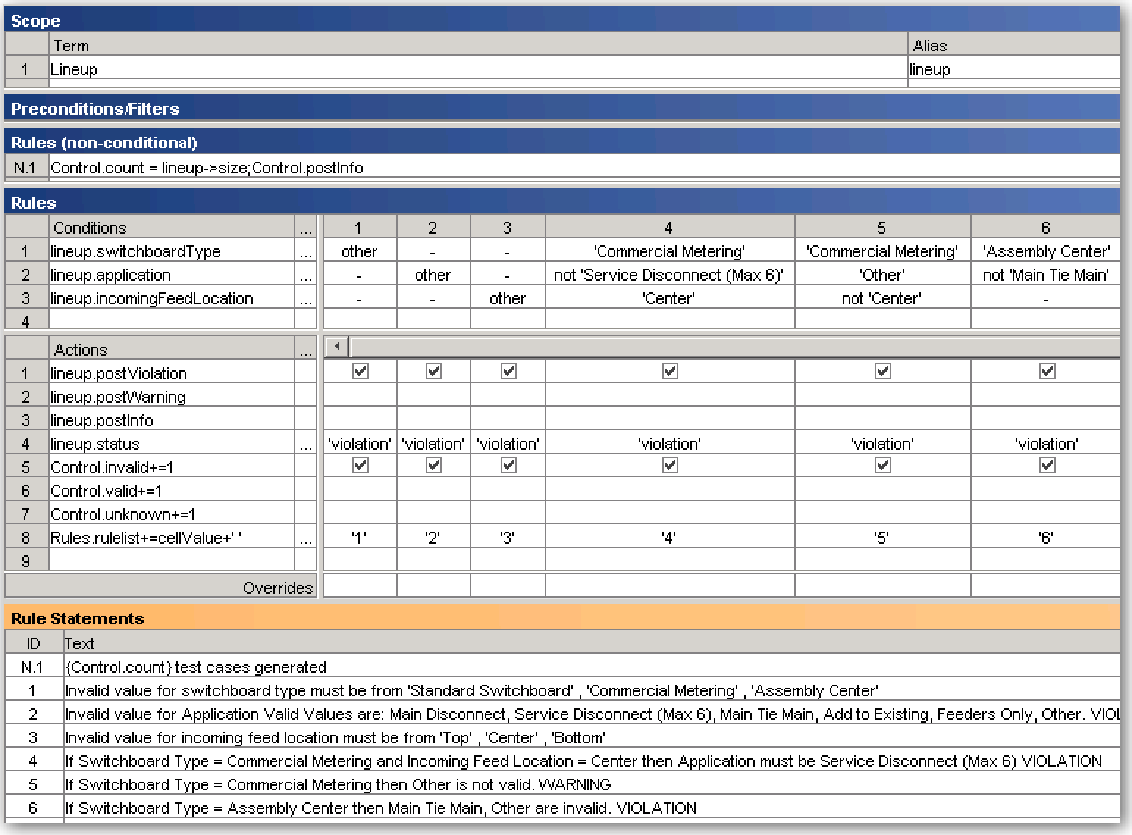
Automatic Generation of Test Data

This rule model shows how you can automatically generate all combinations of test data.

First, let's look at the business rules. In this example the business decision that is being made is to determine if a particular configuration (of electrical equipment) is valid.

Here's the rulesheet



Scope

	Term	Alias
1	Lineup	lineup

Preconditions/Filters

Rules (non-conditional)

N.1	Control.count = lineup->size;Control.postInfo

Rules

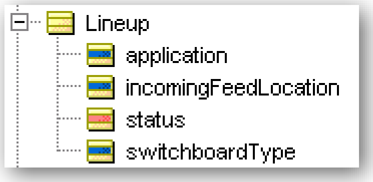
	Conditions	...	1	2	3	4	5	6
1	lineup.switchboardType	...	other	-	-	'Commercial Metering'	'Commercial Metering'	'Assembly Center'
2	lineup.application	...	-	other	-	not 'Service Disconnect (Max 6)'	'Other'	not 'Main Tie Main'
3	lineup.incomingFeedLocation	...	-	-	other	'Center'	not 'Center'	-
4								

	Actions	...						
1	lineup.postViolation		✓	✓	✓	✓	✓	✓
2	lineup.postWarning							
3	lineup.postInfo							
4	lineup.status	...	'violation'	'violation'	'violation'	'violation'	'violation'	'violation'
5	Control.invalid+=1		✓	✓	✓	✓	✓	✓
6	Control.valid+=1							
7	Control.unknown+=1							
8	Rules.rulelist+=cellValue+' '	...	'1'	'2'	'3'	'4'	'5'	'6'
9								
	Overrides							

Rule Statements

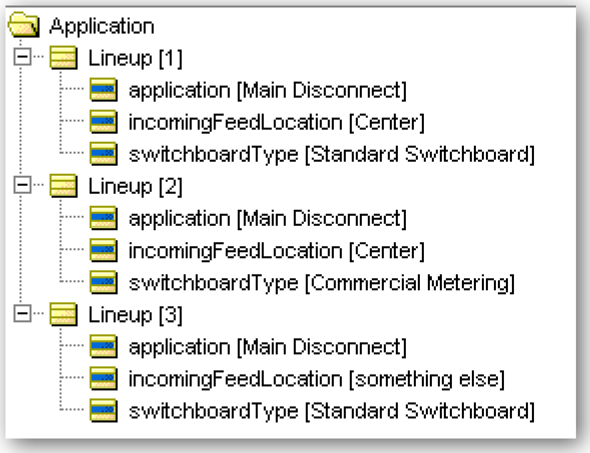
ID	Text
N.1	{Control.count} test cases generated
1	Invalid value for switchboard type must be from 'Standard Switchboard' , 'Commercial Metering' , 'Assembly Center'
2	Invalid value for Application Valid Values are: Main Disconnect, Service Disconnect (Max 6), Main Tie Main, Add to Existing, Feeders Only, Other. VIOL
3	Invalid value for incoming feed location must be from 'Top' , 'Center' , 'Bottom'
4	If Switchboard Type = Commercial Metering and Incoming Feed Location = Center then Application must be Service Disconnect (Max 6) VIOLATION
5	If Switchboard Type = Commercial Metering then Other is not valid. WARNING
6	If Switchboard Type = Assembly Center then Main Tie Main, Other are invalid. VIOLATION

The vocabulary that supports this is:



- Lineup
  - application
  - incomingFeedLocation
  - status
  - switchboardType

And if we created test data by hand we'd have something like this:

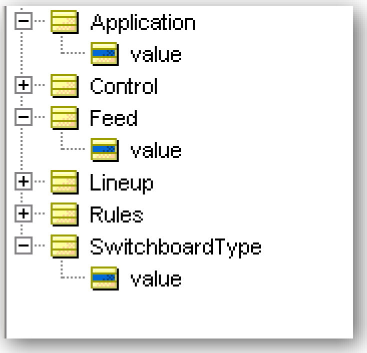

Application
Lineup [1]
application [Main Disconnect]
incomingFeedLocation [Center]
switchboardType [Standard Switchboard]
Lineup [2]
application [Main Disconnect]
incomingFeedLocation [Center]
switchboardType [Commercial Metering]
Lineup [3]
application [Main Disconnect]
incomingFeedLocation [something else]
switchboardType [Standard Switchboard]

Since there are several possible values for each of the attributes there may be a large number of possible test cases and creating them all by hand may be lengthy.

Fortunately Corticon can do this automatically.

In order to do this we first need some additional business objects. One for each of the attributes in the lineup object (application, incomingFeedLocation and switchboardType:

You can see them here:



Notice that each has just one attribute called “value”.

If you look in the vocabulary editor you can see that they each have a defined set of possible values. Here are the values for Application:

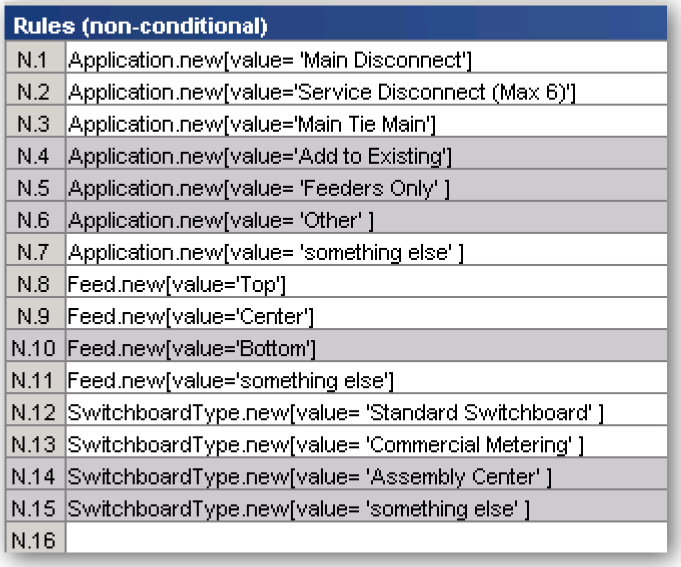
{ 'Main Disconnect' , 'Service Disconnect (Max 6)' , 'Main Tie Main' , 'Add to Existing' , 'Feeders Only' , 'Other' }

And for Feed: { 'Top' , 'Center' , 'Bottom', other }

And for SwitchboardType: { 'Standard Switchboard' , 'Commercial Metering' , 'Assembly Center' , other }

The total number of possible distinct combinations is 7 * 4 * 4 = 112

Now examine the rulesheet called "PossibleValues"

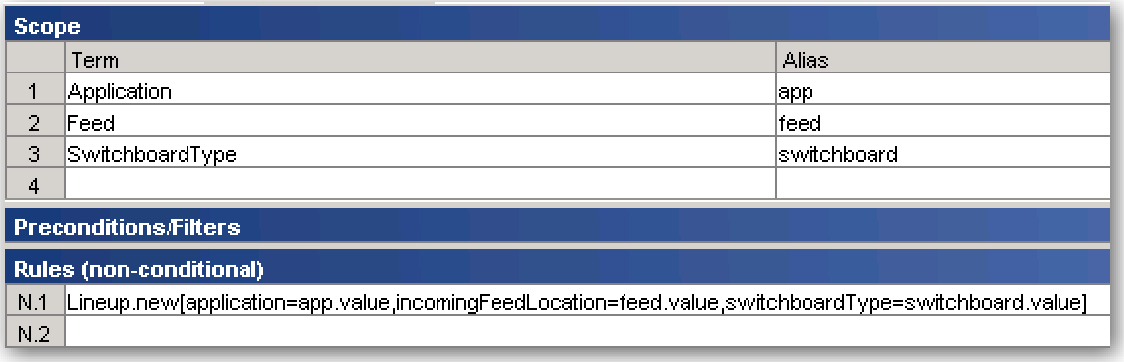


Rules (non-conditional)	
N.1	Application.new[value= 'Main Disconnect']
N.2	Application.new[value='Service Disconnect (Max 6)']
N.3	Application.new[value='Main Tie Main']
N.4	Application.new[value='Add to Existing']
N.5	Application.new[value= 'Feeders Only']
N.6	Application.new[value= 'Other']
N.7	Application.new[value= 'something else']
N.8	Feed.new[value='Top']
N.9	Feed.new[value='Center']
N.10	Feed.new[value='Bottom']
N.11	Feed.new[value='something else']
N.12	SwitchboardType.new[value= 'Standard Switchboard']
N.13	SwitchboardType.new[value= 'Commercial Metering']
N.14	SwitchboardType.new[value= 'Assembly Center']
N.15	SwitchboardType.new[value= 'something else']
N.16	

Using the **new** operator it creates instances of these business objects each with one of the various possible values.

Notice that some of they are greyed out (this is deliberately to ensure that we don't have a full set of test data)

Now look at the rulesheet called "GenerateLineups".



Scope

	Term	Alias
1	Application	app
2	Feed	feed
3	SwitchboardType	switchboard
4		

Preconditions/Filters

Rules (non-conditional)

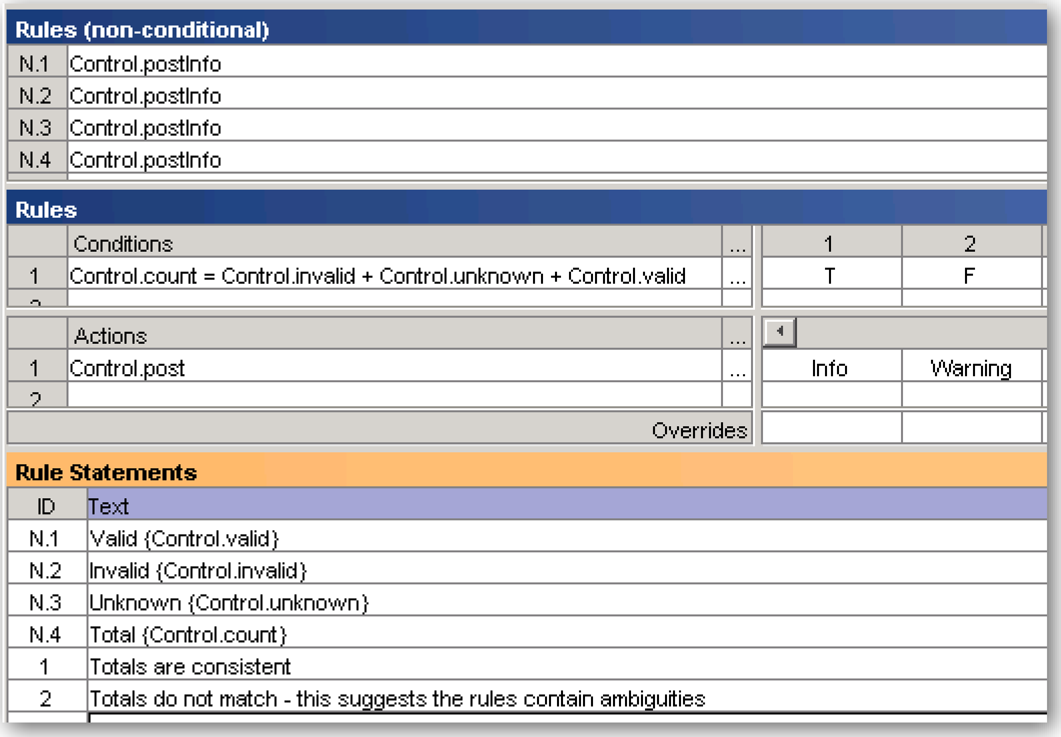
N.1	Lineup.new[application=app.value,incomingFeedLocation=feed.value,switchboardType=switchboard.value]
N.2	

This seemingly simple rulesheet will generate every possible combination of data values for the Lineup business object (based on the possible values we specified). This is how the scope section works by successively binding the alias to all instances of that business object that are in memory (in this case created by the **PossibleValues** rulesheet).

You will meet this same technique when you examine how Corticon can solve constraint based problems such as generating a magic square or solving a Sudoku puzzle.

All of these Lineups now get processed by the ValidateLineup rulesheet (the primary business rules which you already saw).

And then we have a sheet that summarizes the results



Rules (non-conditional)

N.1	Control.postInfo
N.2	Control.postInfo
N.3	Control.postInfo
N.4	Control.postInfo

Rules

	Conditions	...	1	2
1	Control.count = Control.invalid + Control.unknown + Control.valid	...	T	F
2				

	Actions	...		
1	Control.post	...	Info	Warning
2				
	Overrides			

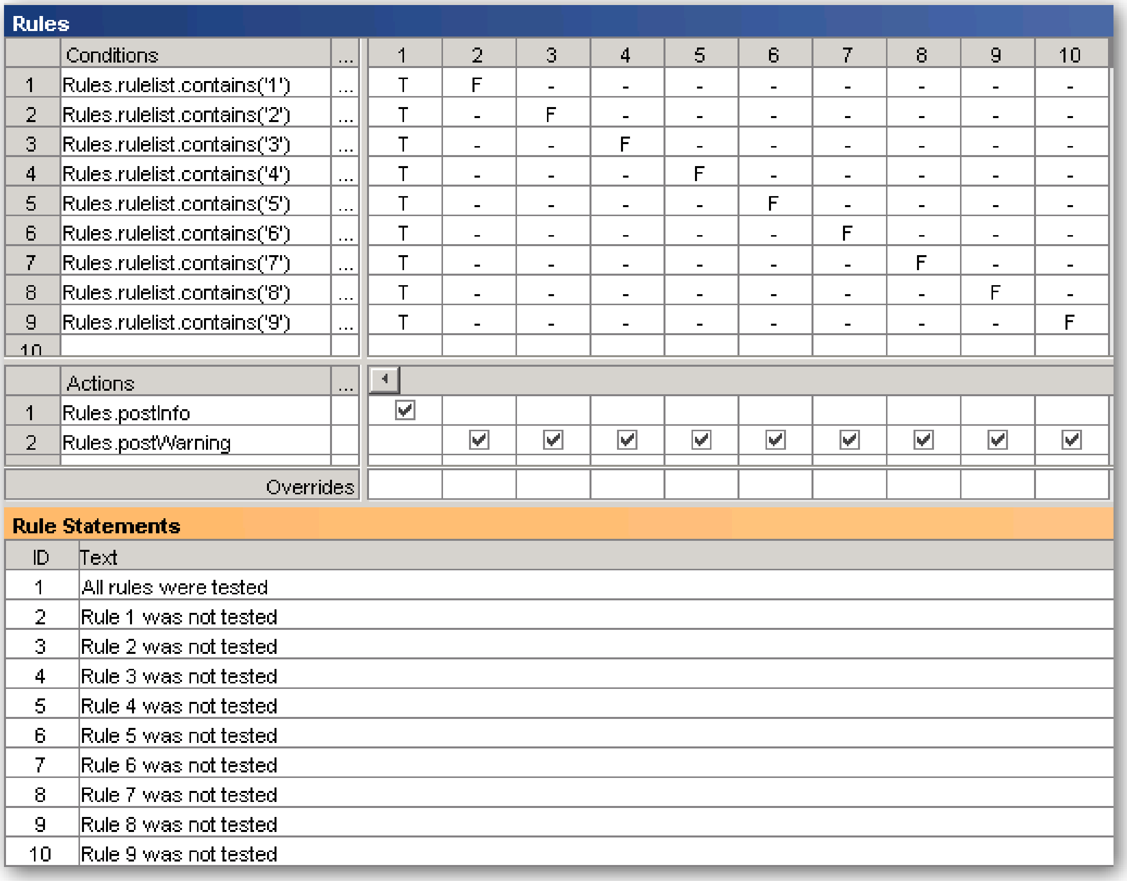
Rule Statements

ID	Text
N.1	Valid {Control.valid}
N.2	Invalid {Control.invalid}
N.3	Unknown {Control.unknown}
N.4	Total {Control.count}
1	Totals are consistent
2	Totals do not match - this suggests the rules contain ambiguities

This just looks at the various counters we used during execution of the business rules.

Note that these counters are in the single business object called Control and the posting is done against this object to ensure that there is only one set of totals. If you did the posting against the Lineup object you would get the totals repeated many times (once per lineup)

Finally we have this rulesheet to identify any rules that did not fire as the result of the test data – and since we deliberately omitted some test values we should find that some rules don't fire



Rules

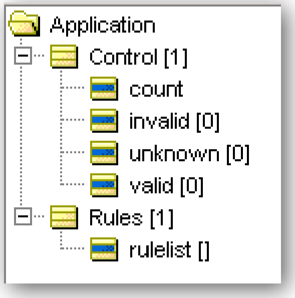
	Conditions	...	1	2	3	4	5	6	7	8	9	10
1	Rules.rulelist.contains('1')	...	T	F	-	-	-	-	-	-	-	-
2	Rules.rulelist.contains('2')	...	T	-	F	-	-	-	-	-	-	-
3	Rules.rulelist.contains('3')	...	T	-	-	F	-	-	-	-	-	-
4	Rules.rulelist.contains('4')	...	T	-	-	-	F	-	-	-	-	-
5	Rules.rulelist.contains('5')	...	T	-	-	-	-	F	-	-	-	-
6	Rules.rulelist.contains('6')	...	T	-	-	-	-	-	F	-	-	-
7	Rules.rulelist.contains('7')	...	T	-	-	-	-	-	-	F	-	-
8	Rules.rulelist.contains('8')	...	T	-	-	-	-	-	-	-	F	-
9	Rules.rulelist.contains('9')	...	T	-	-	-	-	-	-	-	-	F
10												

	Actions	...	1	2	3	4	5	6	7	8	9	10
1	Rules.postInfo		✓									
2	Rules.postWarning			✓	✓	✓	✓	✓	✓	✓	✓	✓
	Overrides											

Rule Statements

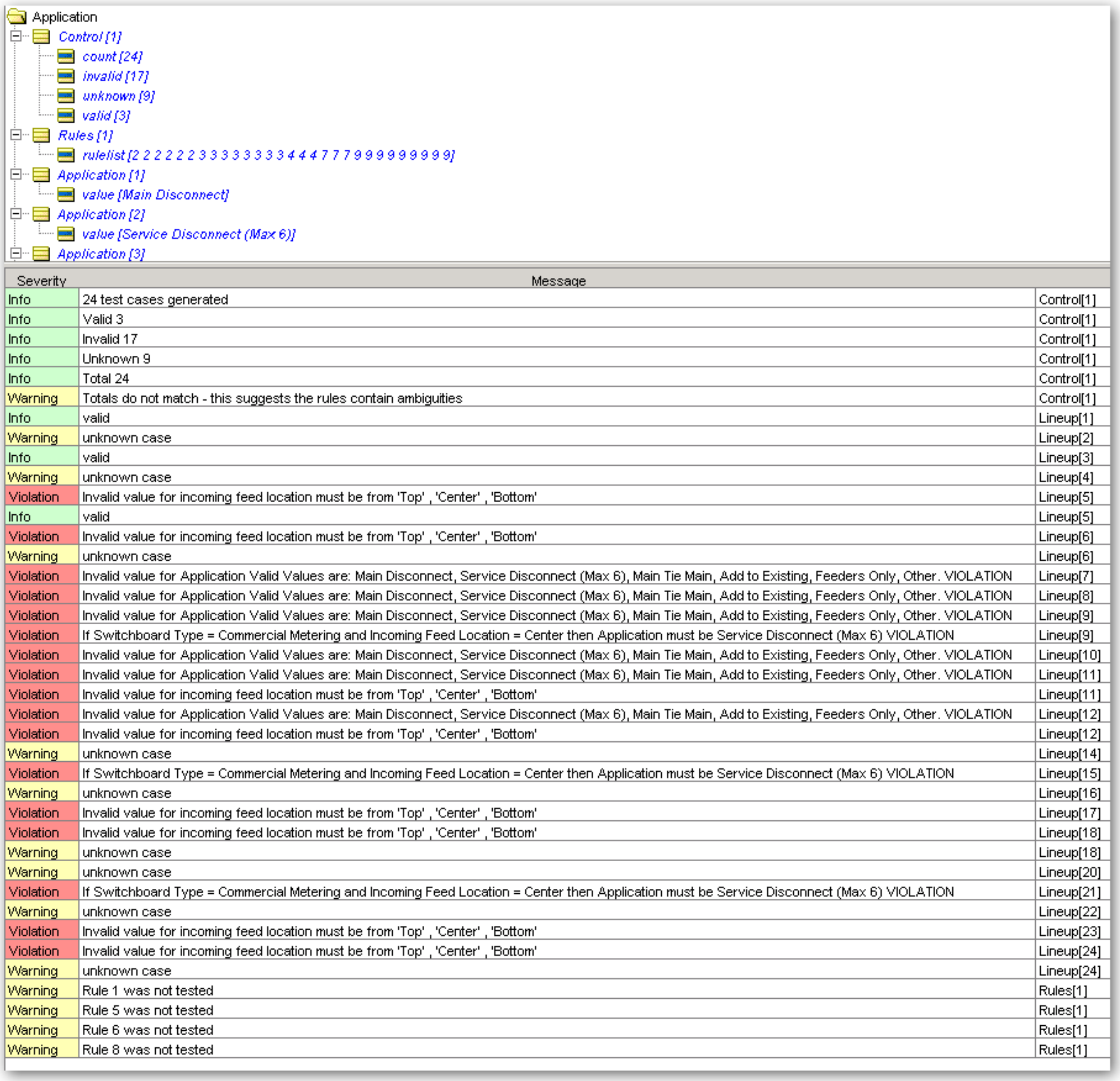
ID	Text
1	All rules were tested
2	Rule 1 was not tested
3	Rule 2 was not tested
4	Rule 3 was not tested
5	Rule 4 was not tested
6	Rule 5 was not tested
7	Rule 6 was not tested
8	Rule 7 was not tested
9	Rule 8 was not tested
10	Rule 9 was not tested

Here is the data we need to create to start things:



```
Application
  Control [1]
    count
    invalid [0]
    unknown [0]
    valid [0]
  Rules [1]
    rulelist []
```

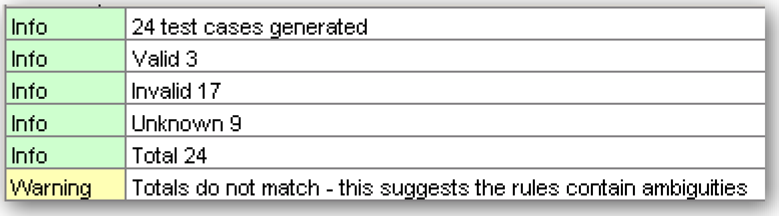
Corticon does the rest:



```
Application
  Control [1]
    count [24]
    invalid [17]
    unknown [9]
    valid [3]
  Rules [1]
    rulelist [2 2 2 2 2 2 3 3 3 3 3 3 3 3 4 4 4 7 7 7 9 9 9 9 9 9 9 9 9]
  Application [1]
    value [Main Disconnect]
  Application [2]
    value [Service Disconnect (Max 6)]
  Application [3]
```

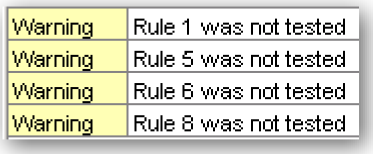
Severity	Message	
Info	24 test cases generated	Control[1]
Info	Valid 3	Control[1]
Info	Invalid 17	Control[1]
Info	Unknown 9	Control[1]
Info	Total 24	Control[1]
Warning	Totals do not match - this suggests the rules contain ambiguities	Control[1]
Info	valid	Lineup[1]
Warning	unknown case	Lineup[2]
Info	valid	Lineup[3]
Warning	unknown case	Lineup[4]
Violation	Invalid value for incoming feed location must be from 'Top' , 'Center' , 'Bottom'	Lineup[5]
Info	valid	Lineup[5]
Violation	Invalid value for incoming feed location must be from 'Top' , 'Center' , 'Bottom'	Lineup[6]
Warning	unknown case	Lineup[6]
Violation	Invalid value for Application Valid Values are: Main Disconnect, Service Disconnect (Max 6), Main Tie Main, Add to Existing, Feeders Only, Other. VIOLATION	Lineup[7]
Violation	Invalid value for Application Valid Values are: Main Disconnect, Service Disconnect (Max 6), Main Tie Main, Add to Existing, Feeders Only, Other. VIOLATION	Lineup[8]
Violation	Invalid value for Application Valid Values are: Main Disconnect, Service Disconnect (Max 6), Main Tie Main, Add to Existing, Feeders Only, Other. VIOLATION	Lineup[9]
Violation	If Switchboard Type = Commercial Metering and Incoming Feed Location = Center then Application must be Service Disconnect (Max 6) VIOLATION	Lineup[9]
Violation	Invalid value for Application Valid Values are: Main Disconnect, Service Disconnect (Max 6), Main Tie Main, Add to Existing, Feeders Only, Other. VIOLATION	Lineup[10]
Violation	Invalid value for Application Valid Values are: Main Disconnect, Service Disconnect (Max 6), Main Tie Main, Add to Existing, Feeders Only, Other. VIOLATION	Lineup[11]
Violation	Invalid value for incoming feed location must be from 'Top' , 'Center' , 'Bottom'	Lineup[11]
Violation	Invalid value for Application Valid Values are: Main Disconnect, Service Disconnect (Max 6), Main Tie Main, Add to Existing, Feeders Only, Other. VIOLATION	Lineup[12]
Violation	Invalid value for incoming feed location must be from 'Top' , 'Center' , 'Bottom'	Lineup[12]
Warning	unknown case	Lineup[14]
Violation	If Switchboard Type = Commercial Metering and Incoming Feed Location = Center then Application must be Service Disconnect (Max 6) VIOLATION	Lineup[15]
Warning	unknown case	Lineup[16]
Violation	Invalid value for incoming feed location must be from 'Top' , 'Center' , 'Bottom'	Lineup[17]
Violation	Invalid value for incoming feed location must be from 'Top' , 'Center' , 'Bottom'	Lineup[18]
Warning	unknown case	Lineup[18]
Warning	unknown case	Lineup[20]
Violation	If Switchboard Type = Commercial Metering and Incoming Feed Location = Center then Application must be Service Disconnect (Max 6) VIOLATION	Lineup[21]
Warning	unknown case	Lineup[22]
Violation	Invalid value for incoming feed location must be from 'Top' , 'Center' , 'Bottom'	Lineup[23]
Violation	Invalid value for incoming feed location must be from 'Top' , 'Center' , 'Bottom'	Lineup[24]
Warning	unknown case	Lineup[24]
Warning	Rule 1 was not tested	Rules[1]
Warning	Rule 5 was not tested	Rules[1]
Warning	Rule 6 was not tested	Rules[1]
Warning	Rule 8 was not tested	Rules[1]

Notice that we are cross checking the totals calculated during execution



Info	24 test cases generated
Info	Valid 3
Info	Invalid 17
Info	Unknown 9
Info	Total 24
Warning	Totals do not match - this suggests the rules contain ambiguities

as well as identifying the rules that did not fire:



Warning	Rule 1 was not tested
Warning	Rule 5 was not tested
Warning	Rule 6 was not tested
Warning	Rule 8 was not tested