Decision Modeling: Good, Bad, Ugly

Jacob Feldman, PhD
OpenRules, Inc.
Chief Technology Officer

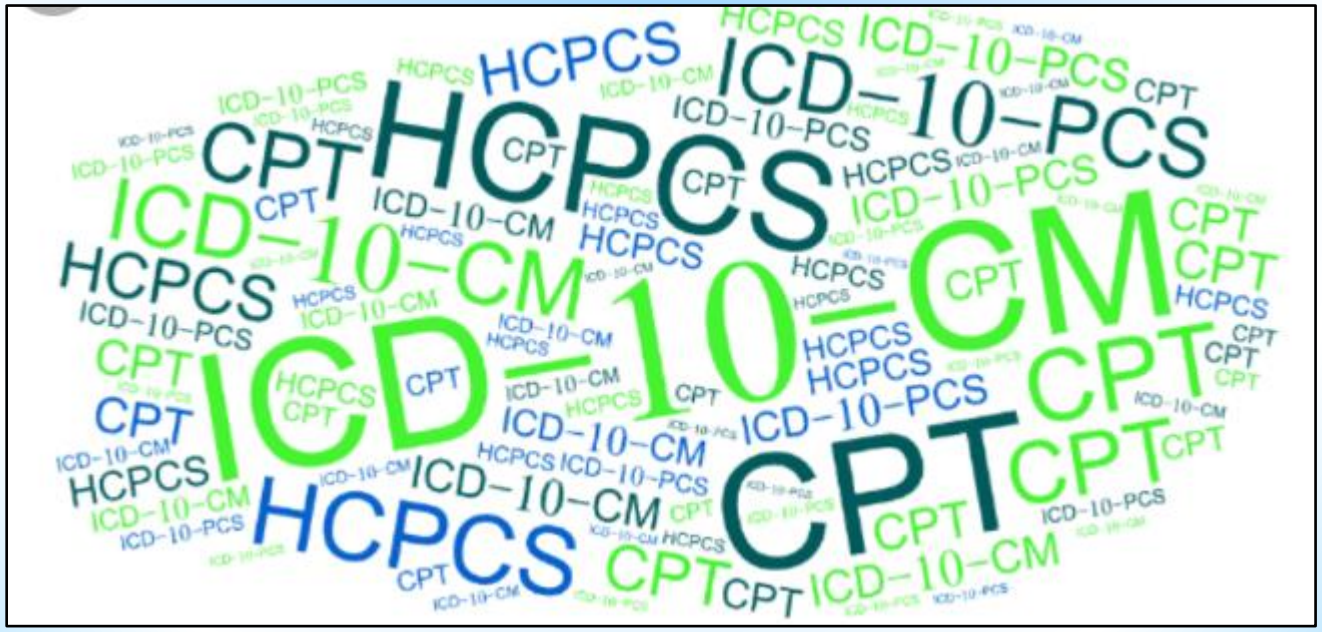
www.OpenRules.com

Decision Modeling: Theory and Practice

- **Commonly agreed design approaches:**
  - Orientation to Business Users (subject matter experts)
  - Top-down decision modeling
  - Low Code / No Code Decision Services
  - Support for *ongoing* improvements decision-making apps
- **This presentation will:**
  - Check general design principles vs real-world decision models
  - Discuss different implementations of the same decision model
    - Good
    - Bad
    - Ugly
    - A better one?

Selecting a Decision Model From Claim Processing Domain

- Highly popular domain where rule engines frequently applied
- Uses a lot of “Business Data” about compatible and incompatible diagnoses and claimed activities
- Real-world claim processing applications deal with very large and complex billing and coding lists and compliance rules



Building Decision Model for DMCommunity Challenge "Medical Claim Processing"

- **A simplified use case in DMCommunity.org Challenge published on May-2022**
  - Given a medical claim that contains multiple diagnoses, e.g., K75.1, A065.1, A48.5, C94.42
  - Our decision service is supposed to validate this claim against one large CSV file that consists ~70,000 pairs of incompatible diagnoses
- **We need to keep in mind that real-world claim processing applications deal with much more complex cases:**
  - complex compatibility and incompatibilities conditions with multiple columns and much larger CSV files with 500,000+ lines

International Statistical Classification of Diseases and Related Health Problems

```
Column 1,Column 2
A48.5,A05.1
K75.0,A06.4
K75.0,K83.09
K75.0,K75.1
G07,A06.6
G07,B43.1
...
```

Sample

- **Given Diagnoses**: R29.891, M43.6, F45.8
- **Pairs of diagnoses:**

R29.891, M43.6
R29.891,F45.8
M43.6, F45.8

Diagnosis 1	Diagnosis 2
R29.891	F45.8

- **Errors:**

R29.891 cannot be reported together with M43.6
R29.891 cannot be reported together with F45.8
M43.6 cannot be reported together with F45.8

ICD10Codes.csv

```
Column 1,Column 2
A48.5,A05.1
K75.0,A06.4
K75.0,K83.09
K75.0,K75.1
G07,A06.6
G07,B43.1
G07,A54.82
G07,A17.81
G07,A17.1
N51,A06.8
N51,B37.42
N51,A54.23
N51,A54.22
N51,A60.01
N51,A59.02
N51,A18.14
N48.1,A06.8
N48.1,N48.0
N48.1,B37.42
N48.1,A54.23
N48.1,A60.01
B60,A07.2
B60,A07.8
B60,A07.3
J98.11,A15
...
```

Problem Scope

- **Claim Validation Service:**
  - Receives a set of diagnosis codes {C_1,C_2,C_3,…}
  - Should validate all these codes against the large CSV file
  - Produce errors "Diagnosis Code [C_i] cannot be reported together with [C_j]" when:
    - C_i found in Column 1 and C_J found in Column 2 of the same row
    - C_i found in Column 2 and C_J found in Column 1 of the same row
  - Same diagnosis codes can be found in *both* columns
  - Do not produce duplicate errors like

    [E71.313] cannot be reported together with [E72.3]
    [E72.3] cannot be reported together with [E71.313]
- **How to build the corresponding Decision Model?**

Claim:

- Patient: …
- Diagnoses:
  - E71.313
  - E72.3
  - G07
- …

ICD10Codes.csv

```
Column 1,Column 2
A48.5,A05.1
K75.0,A06.4
K75.0,K83.09
K75.0,K75.1
G07,A06.6
G07,B43.1
G07,A54.82
G07,A17.81
G07,A17.1
N51,A06.8
N51,B37.42
N51,A54.23
N51,A54.22
N51,A60.01
N51,A59.02
N51,A18.14
N48.1,A06.8
N48.1,N48.0
N48.1,B37.42
N48.1,A54.23
N48.1,A60.01
B60,A07.2
B60,A07.8
B60,A07.3
J98.11,A15
...
```

Applying different decision modeling approaches

- **Top-Down Approach**
  - Usually works fine
  - I applied it initially
  - But it distracted me forcing to concentrate up-front on how to select different pairs of diagnoses
- **Bottom-Up Approach**
  - Let's assume that the pair {Diagnosis 1; Diagnosis 2} already selected
  - We need to look for these diagnoses in the CSV file using the following logic:

    IF (Diagnosis 1 found in the Column1 **AND** Diagnosis 2 found in the Column2)

    OR (Diagnosis 1 found in the Column2 **AND** Diagnosis 2 found in the Column1)

    THEN Report the error "Diagnosis 1 cannot be reported together Diagnosis 2"

Good (Enough) Solution

Search and Comparison Logic in OpenRules

- We need to search for Diagnosis 1 and Diagnosis 2 in the CSV file using the following logic:

  IF (Diagnosis 1 found in the Column1 **AND** Diagnosis 2 found in the Column2)

  OR (Diagnosis 1 found in the Column2 **AND** Diagnosis 2 found in the Column1)

  THEN Report the error "Diagnosis 1 cannot be reported together Diagnosis 2"

- It is easy to present this logic using the standard OpenRules decision table of the type "BigTable":

"BigTable" guarantees superfast search

This is a single-hit table. [ICD10Codes.csv] tells OpenRules to apply one of two rules below to every row in the CSV file

BigTable SearchCSV [ICD10Codes.csv]		
Condition	Condition	Action
Diagnosis 1	**Diagnosis 2**	**Errors**
=	=	+=
Column 1	Column 2	{{Diagnosis 1}} cannot be reported together with {{Diagnosis 2}}
Column 2	Column 1	

ICD10Codes.csv

```
Column 1,Column 2
A48.5,A05.1
K75.0,A06.4
K75.0,K83.09
```

How “BigTable” Works

- **BigTable** is an OpenRules extension of standard decision tables. We could use the keyword “DecisionTable” instead of “BigTable”. However, in some cases it may be 10-100 times slower. Why?
- **BigTable** uses a special execution algorithm based on *self-balancing binary search*. For large volumes of “business data” it **increases decision table the performance 10-100 times!**
- Additional capabilities:
  - We can use **BigTableMultiHit** to accumulate certain values while we navigate through the CSV file
  - You may save exact row numbers for which the rules were successfully executed
  - Instead of keeping “business data” in a separate CSV file you may move all data rows directly into the Excel-based decision table

Selecting Diagnosis Pairs

- So, now we know that the table "SearchCSV" will be good for search and comparison logic:

BigTable SearchCSV [ICD10Codes.csv]		
Condition	Condition	Action
Diagnosis 1	**Diagnosis 2**	**Errors**
=	=	+=
Column 1	Column 2	{{Diagnosis 1}} cannot be reported together with {{Diagnosis 2}}
Column 2	Column 1	

- **Next question**: How to invoke the table "SearchCSV" for different pairs of diagnoses reported in the claim?

Selecting Diagnosis Pairs: Java

- Being a Java developer, my first impulse was to implement this logic as a Java method directly in Excel:

Code IterateDiagnoses

```
String[] diagnoses = (String[]) decision.getObjects("Diagnoses");
for(int i=0; i< diagnoses.length-1; i++) {
    decision.setVarValue("Diagnosis 1", diagnoses[i]);
    for(int j=i+1; j< diagnoses.length; j++) {
      decision.setVarValue("Diagnosis 2", diagnoses[j]);
      decision.execute("SearchCSV");
    }
}
```

- People familiar with Java or C can quickly understand what I did here:
  - I used two for-loops iterating over the same array “diagnoses”
  - The second (nested) loop uses only those diagnoses which were not selected yet in the first loop
  - When the pair {Diagnose 1; Diagnose 2} is defined, I invoke “SearchCSV” by using OpenRules API call:

    decision.execute(“SearchCSV”);

Adding Glossary, Test Cases, and Executing Decision Model

- Glossary:

Glossary glossary			
Variable Name	**Business Concept**	**Attribute**	**Type**
Claim Id	Claim	id	String
Diagnoses		diagnoses	String[]
Errors		errors	String[]
Diagnosis 1	Intermediate	diagnosis1	String
Diagnosis 2		diagnosis2	String

- Test Cases:

DecisionTest testCases			
#	ActionDefine	ActionDefine	ActionExpect
Test	**Claim Id**	**Diagnoses**	**Errors**
1	A	D47.02 C94.32	D47.02 cannot be reported together with C94.32
2	B	E71.313 E72.3	E71.313 cannot be reported together with E72.3
3	C	R29.891 M43.6 F45.8	R29.891 cannot be reported together with M43.6 R29.891 cannot be reported together with F45.8 M43.6 cannot be reported together with F45.8
4	D	D75.81 C94.42	D75.81 cannot be reported together with C94.42
5	D	D75.81 C94.32	

- The decision model was correctly executed within milliseconds

Should we get rid of Java and if "Yes" then "How"?

- The working Java code is not changed frequently and can be used in production "as is".
- However, how about our orientation to business users not familiar with basic Java or C? They don't want to see any code.

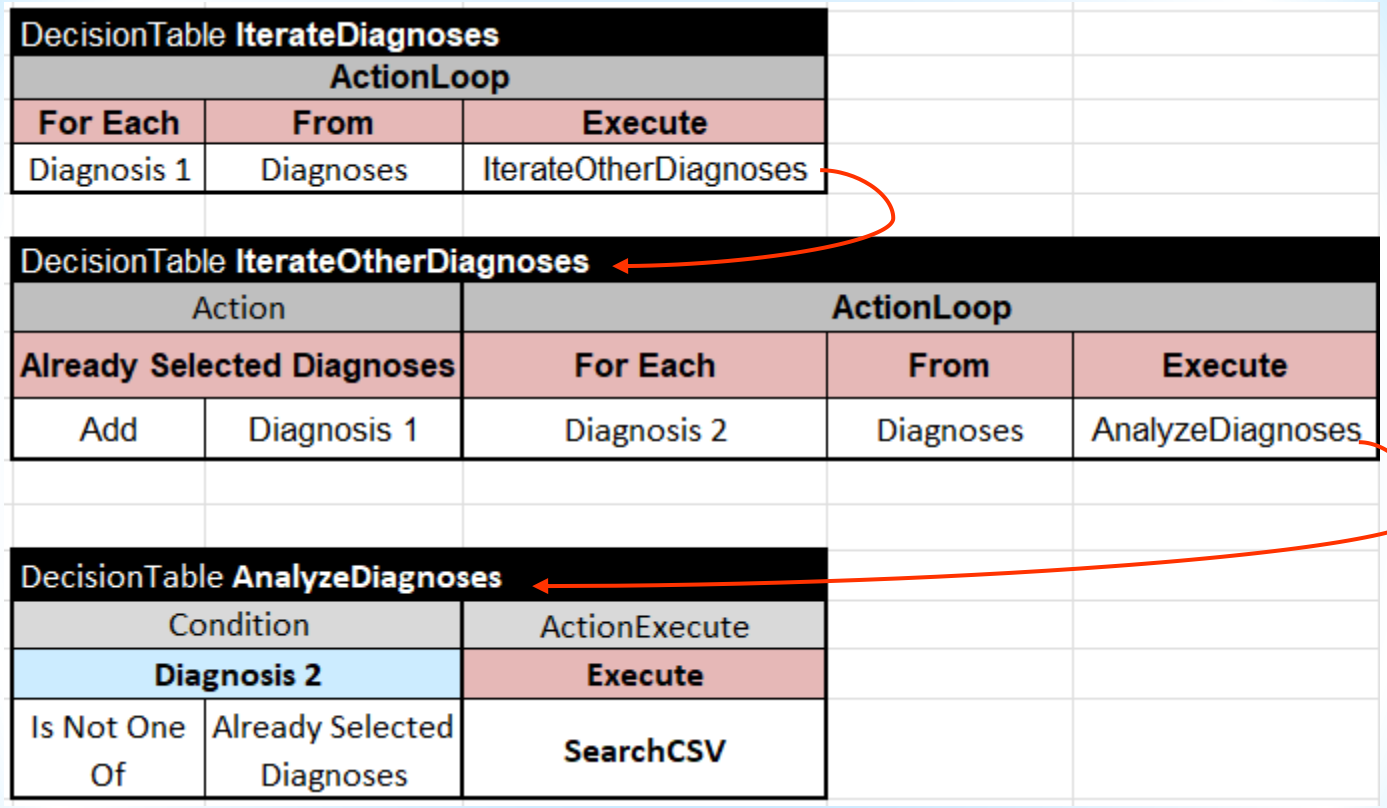
- I will show how we can implement similar nested loops not in Java but using regular decision tables with a special column "**ActionLoop"**
- For instance, let's consider the following action column inside a regular decision table:

ActionLoop		
For Each	**From**	**Execute**
Diagnosis	Diagnoses	DoSomething

  - It iterates through all diagnoses **from** the array "Diagnoses" and **for each** selected "Diagnosis" **executes** the decision table "DoSomething"
  - Here "Diagnoses" and "Diagnosis" are regular decision variables, and the decision table "DoSomething" can do something with the current "Diagnosis"

Selecting Diagnosis Pairs: Without Java

- Here our Java loops replaced with the following decision tables:



DecisionTable **IterateDiagnoses**

ActionLoop		
For Each	**From**	**Execute**
Diagnosis 1	Diagnoses	IterateOtherDiagnoses

DecisionTable **IterateOtherDiagnoses**

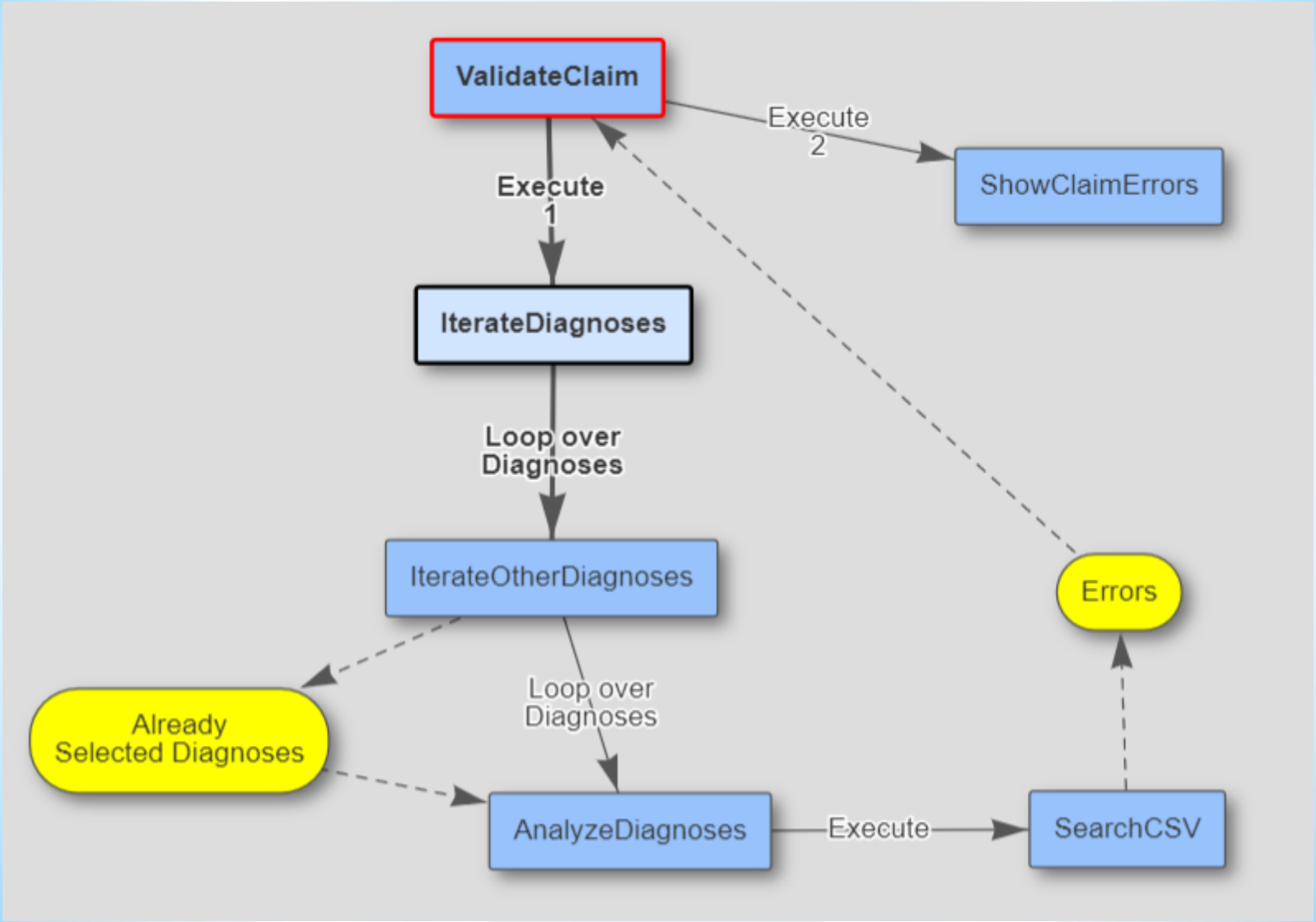
Action		ActionLoop		
Already Selected Diagnoses		**For Each**	**From**	**Execute**
Add	Diagnosis 1	Diagnosis 2	Diagnoses	AnalyzeDiagnoses

DecisionTable **AnalyzeDiagnoses**

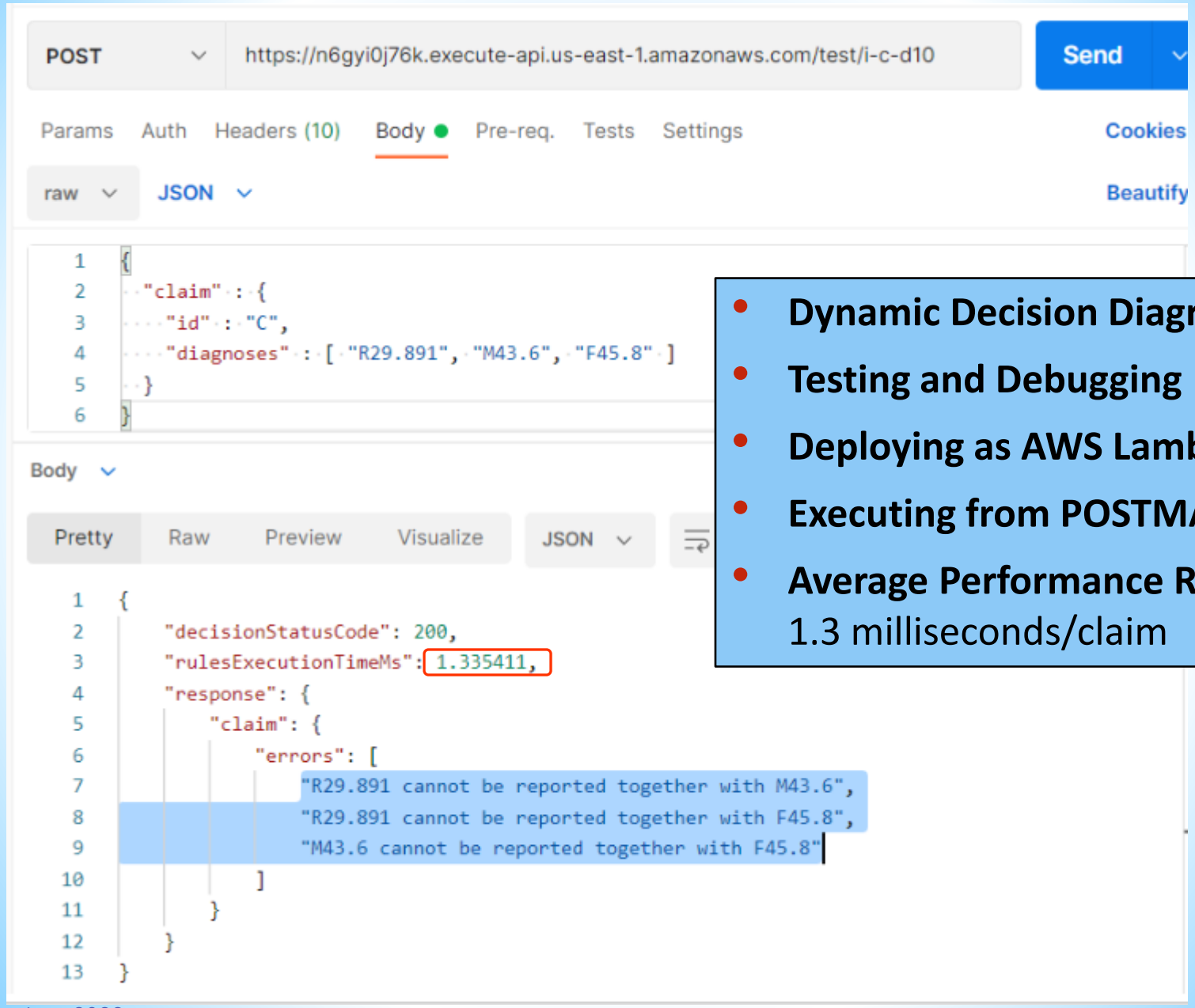
Condition		ActionExecute
Diagnosis 2		**Execute**
Is Not One Of	Already Selected Diagnoses	**SearchCSV**

- So, to avoid the same diagnoses inside the nested loop I added an intermediate array "Already Selected Diagnoses". Why?
- Because "ActionLoop" does not support indexes (we thought it would be too much for business users)

Decision Diagram



Live Demo of this Decision Model



- **Dynamic Decision Diagrams**
- **Testing and Debugging**
- **Deploying as AWS Lambda**
- **Executing from POSTMAN**
- **Average Performance Results:** 1.3 milliseconds/claim

Decision Tables vs Java

---------- Diagnosis Pair Selection Logic ----------

Decision Tables

DecisionTable **IterateDiagnoses**

ActionLoop		
For Each	**From**	**Execute**
Diagnosis 1	Diagnoses	IterateOtherDiagnoses

DecisionTable **IterateOtherDiagnoses**

Action	ActionLoop		
Already Selected Diagnoses	**For Each**	**From**	**Execute**
Add	Diagnosis 1	Diagnosis 2	Diagnoses	AnalyzeDiagnoses

DecisionTable **AnalyzeDiagnoses**

Condition		ActionExecute
Diagnosis 2		**Execute**
Is Not One Of	Already Selected Diagnoses	**SearchCSV**

Java

Code IterateDiagnoses

```
String[] diagnoses = (String[]) decision.getObjects("Diagnoses");
for(int i=0; i< diagnoses.length-1; i++) {
    decision.setVarValue("Diagnosis 1", diagnoses[i]);
    for(int j=i+1; j< diagnoses.length; j++) {
       decision.setVarValue("Diagnosis 2", diagnoses[j]);
       decision.execute("SearchCSV");
    }
}
```

---------- Search and Comparison Logic ----------

BigTable SearchCSV [ICD10Codes.csv]

Condition	Condition	Action
Diagnosis 1	**Diagnosis 2**	**Errors**
=	=	+=
Column 1	Column 2	{{Diagnosis 1}} cannot be reported together with {{Diagnosis 2}}
Column 2	Column 1	

Decision Tables vs Java

Decision Table with CSV

BigTable SearchCSV [ICD10Codes.csv]		
Condition	Condition	Action
Diagnosis 1	**Diagnosis 2**	**Errors**
=	=	+=
Column 1	Column 2	{{Diagnosis 1}} cannot be reported together with {{Diagnosis 2}}
Column 2	Column 1	

Java (courtesy of Dr. Bob Moore)

```
public ArrayList<String> checkCase(int caseNum, String[] caseData, Set<String> forbiddenCombos) {
	ArrayList<String> clashes = new ArrayList<String>();
	int length = caseData.length;
	// note if length <= 1, the loop does not execute
	for (int first = 0; first < length - 1; first++) {
		String firstStr = caseData[first];
		for (int second = first + 1; second < length; second++) {
			String secondStr = caseData[second];
			String key = firstStr + "|" + secondStr;
			if (forbiddenCombos.contains(key)) {
				clashes.add(String.format(
					"Case Record %5d: Diagnosis Code %7s cannot be reported together with %7s",
						caseNum, firstStr, secondStr));
			}
		}
	}
	return clashes;
}
```

Decision Tables vs Java

- For simple cases (like in this challenge) both Decision Tables and Java may provide relatively good and highly efficient solutions
- In real-world, we deal with much more complex conditions, e.g.

Condition	Condition	Condition	Condition
Activity Code	**Activity Code**	**Diagnosis Code**	**Diagnosis Code**
>=	<=	>=	<=
Activity Code Min	Activity Code Max	Diagnosis Code Min	Diagnosis Code Max

- They may include special indicators that allow certain incompatibilities to be ignored, they may deal with time intervals, several CSV files, and much more
- In these cases, simple and highly efficient Java Set's method "***contains***" would not work anymore while decision tables with CVS files will continue effectively handle the most complex logic.

Bad Solution

Falling into the Trap by bringing a tool capabilities to the model and not vice versa

- My first attempt to solve the challenge failed badly. Here is why.
- I knew that OpenRules provides a nice operator "**Intersect With**" which allows us to check if two arrays include the same elements. So, I my first impulse was to use this decision table to analyze each Diagnosis:

DecisionTable DiagnosisMismatches					
Condition	Condition	Condition		Action	
Found in Column 2	**Found in Column 1**	**Other Diagnoses**		**Errors**	
	TRUE	Intersect With	Matches in Column 2	Add	{{Diagnosis}} cannot be reported together with {{Other Diagnoses}}
TRUE		Intersect With	Matches in Column 1		

- It would avoid using nested loops, but it required creating array of "Other Diagnoses" for each diagnosis. This has to be done in Java.
- Additionally, for each diagnosis I still needed to search the CSV file to determine two arrays: "Matches in Column 1" and "Matches in Column 2"
- And when I did it, my model still produced duplicated errors!

Ugly Solution

Ugly Solution

- If my initial bad solution was not ugly enough, I'd share another bad one from our real-world experience
- In this case business analysts asked their IT colleagues to help them to modify their business logic to avoid creation of similar duplication errors
- And IT did "help" business analysts:
  - They wrote a "post-processor" in Java that took the array of all produced errors and removed duplications
  - Of course, the logic that defined "duplications" was hardcoded in Java!
- Hopefully, I don't need to comment why this solution is ugly.

DECISION CAMP

Better Solution?

Summary

- **Commonly accepted theoretical approaches don't always work in practical decision modeling**
- **We considered different implementations of the same decision model**
  - Good (enough)
  - Bad
  - Ugly
- **Challenge: Build a better solution**