of-file
```

Structured English is used here to describe repetition.

Structured English representations of the four processes depicted in Figure 8-2



```
Process 1.0: Update Inventory Added
DO
    READ next Invoice-item-record
    FIND matching Inventory-record
    ADD Quantity-added from Invoice-item-record to Quantity-in-stock on
      Inventory-record
UNTIL End-of-file

Process 2.0: Update Inventory Used
DO
    READ next Stock-item-record
    FIND matching Inventory-record
    SUBTRACT Quantity-used on Stock-item-record from Quantity-in-stock on
      Inventory-record
UNTIL End-of-file

Process 3.0: Generate Orders
DO
    READ next Inventory-record
    BEGIN IF
      If Quantity-in-stock is less than Minimum-order-quantity
      THEN GENERATE Order
    END IF
UNTIL End-of-file

Process 4.0: Generate Payments
READ Today's-date
DO
    SORT Invoice-records by Date
    READ next Invoice-record
    BEGIN IF
      IF Date is 30 days or greater than Today's-date
      THEN GENERATE Payments
    END IF
UNTIL End-of-file
```

Structured English is used here to describe decisions.

Structured English representations of the four processes depicted in Figure 8-2

```
Process 1.0: Update Inventory Added
DO
    READ next Invoice-item-record
    FIND matching Inventory-record
    ADD Quantity-added from Invoice-item-record to Quantity-in-stock on
      Inventory-record
UNTIL End-of-file
```

```
Process 2.0: Update Inventory Used
DO
    READ next Stock-item-record
    FIND matching Inventory-record
    SUBTRACT Quantity-used on Stock-item-record from Quantity-in-stock on
      Inventory-record
UNTIL End-of-file
```

```
Process 3.0: Generate Orders
DO
    READ next Inventory-record
    BEGIN IF
      If Quantity-in-stock less than Minimum-order-quantity
      THEN GENERATE Order
    END IF
UNTIL End-of-file
```

```
Process 4.0: Generate Payments
READ Today's-date
DO
    SORT Invoice-records by Date
    READ next Invoice-record
    BEGIN IF
      IF Date is 30 days or greater than Today's-date
      THEN GENERATE Payments
    END IF
UNTIL End-of-file
```

Structured English is used here to describe invoking other processes.

Modeling Logic with Decision Tables

- Structured English is not good to represent ***complicated logic* (having several different conditions)** as it becomes difficult to understand
- **Decision table**: A **matrix** representation of the logic of a decision
- Specifies all the **possible conditions and the resulting actions** in a **tabular form**
- Best used for complicated decision logic

3 Parts of a Decision Table

1. Condition stubs
   - Lists condition relevant to decision
2. Action stubs
   - Actions that result from a given set of conditions
3. Rules
   - Specify which actions are to be followed for a given set of conditions

Indifferent Condition

- Condition whose value *does* **not affect which action** is taken for two or more rules

Procedure for Creating Decision Tables

- Name the conditions and values each condition can assume
  - some conditions values will be just "**yes**" or "**no**
- Name all possible actions that can occur
- List all possible rules
- Define the actions for each rule
- Simplify the table
  - Remove any rules with **impossible** actions

Decision Table

Complete decision table for payroll system example

	Conditions/ Courses of Action	Rules					
		1	2	3	4	5	6
Condition Stubs	Employee type	S	H	S	H	S	H
	Hours worked	<40	<40	40	40	>40	>40
Action Stubs	Pay base salary	X		X		X	
	Calculate hourly wage		X		X		X
	Calculate overtime						X
	Produce Absence Report		X				

Employee Type:

S: Salaried

H: Hourly paid

Note: for salaried employees the action stub chosen will always be the same…therefore hours worked is an *indifferent condition*

Reduced Decision Table

Indifferent condition

Condition whose value does not affect which action is taken for two or more rules

Reduced decision table for payroll system example

Conditions/ Courses of Action	Rules			
	1	2	3	4
Employee type	S	H	H	H
Hours worked	–	<40	40	>40
Pay base salary	X			
Calculate hourly wage		X	X	X
Calculate overtime				X
Produce Absence Report		X		

Because of indifferent condition, the complete decision table can be reduced to one with fewer rules

Example of decision table for

Hoosier Burger's Inventory reordering

Conditions/ Courses of Action	Rules											
	1	2	3	4	5	6	7	8	9	10	11	12
Type of item	P	N	P	N	P	N	P	N	P	N	P	N
Time of week	D	D	W	W	D	D	W	W	D	D	W	W
Season of year	A	A	A	A	S	S	S	S	H	H	H	H
Standing daily order	X				X				X			
Standing weekend order			X				X				X	
Minimum order quantity		X		X		X		X		X		X
Holiday reduction									X		X	
Summer reduction					X		X					

Type of item:
P = perishable
N = nonperishable

Time of week:
D = weekday
W = weekend

Season of year:
A = academic year
S = summer
H = holiday

Reduced decision table for Hoosier Burger's Inventory reordering

Conditions/ Courses of action	Rules						
	1	2	3	4	5	6	7
Type of item	P	P	P	P	P	P	N
Time of week	D	W	D	W	D	W	-
Season of year	A	A	S	S	H	H	-
Standing daily order	X		X		X		
Standing weekend order		X		X		X	
Minimum order quantity							X
Holiday reduction					X	X	
Summer reduction			X	X			

Example 2

- Policy for charging charter flight costumers for certain in-flight services:
- If the flight is **more than half-full** and costs more than **$350 per seat**, we serve **free cocktails** unless it is a domestic flight.
- We **charge** for cocktails on all **domestic flights**; that is, for all the ones where we serve cocktails.

  (Cocktails are only served on flights that are more than half-full.)

List all the conditions that determine which action to take

Conditions	Values
The flight more than half-full?	Yes (Y), No (N)
Cost is more than $350?	Y, N
Is it a domestic flight?	Y, N

Fill all rules in the table

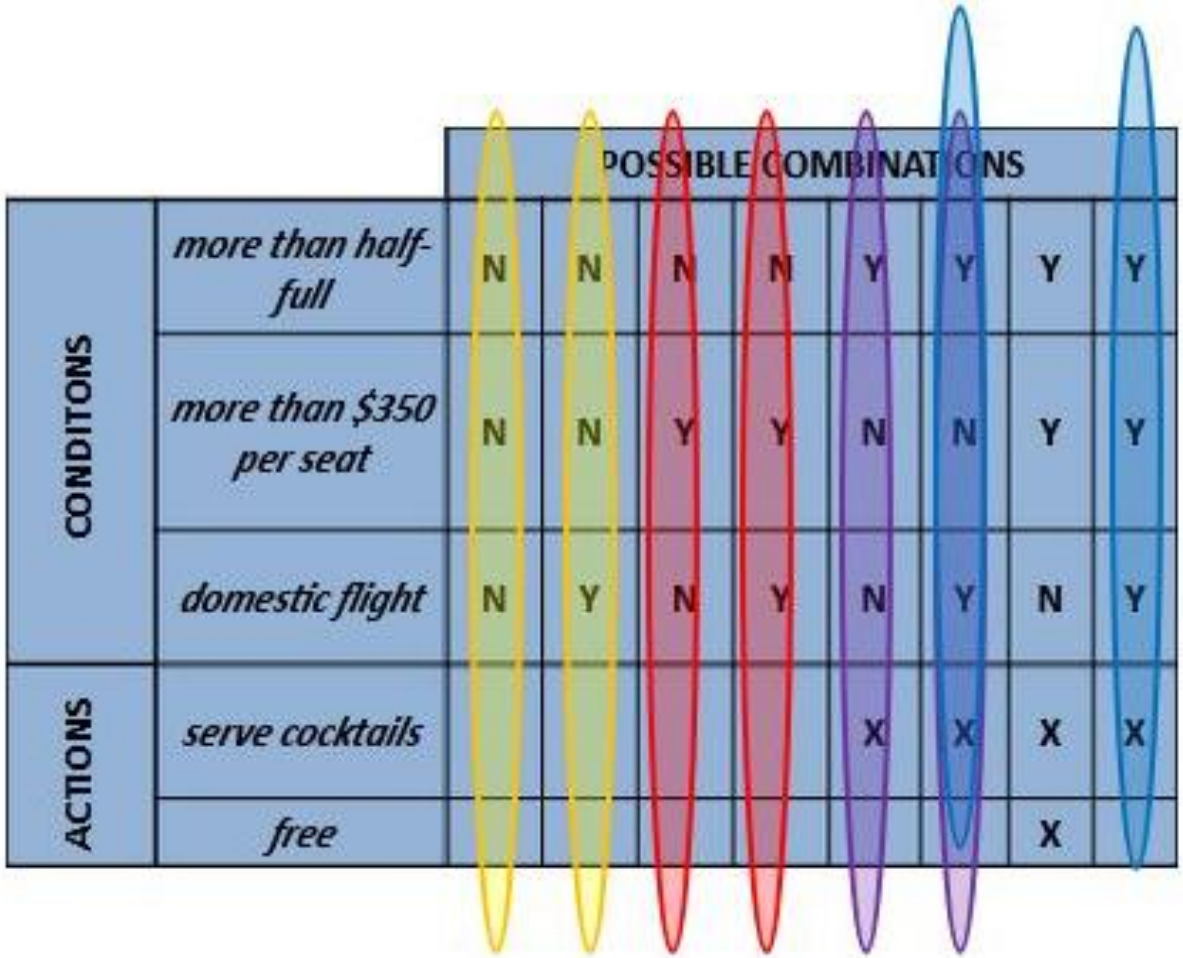
		POSSIBLE RULES							
CONDITONS	*more than half-full*	N	N	N	N	Y	Y	Y	Y
	more than $350 per seat	N	N	Y	Y	N	N	Y	Y
	domestic flight	N	Y	N	Y	N	Y	N	Y
ACTIONS									

CONT…..

- Analyze column by column to determine which actions are appropriate for each combination

		POSSIBLE RULES							
CONDITONS	*more than half-full*	N	N	N	N	Y	Y	Y	Y
	more than $350 per seat	N	N	Y	Y	N	N	Y	Y
	domestic flight	N	Y	N	Y	N	Y	N	Y
ACTIONS	*serve cocktails*					X	X	X	X
	free							X	

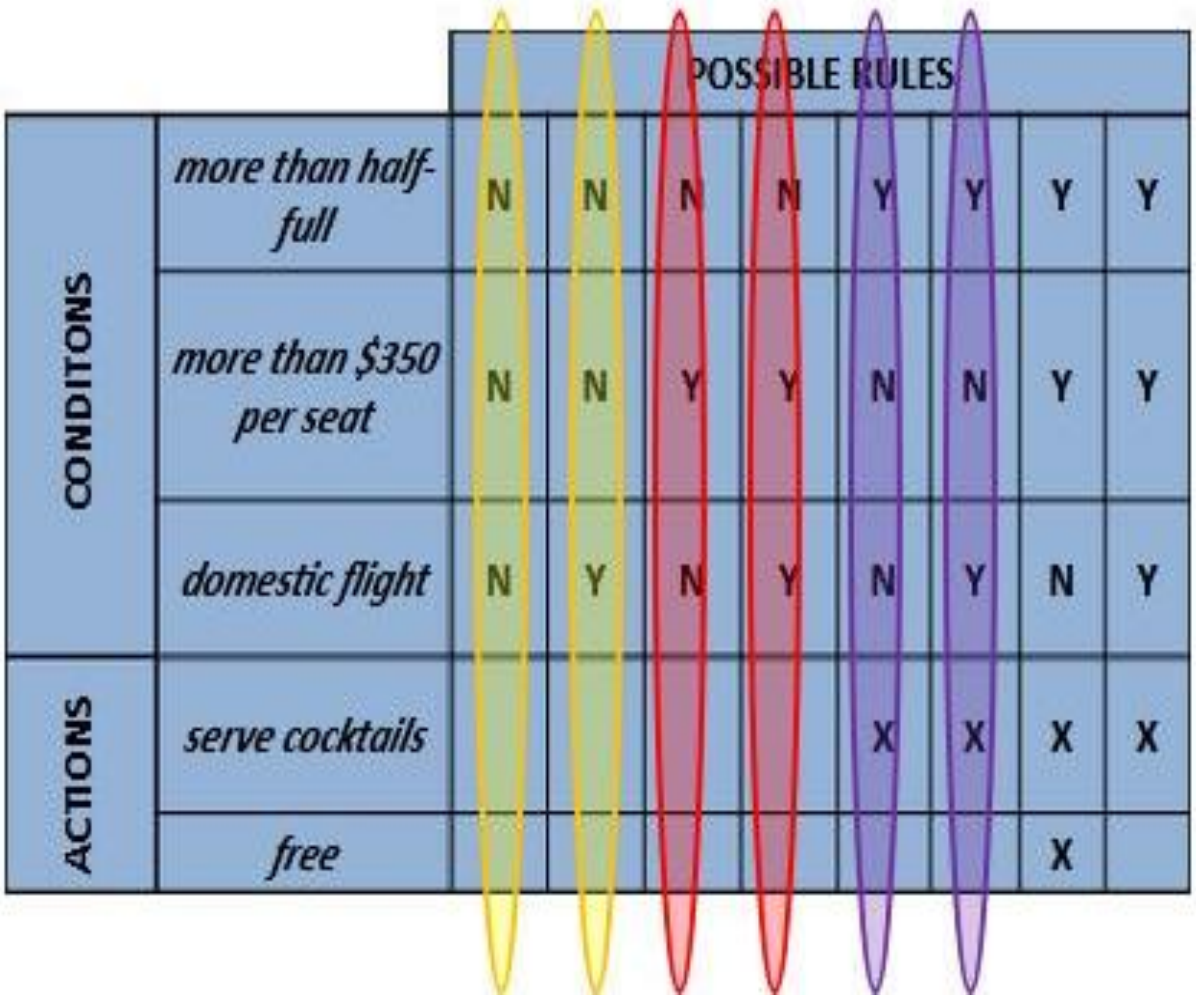
Reduce the table by eliminating redundant columns.



		POSSIBLE COMBINATIONS							
CONDITONS	*more than half-full*	N	N	N	N	Y	Y	Y	Y
	more than $350 per seat	N	N	Y	Y	N	N	Y	Y
	domestic flight	N	Y	N	Y	N	Y	N	Y
ACTIONS	*serve cocktails*					X	X	X	X
	free							X	

Note that some columns are identical except for one condition.

Reduce the table by eliminating redundant columns.

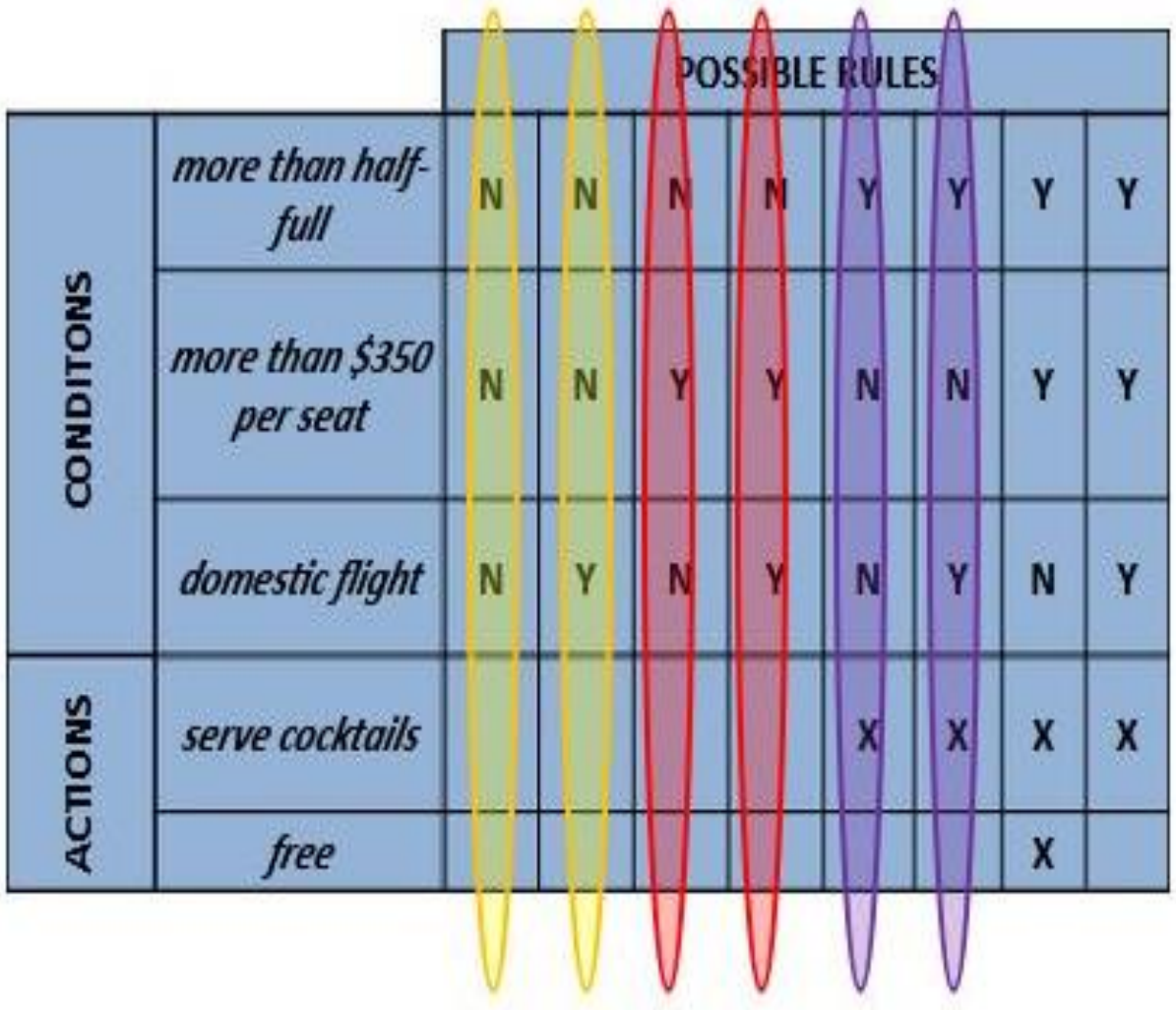


		POSSIBLE RULES							
CONDITONS	*more than half-full*	N	N	N	N	Y	Y	Y	Y
	more than $350 per seat	N	N	Y	Y	N	N	Y	Y
	domestic flight	N	Y	N	Y	N	Y	N	Y
ACTIONS	*serve cocktails*					X	X	X	X
	free							X	

Note that some columns are identical except for one condition.

Which means that actions are independent from the value of that particular condition.

Reduce the table by eliminating redundant columns.



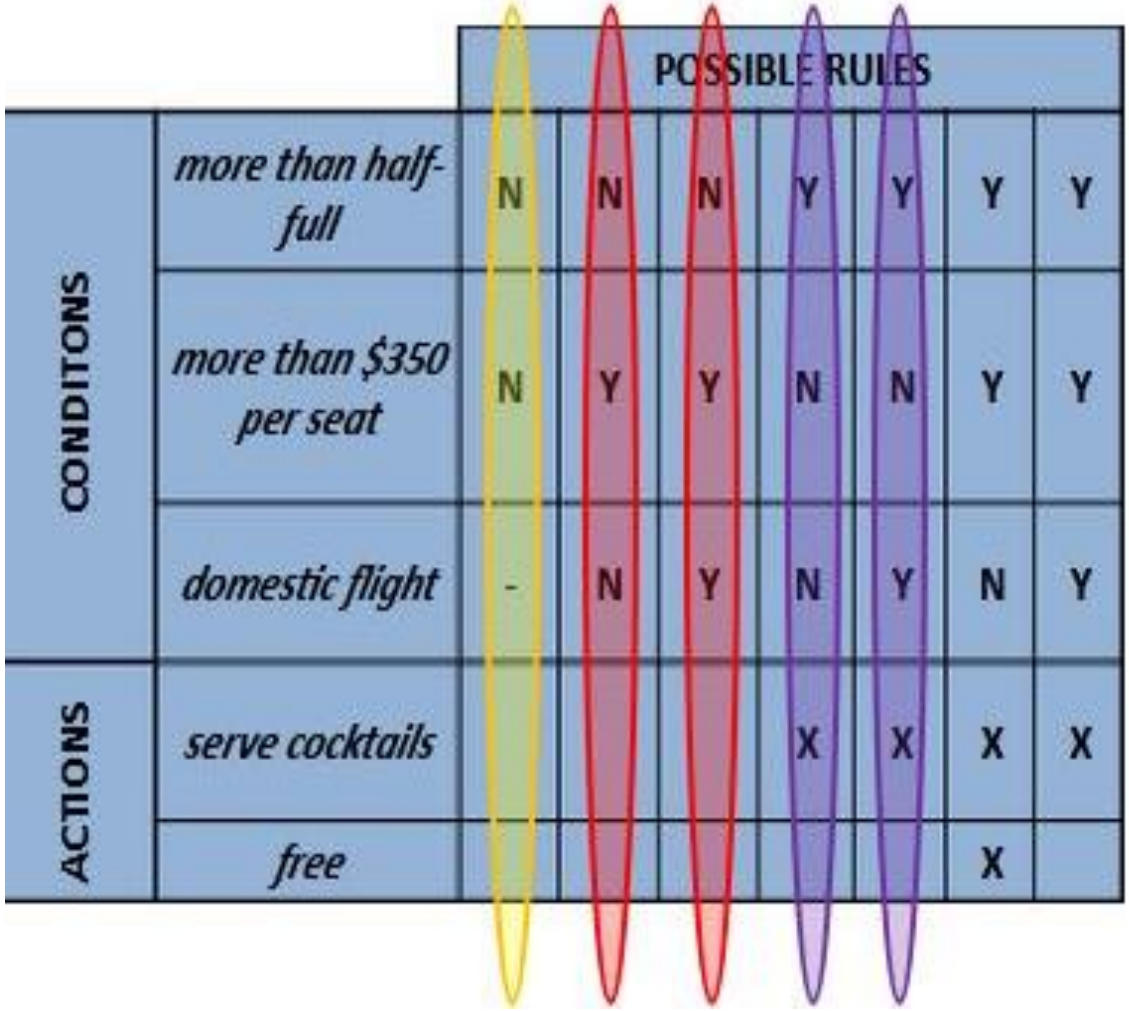
		POSSIBLE RULES							
CONDITONS	*more than half-full*	N	N	N	N	Y	Y	Y	Y
	more than $350 per seat	N	N	Y	Y	N	N	Y	Y
	domestic flight	N	Y	N	Y	N	Y	N	Y
ACTIONS	*serve cocktails*					X	X	X	X
	free							X	

Note that some columns are identical except for one condition.

Which means that actions are independent from the value of that particular condition.

Hence, the table can be simplified.

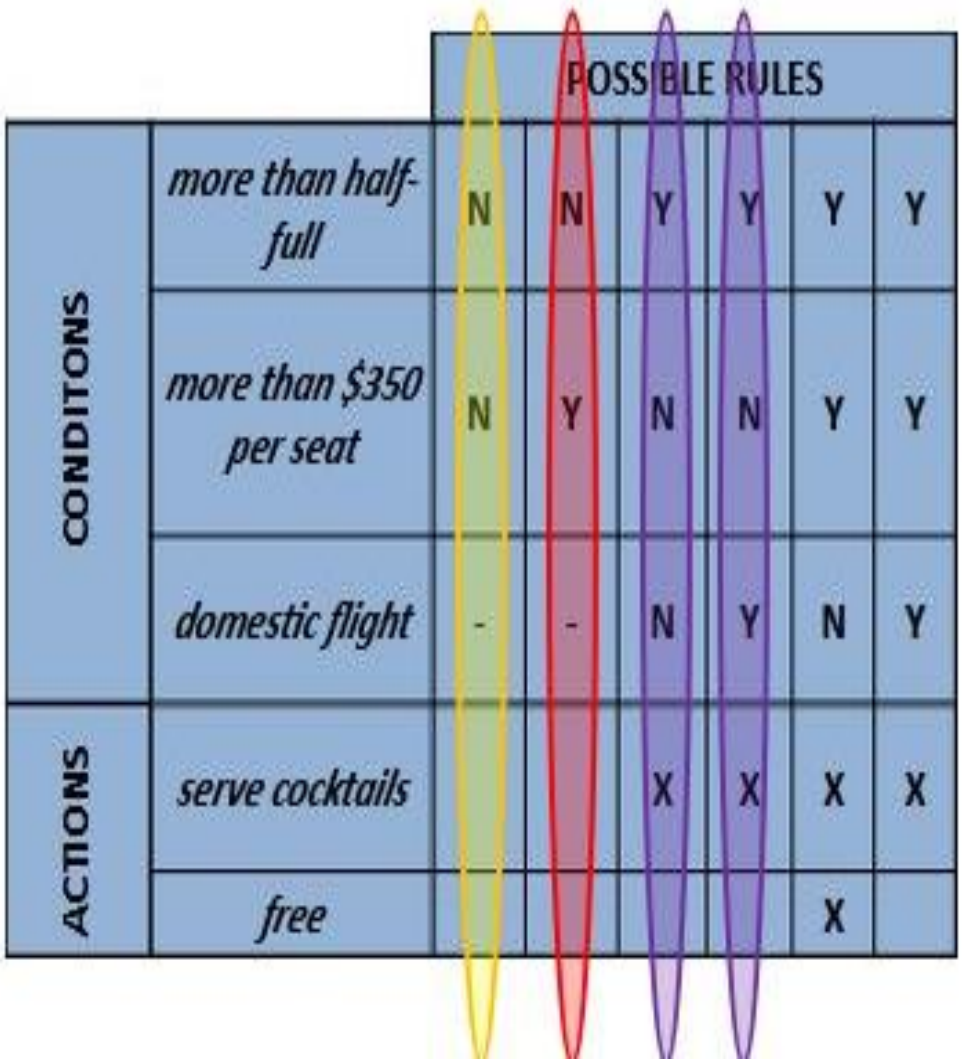
Reduce the table by eliminating redundant columns.



		POSSIBLE RULES						
CONDITIONS	*more than half-full*	N	N	N	Y	Y	Y	Y
	more than $350 per seat	N	Y	Y	N	N	Y	Y
	domestic flight	-	N	Y	N	Y	N	Y
ACTIONS	*serve cocktails*				X	X	X	X
	free						X	

First we combine the yellow ones nullifying the condition.

Reduce the table by eliminating redundant columns.

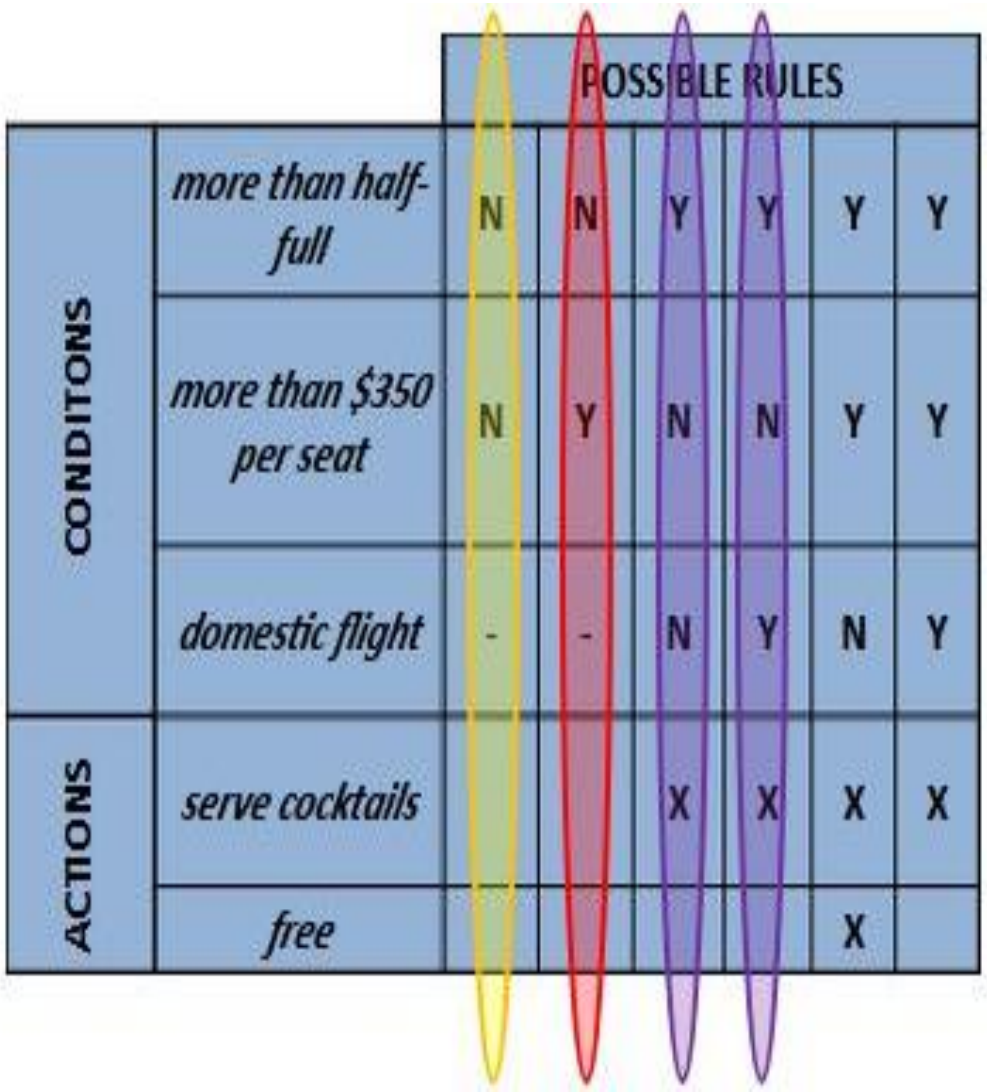


		POSSIBLE RULES					
CONDITONS	*more than half-full*	N	N	Y	Y	Y	Y
	more than $350 per seat	N	Y	N	N	Y	Y
	domestic flight	-	-	N	Y	N	Y
ACTIONS	*serve cocktails*			X	X	X	X
	free					X	

First we combine the yellow ones nullifying the condition.

Then the red ones.

Reduce the table by eliminating redundant columns.



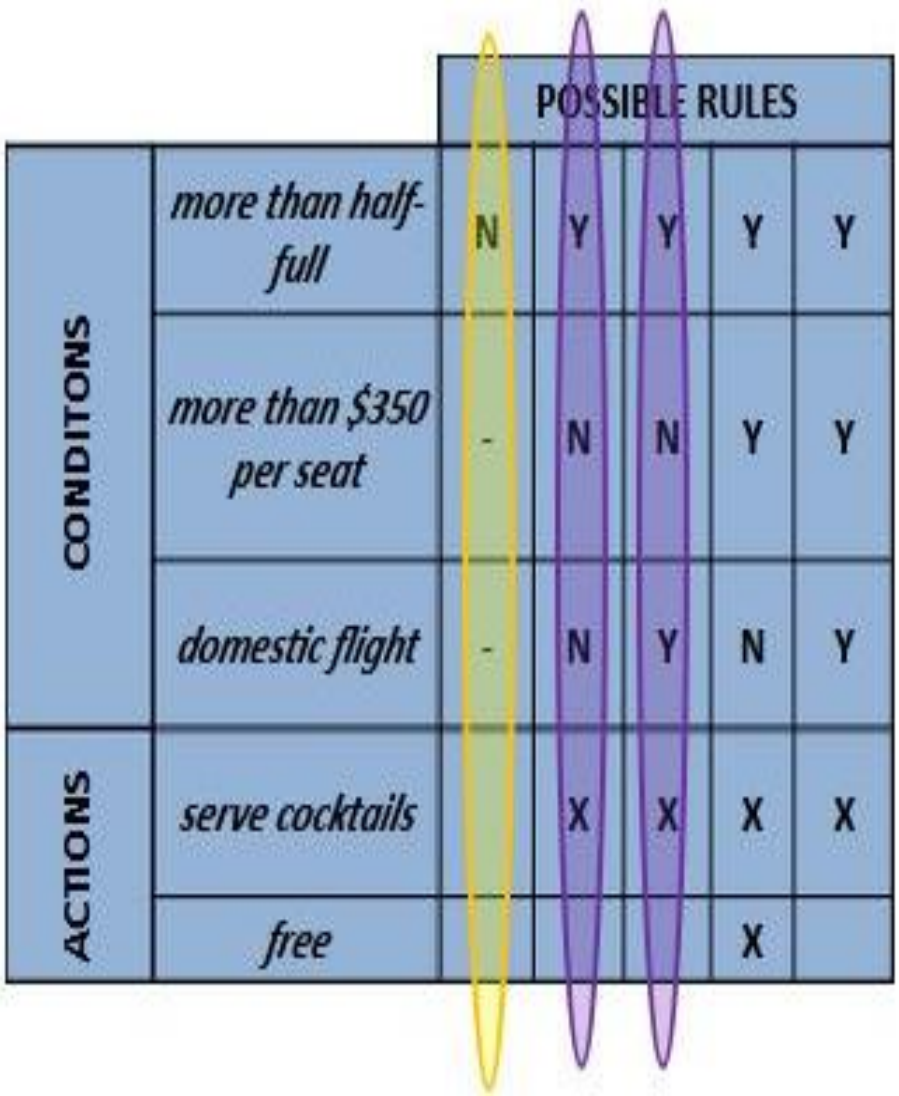
		POSSIBLE RULES					
CONDITONS	*more than half-full*	N	N	Y	Y	Y	Y
	more than $350 per seat	N	Y	N	N	Y	Y
	domestic flight	-	-	N	Y	N	Y
ACTIONS	*serve cocktails*			X	X	X	X
	free					X	

First we combine the yellow ones nullifying the condition.

Then the red ones.

Notice that yellow and red columns are identical but by one condition.

Reduce the table by eliminating redundant columns.



		POSSIBLE RULES				
CONDITONS	*more than half-full*	N	Y	Y	Y	Y
	more than $350 per seat	-	N	N	Y	Y
	domestic flight	-	N	Y	N	Y
ACTIONS	*serve cocktails*		X	X	X	X
	free				X	

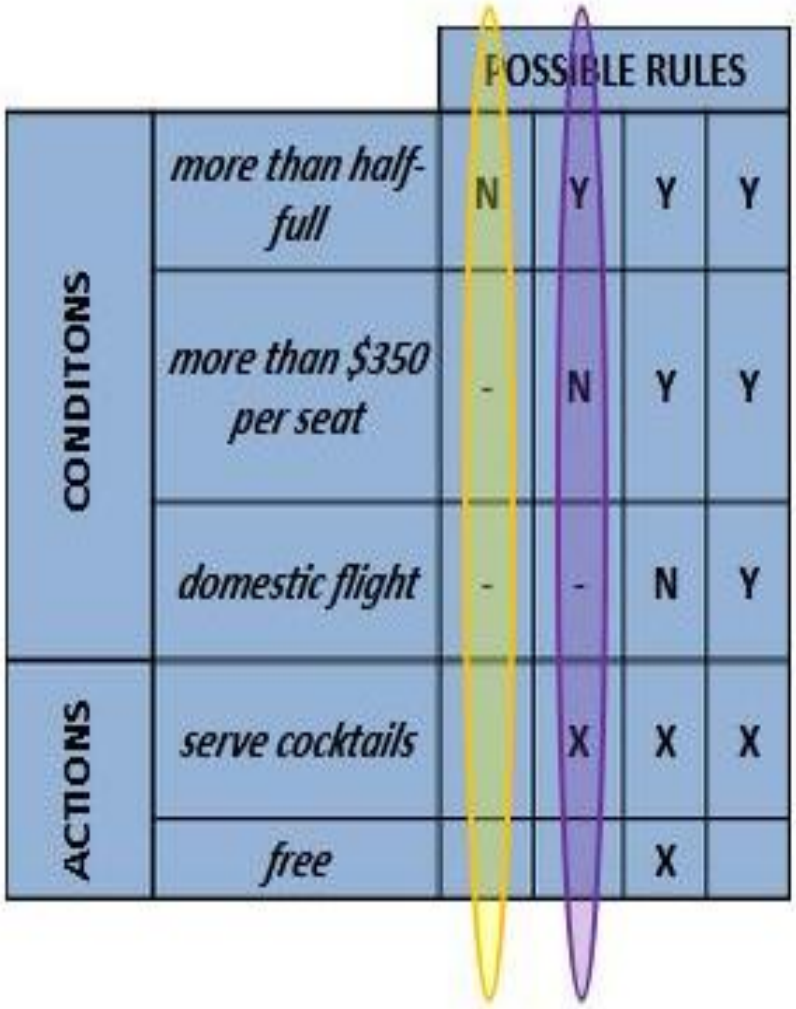
First we combine the yellow ones nullifying the condition.

Then the red ones.

Notice that yellow and red columns are identical but by one condition.

So, we combine them.

Reduce the table by eliminating redundant columns.



		POSSIBLE RULES			
CONDITONS	*more than half-full*	N	Y	Y	Y
	more than $350 per seat	-	N	Y	Y
	domestic flight	-	-	N	Y
ACTIONS	*serve cocktails*		X	X	X
	free			X	

First we combine the yellow ones nullifying the condition.

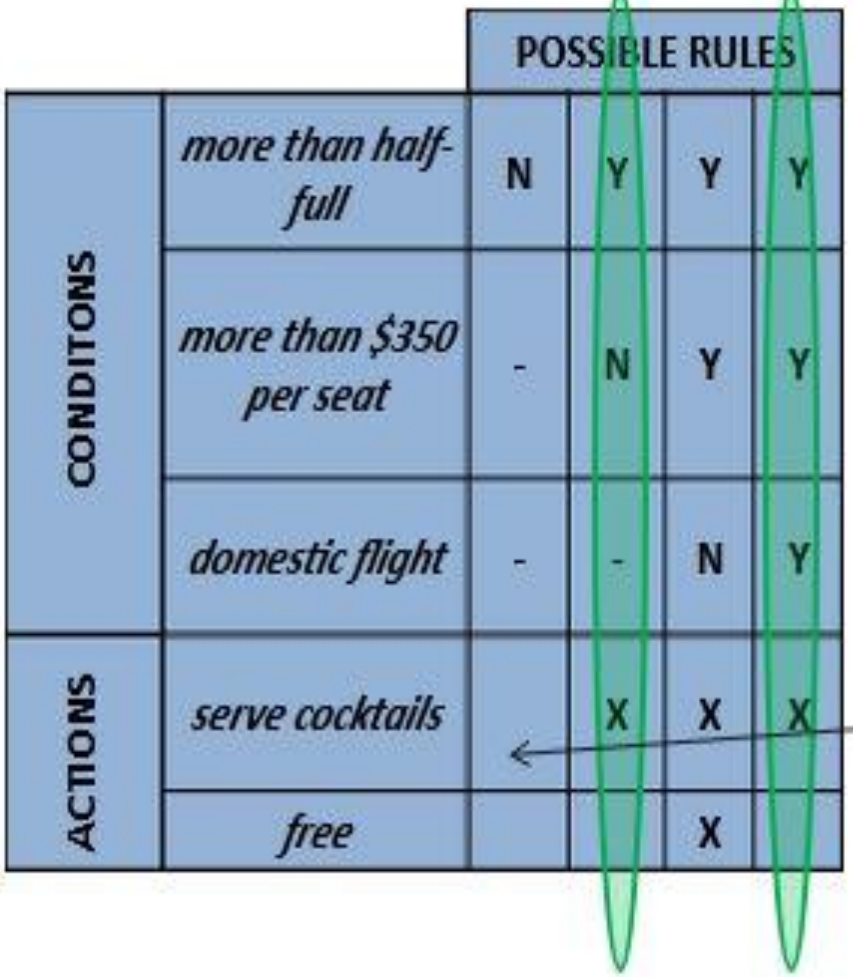
Then the red ones.

Notice that yellow and red columns are identical but by one condition.

So, we combine them.

Then we combine the violet colored ones.

Reduce the table by eliminating redundant columns.



		POSSIBLE RULES			
CONDITONS	*more than half-full*	N	Y	Y	Y
	more than $350 per seat	-	N	Y	Y
	domestic flight	-	-	N	Y
ACTIONS	*serve cocktails*		X	X	X
	free			X	

Notice that even when we observe that the green columns are identical except for one condition we do not combine them:

A "NULLIFIED" condition is not the same as a valued one.

What about this rule? Have we over looked something?

Reduce the table by eliminating redundant columns.

Final Solution

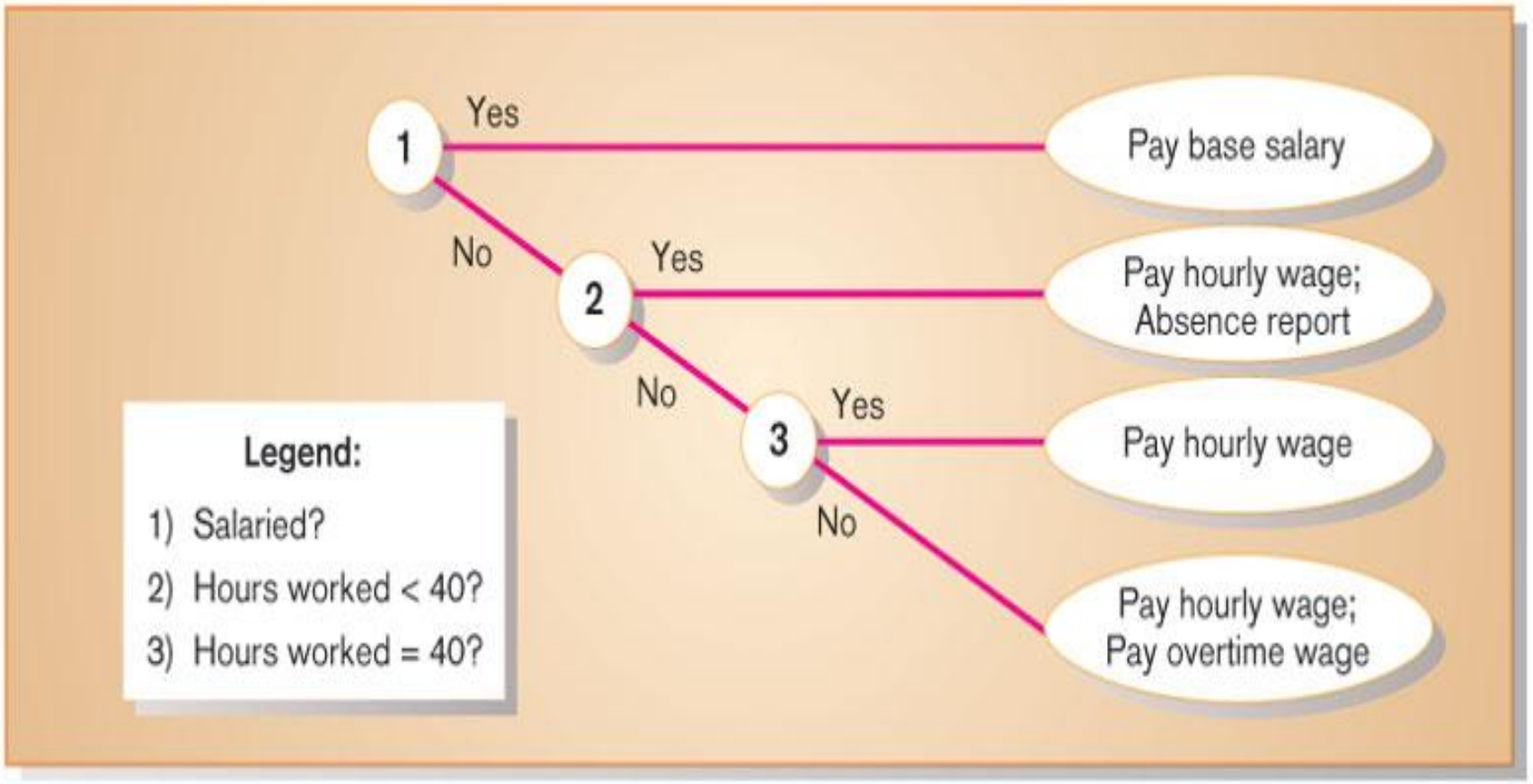
		Rules			
CONDITIONS	*more than half-full*	N	Y	Y	Y
	more than $350 per seat	-	N	Y	Y
	domestic flight	-	-	N	Y
ACTIONS	*serve cocktails*		X	X	X
	free			X	

Modeling Logic with Decision Trees

- A **decision tree** is a graphical representation of a decision situation
- Decision situation points (**nodes**) are connected together by **arcs** and terminate in **ovals**
- Main components
  - **Decision points** represented by **nodes**
  - **Actions** represented by **ovals**
  - Particular choices from a decision point represented by **arcs**
- To **read** a decision tree – begin at root node on far **left**
- Each **node** is numbered and each number corresponds to a **choice**
- Choices are spelled out in a **legend**
- From each node there are *at least two paths* leading to next step – another *decision point* or an *action*
- All possible **actions** are listed on the far **right** in **leaf nodes**
- Each **rule** is represented by tracing a series of *paths* from **root** node to the **next** node and so on until an **action** *oval* is reached

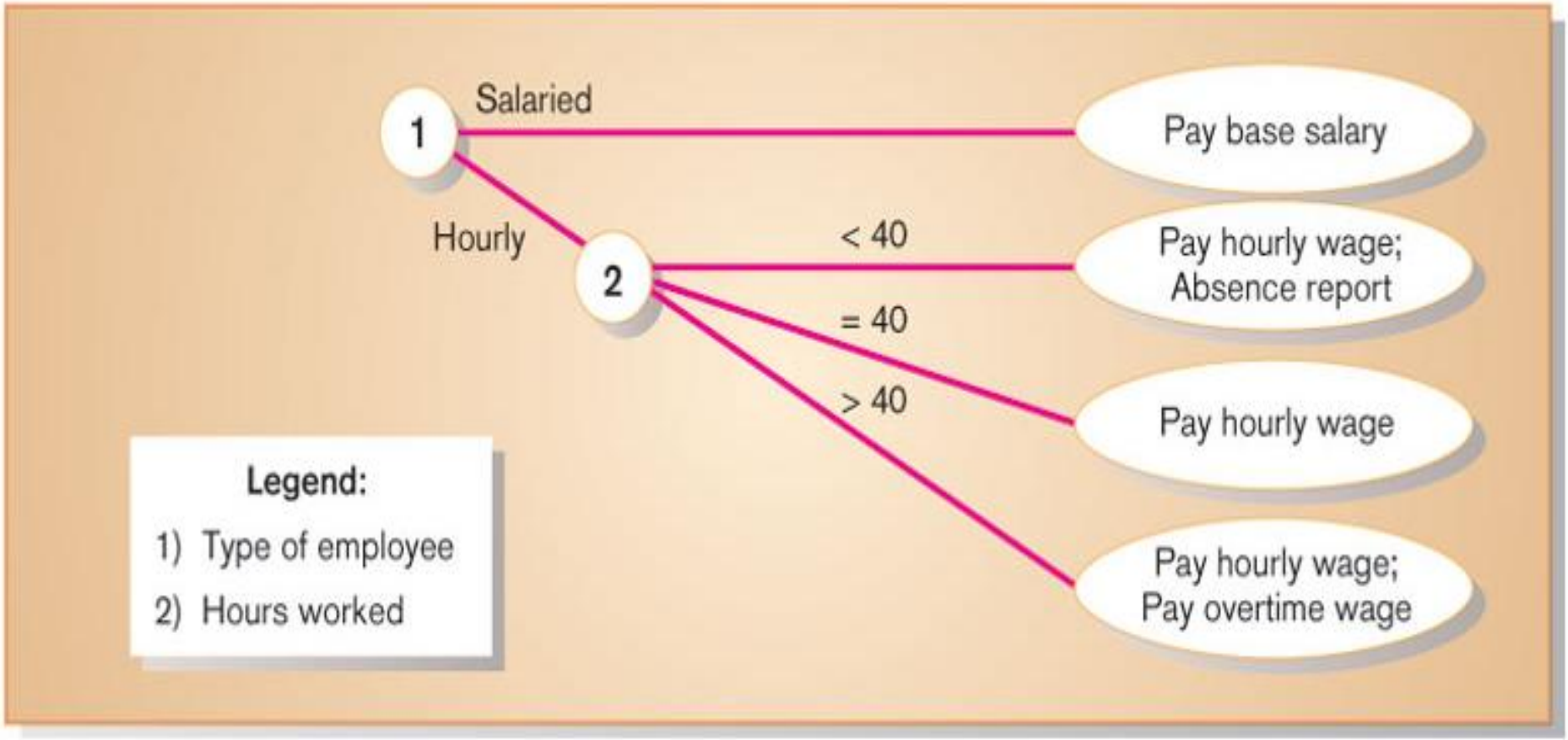
Decision tree representation of salary decision

Decision tree representation of the decision logic in the decision tables in Figures 8-4 and 8-5, with only two choices per decision point



Alternative decision tree representation of salary decision

Decision tree representation of the decision logic in the decision tables in Figure 8-4 and 8-5, with multiple choices per decision point



Deciding Among Structured English, Decision Tables, and Decision Trees

Criteria	Structured English	Decision Tables	Decision Trees
Determining Conditions and Actions	Second Best	Third Best	Best
Transforming Conditions and Actions into Sequence	Best	Third Best	Best
Checking Consistency and Completeness	Third Best	Best	Best

Deciding Between Decision Tables and Decision Trees

Criteria	Decision Tables	Decision Trees
Describe complex logic	Best	Worst
Describe simple rules	Worst	Best
Making decisions	Worst	Best
More compact	Best	Worst
Easier to manipulate	Best	Worst

Example of Using a Decision Tree or Table to Capture Complex Business Logic

Consider the following excerpt from an actual business document:

If the customer account is billed using a fixed rate method, a minimum monthly charge is assessed for consumption of less than 100 kwh. Otherwise, apply a schedule A rate structure. However, if the account is billed using a variable rate method, a schedule A rate structure will apply to consumption below 100 kwh, with additional consumption billed according to schedule B.

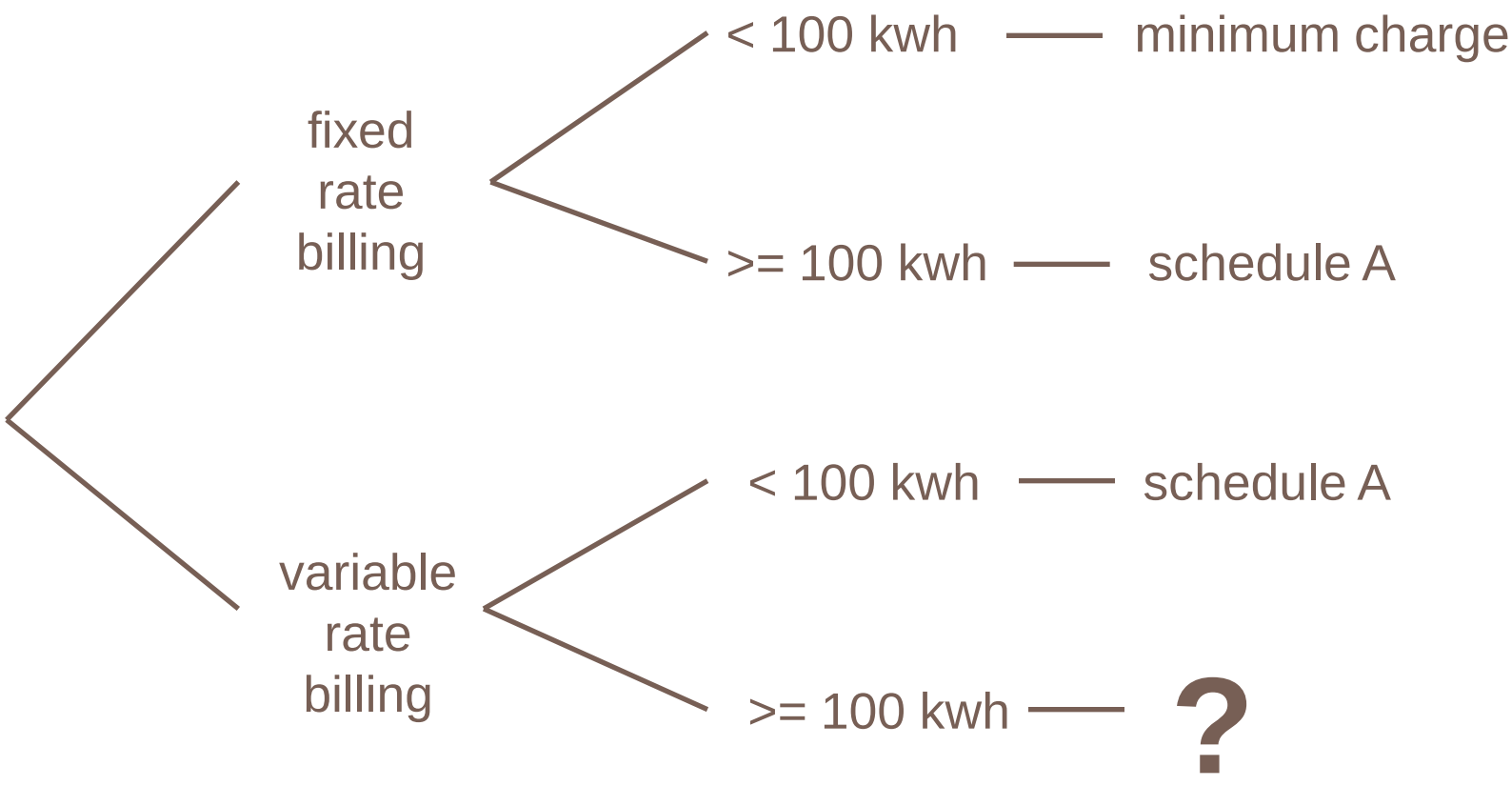
Example of Using a Decision Tree or Table to Capture Complex Business Logic

Consider the following excerpt from an actual business document:

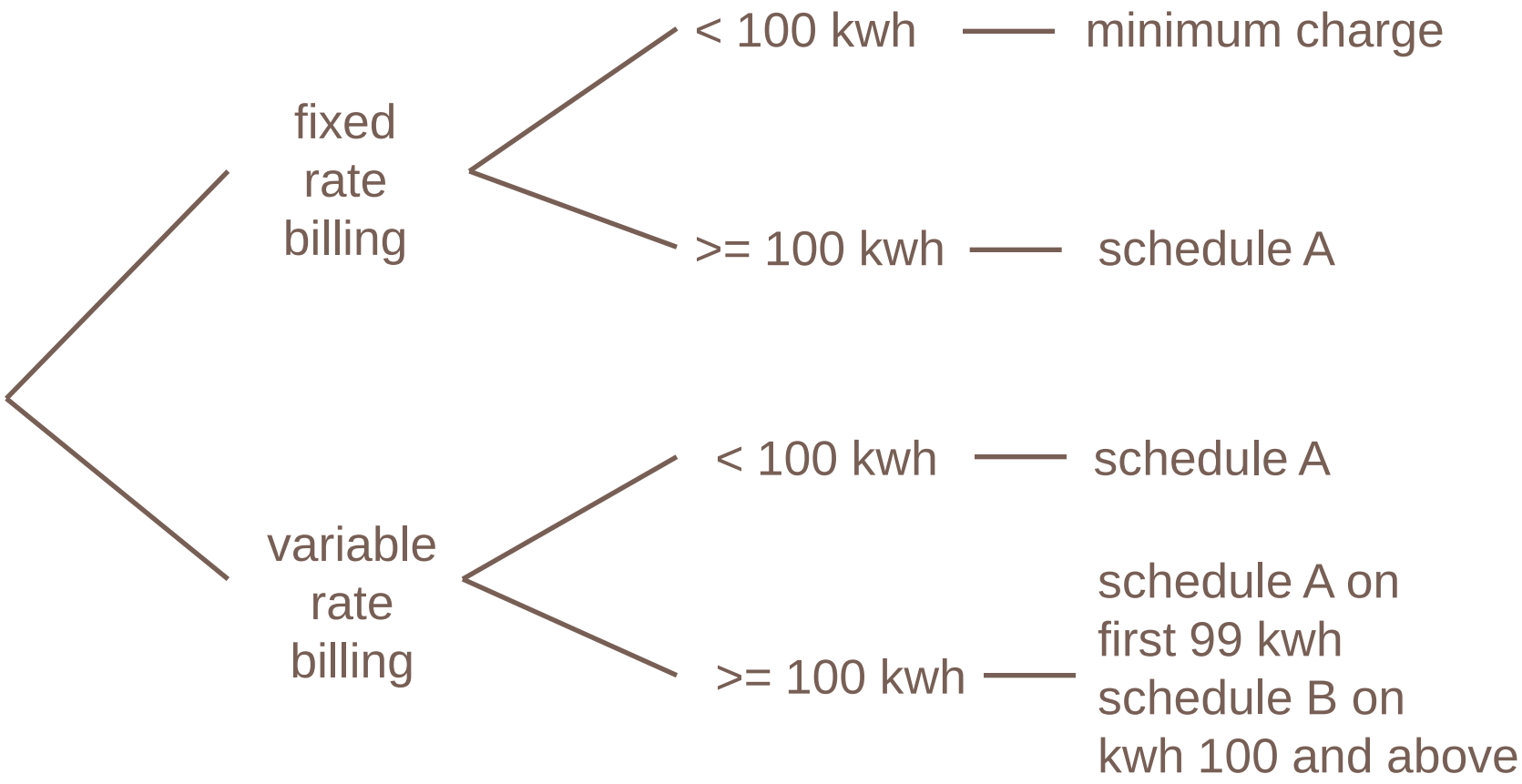
- If the customer account is billed using a **fixed rate method**, a ***minimum monthly charge*** is assessed for consumption of less than 100 kwh. Otherwise, **apply a schedule A rate structure**.
- However, if the account is billed using a **variable rate method**, a ***schedule A*** rate structure will apply to consumption **below 100 kwh**,
- with additional consumption billed according to **schedule B.**

STUDENT TIME

Decision Tree for this Example



Decision Tree for this Example



Decision Table for Example – Version 1

	Rules					
Conditions	1	2	3	4	5	
Fixed rate acct	T	T	F	F	F	
Variable rate acct	F	F	T	T	F	
Consumption < 100 kwh	T	F	T	F		
Consumption >= 100 kwh	F	T	F	T		
Actions						
Minimum charge	X					
Schedule A		X	X			
Schedule A on first 99 kwh, Schedule B on kwh 100 +				X		

Is this a valid business case? Did we miss something?

Decision Table for Example – Version 2

	Rules			
Conditions	1	2	3	4
Account type	fixed	fixed	variable	variable
Consumption	< 100	>=100	<100	>= 100
Actions				
Minimum charge	X			
Schedule A		X	X	
Schedule A on first 99 kwh, Schedule B on kwh 100 +				X