Combinatorial Test Design

Eitan Farchi
Haifa Research Labs

Motivation

- The testing challenge:
  - We have too many combinations to deal with
  - We would like to use our time efficiently
  - We would like to control the risks we are taking
  - We would like to know what we tested
    - Minimize omissions

- A solution: Combinatorial Test Design (CTD)
  - Systematic planning of tests
  - Maximizes the value of each tested scenario
    - Impressive reduction in the number of tests
  - Controlled risk
  - Easy to review
    - Minimizes omissions

Success stories

- For a customer in the Health Insurance Industry
  - The client had 15,000 tests, manually reduced to 6000 based on risk estimates
  - We modeled the claims adjudication process using CTD
  - We identified 41 test cases to perform system test with better coverage
- For a customer in the Telecommunication Industry
  - We reverse-engineered the model present in 117 hand-written test cases
  - Concluded that these tests could be replaced by 12 test cases
- For a system recovery of an industrial operating system
  - The test team suggested ~50
  - Tests in this context take a few days to execute
  - After holes were found and a model was created, there were ~7,800 tests
  - CTD suggested only 17
  - Out of the 17 tests, 14 revealed unknown defects
  - A total of 20 new defects identified

Agenda

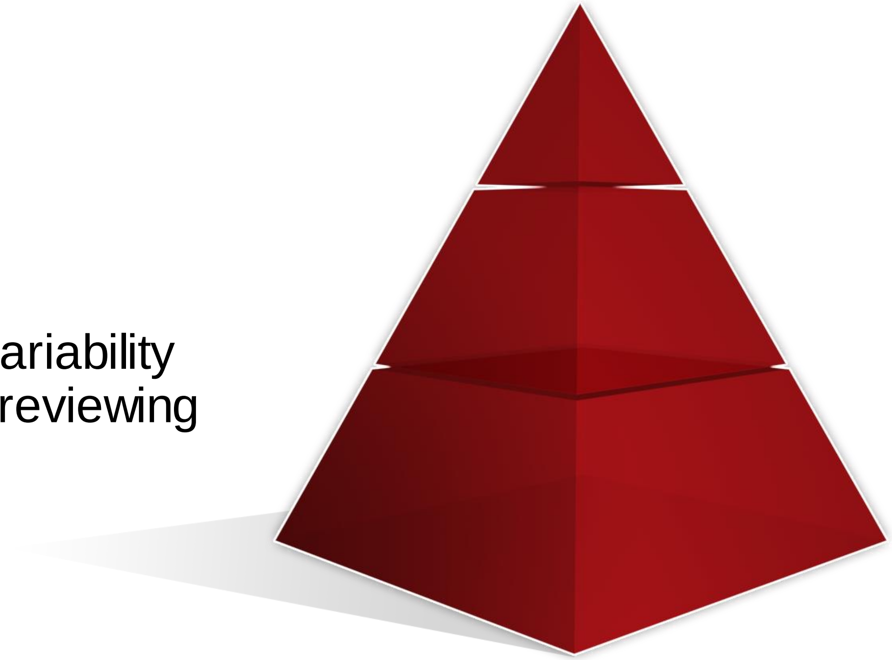
Topic 1 – Cartesian products

Topic 2 – Restrictions

Topic 3 – CTD implementation using BDD

Topic 4 – Creating a model of the points of variability
in a system, using interviews and reviewing
documents

Topic 5 – Handling existing tests



Topic 1

Cartesian Products

The Cartesian Product

- The Cartesian product of two sets *X* and *Y*, denoted *X* × *Y*, is the set of all possible ordered pairs whose first component is a member of *X* and whose second component is a member of *Y.*

- For example, let X be {Ace, 2, 3, …, 9, 10, Jack, Queen, King} and Y be {Diamond, Heart, Club, Spade}, then *X* × *Y* is the 52-element set of all possible playing cards.

- 52 = 13 * 4

- Adding a third set, e.g., Z = {Deck1, Deck2, Deck3}, we have *X* × *Y* × Z, with 13 * 4 * 3 = 156 elements.
- And so on.

Toy Example – Online Shopping System

- Parameters:
  - Availability
  - Payment method
  - Carrier
  - Delivery schedule
  - Export control

Toy Example – Online Shopping System – cont.

Availability	Payment	Carrier	Delivery Schedule	Export Control
▪Available ▪Not in Stock ▪Discontinued ▪No Such Product	▪Credit ▪Paypal ▪Gift Voucher	▪Mail ▪UPS ▪Fedex	▪One Day ▪2-5 Working Days ▪6-10 Working Days ▪Over 10 Working Days	▪True ▪False

A test is represented by an assignment of exactly one value to each parameter

4 x 3 x 3 x 4 x 2 = 288 combinations

Discussion

- How would you choose tests for this system?

- How many tests do you require?

- How do you review your choices?

- How do you prove the validity of your choices?

Levels of interaction

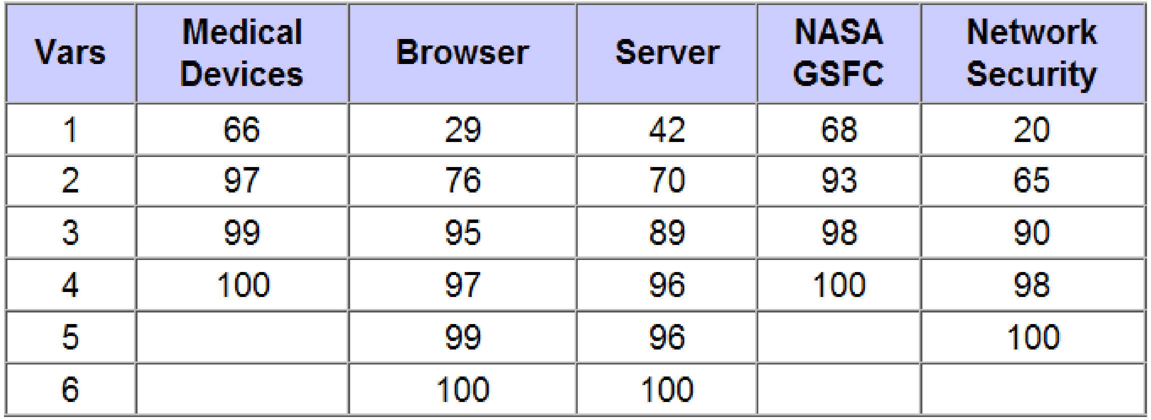
- Suppose there is a bug, and Credit does not work well with One Day delivery
- Any combination that includes Credit and a One Day delivery will expose that bug
  - There are 24 such combinations
    (All combinations in which payment=credit & delivery=one day. Values for availability, carrier and export control are free. 4 x 3 x 2 = 24)
- Suppose Credit does not work well with a One Day delivery, but only with Fedex
- Any combination that includes Credit, a One Day delivery, and Fedex will expose that bug
  - There are 8 such combinations
- We call the first case a level two interaction, and the second case a level three interaction

Do we really need to test all combinations?

The root cause analysis of many bugs shows they depend on a value of one variable (20%-68%)

Most defects can be discovered in tests of the interactions between the values of two variables (65-97%)

Table 1. Number of variables involved in triggering software faults



Vars	Medical Devices	Browser	Server	NASA GSFC	Network Security
1	66	29	42	68	20
2	97	76	70	93	65
3	99	95	89	98	90
4	100	97	96	100	98
5		99	96		100
6		100	100		

- Source http://csrc.nist.gov/groups/SNS/acts/ftfi.html

Coverage of interactions

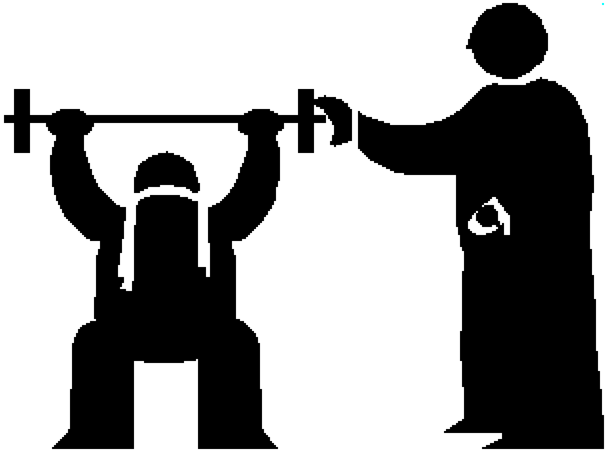
- Let's take interaction level 2 for example:
  - There are 101 different pairs of values:
    - Payment = Credit, Delivery = One Day
    - Payment = Credit, Delivery = 2-5 Days
    - …
    - Availability = Available, Delivery = One Day
    - …
  - A given test plan covers x% of interaction level 2 if it covers x% of these 101 pairs
  - 100% pairwise coverage means that each pair appears at least once
  - A test plan that gives 100% pairwise coverage will reveal all defects that result from an interaction level of 2 (expected 65-97% of the defects)
- Explaining the "101" number above: Sum of the following:
  - 3 x 4 = 12 pairs for Payment & Delivery
  - 4 x 4 = 16 pairs for Availability & Delivery
  - Etc.

Combinatorial Test Design (CTD)

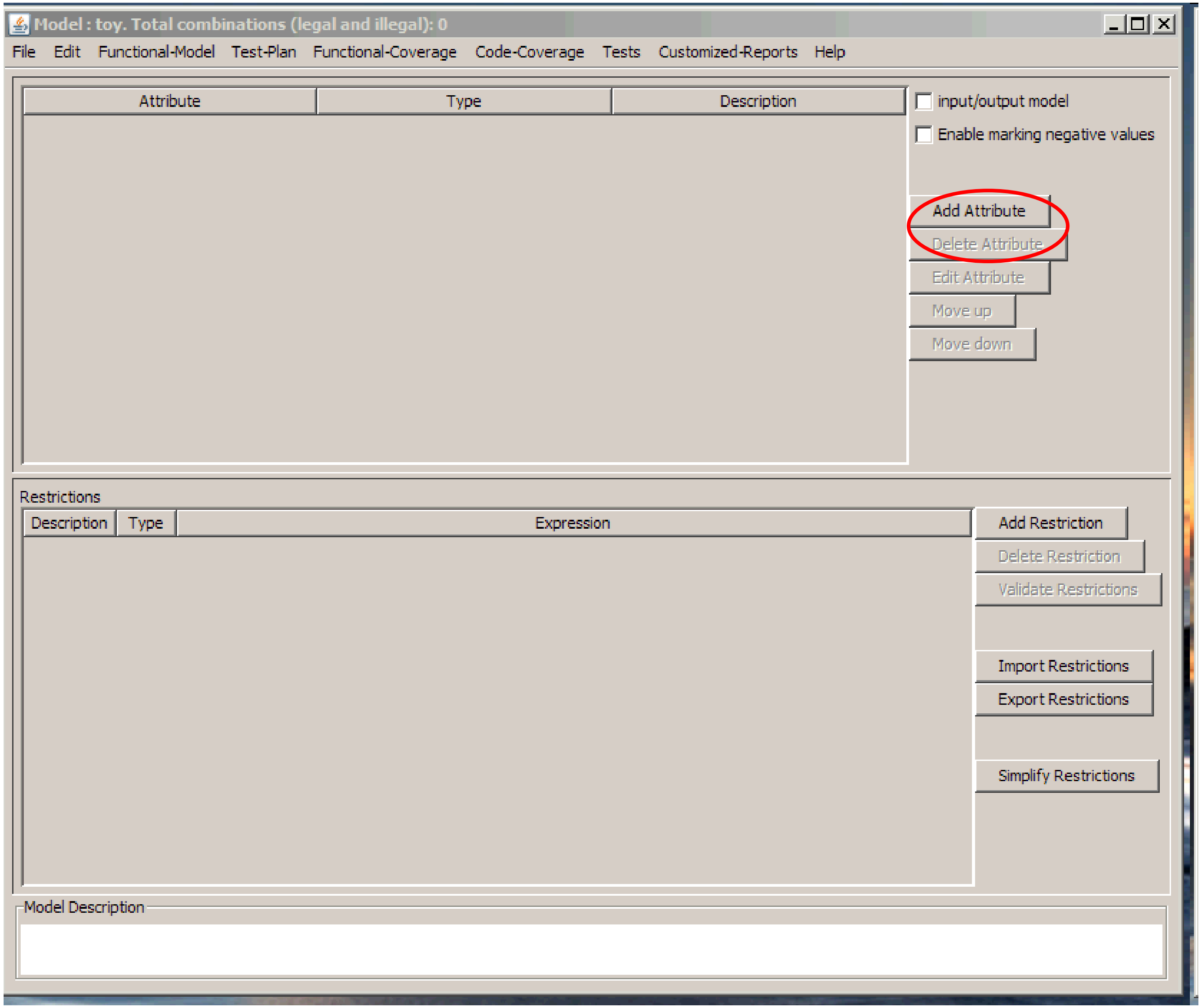
- To balance cost and risk, we select a subset of tests that covers all the interactions of variables at some level of interaction (pairs, three-way, etc.)
- A combinatorial test design (CTD) algorithm finds a small test plan that covers 100% of a given interaction level
- Note that each test (combination of values) covers many interactions

Typical definition of a CTD model

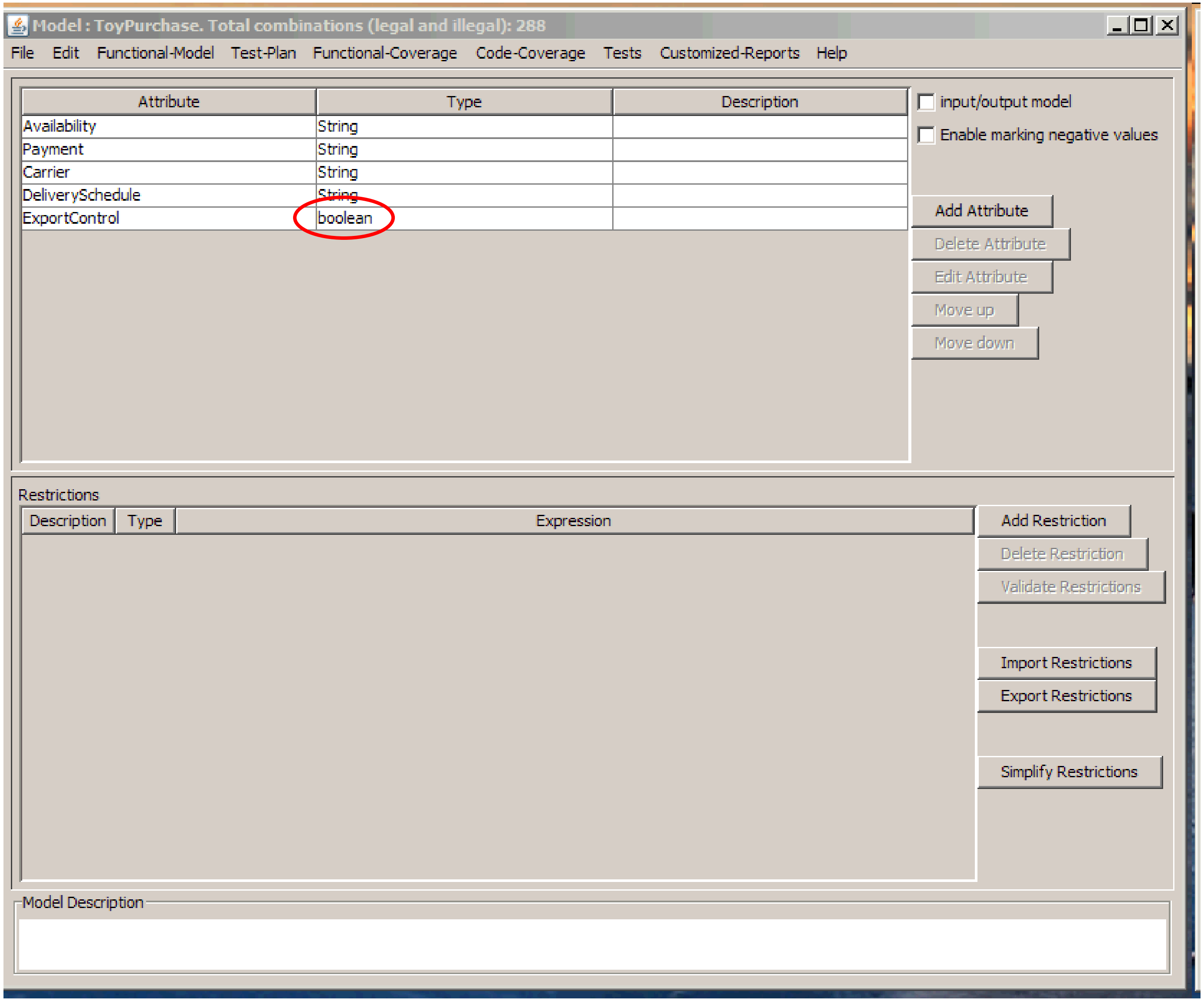
- We will perform the following:
  - Enter the toy model
  - Explore CTD
  - Export our results
- Notes:
  - Defining levels of interaction



New model


Model : toy. Total combinations (legal and illegal): 0
File
Edit
Functional-Model
Test-Plan
Functional-Coverage
Code-Coverage
Tests
Customized-Reports
Help
Attribute
Type
Description
□ input/output model
□ Enable marking negative values
Add Attribute
Delete Attribute
Edit Attribute
Move up
Move down
Restrictions
Description
Type
Expression
Add Restriction
Delete Restriction
Validate Restrictions
Import Restrictions
Export Restrictions
Simplify Restrictions
Model Description

Model



Model : ToyPurchase. Total combinations (legal and illegal): 288

Attribute	Type	Description
Availability	String	
Payment	String	
Carrier	String	
DeliverySchedule	String	
ExportControl	boolean	

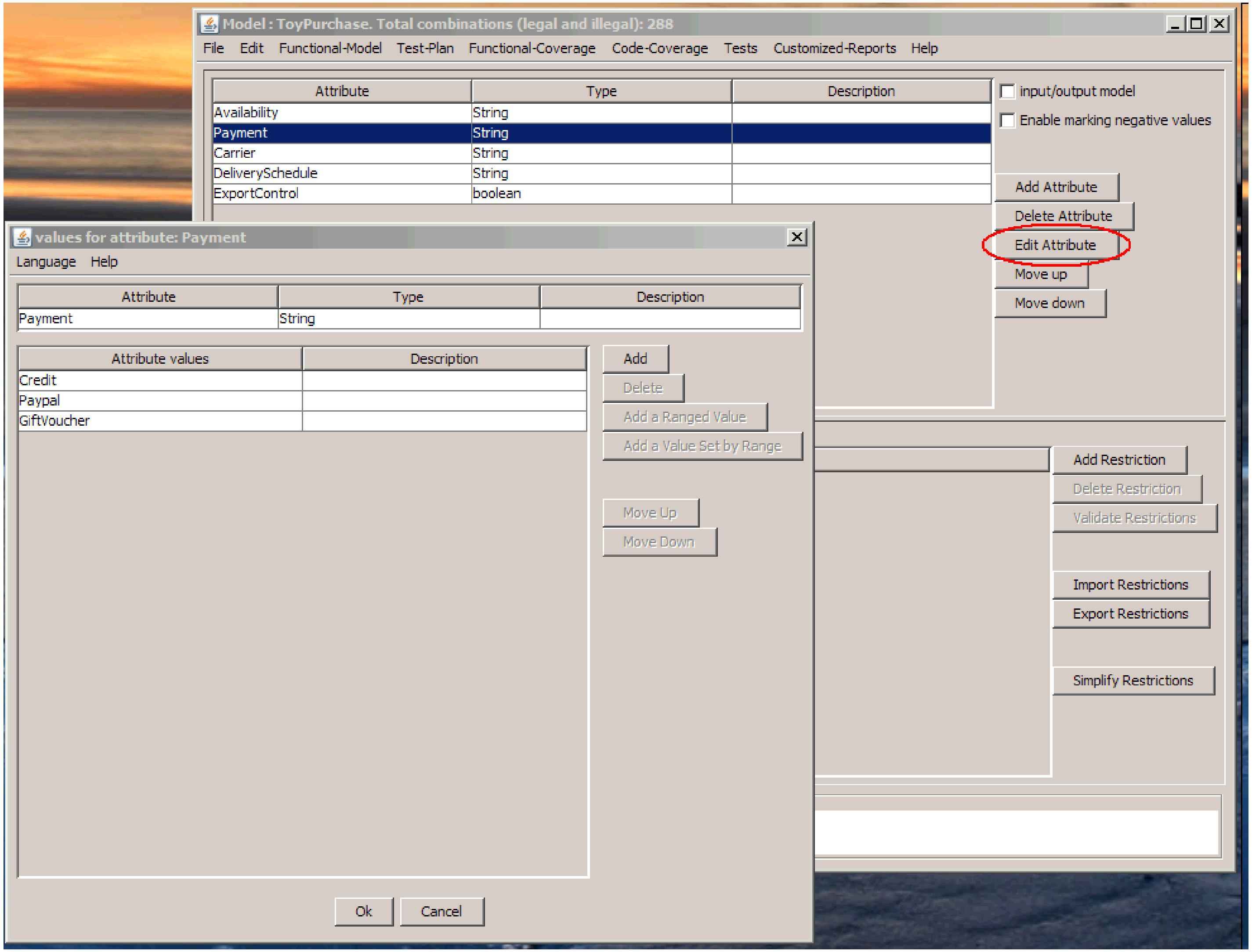
□ input/output model

□ Enable marking negative values

Restrictions

Description	Type	Expression

Edit attribute



Model : ToyPurchase. Total combinations (legal and illegal): 288

Attribute	Type	Description
Availability	String	
Payment	String	
Carrier	String	
DeliverySchedule	String	
ExportControl	boolean	

☐ input/output model

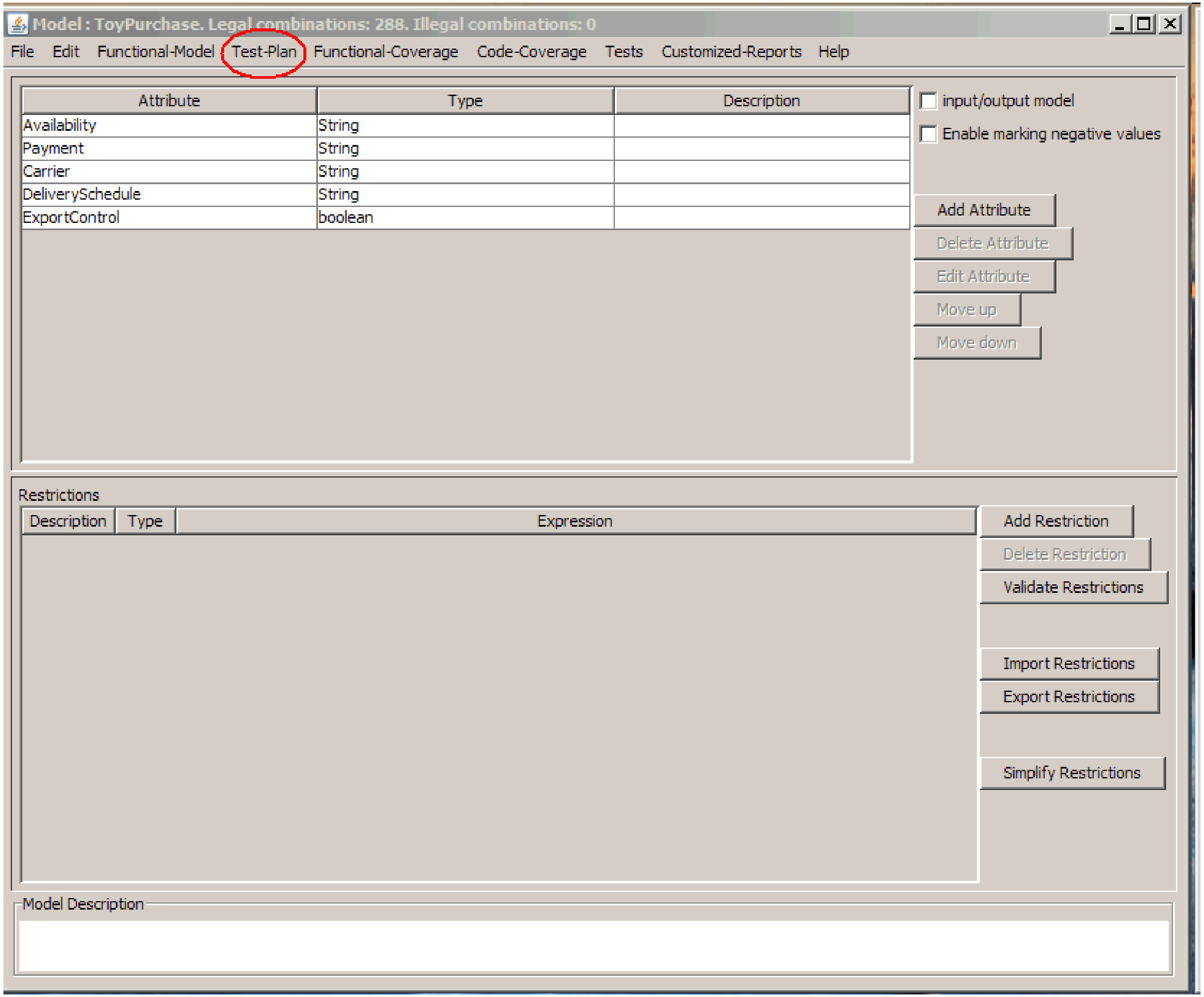
☐ Enable marking negative values

values for attribute: Payment

Attribute	Type	Description
Payment	String	

Attribute values	Description
Credit	
Paypal	
GiftVoucher	

Test planning



Model : ToyPurchase. Legal combinations: 288. Illegal combinations: 0

Attribute	Type	Description
Availability	String	
Payment	String	
Carrier	String	
DeliverySchedule	String	
ExportControl	boolean	

□ input/output model

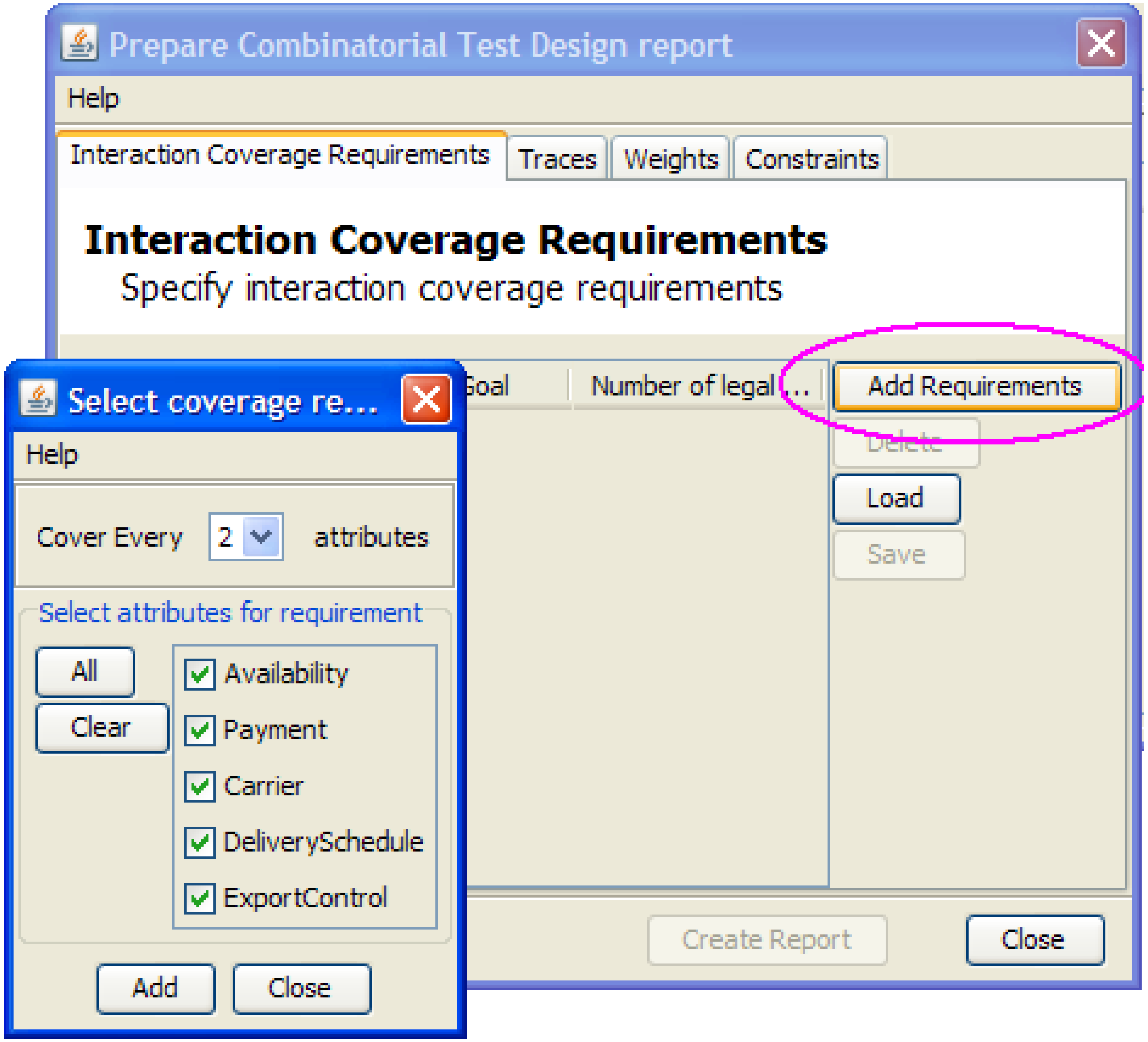
□ Enable marking negative values

Restrictions

Description	Type	Expression

Model Description

Interaction level



Interaction Coverage Requirements

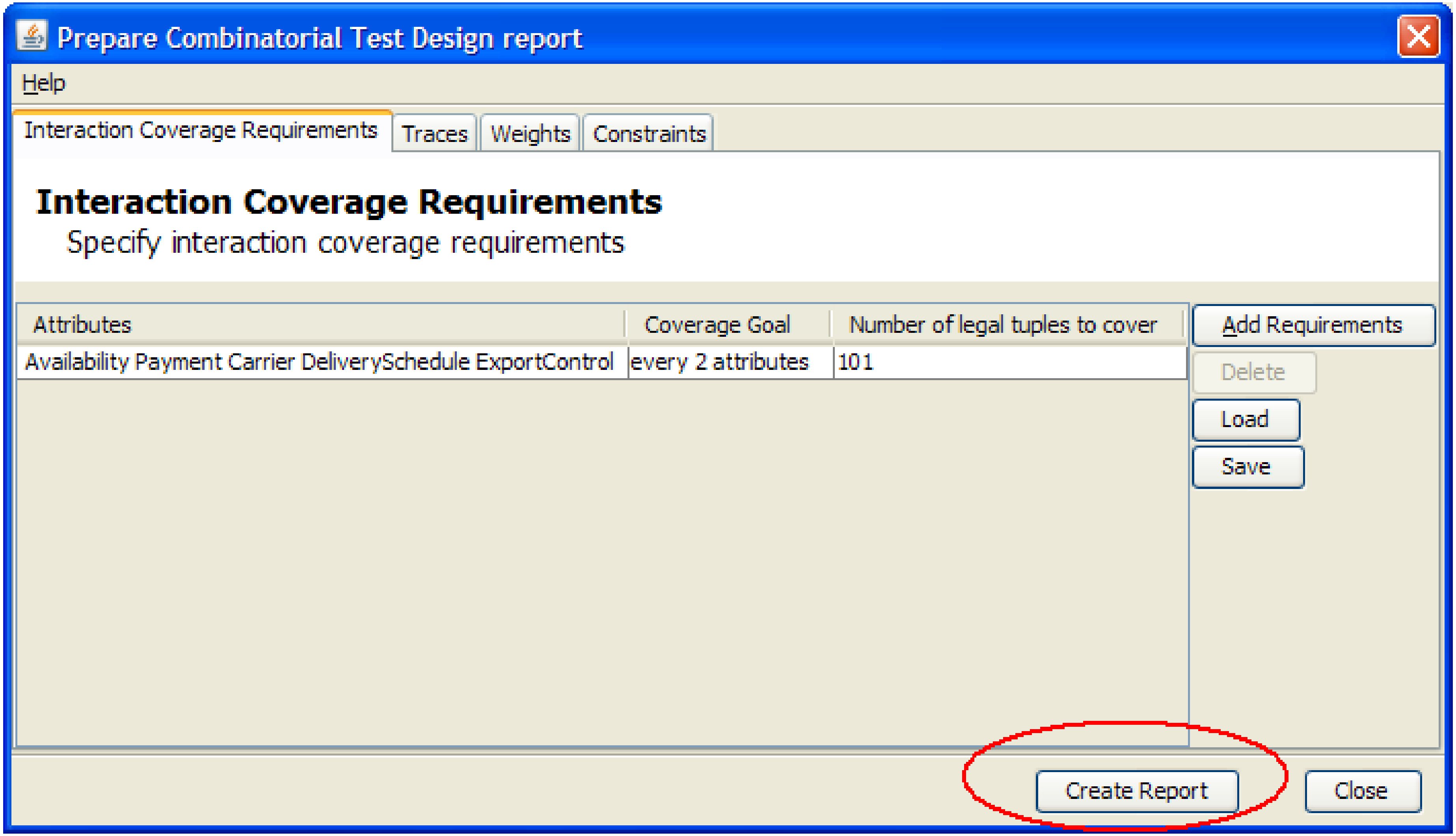
Specify interaction coverage requirements

Cover Every 2 attributes

Select attributes for requirement

- ☑ Availability
- ☑ Payment
- ☑ Carrier
- ☑ DeliverySchedule
- ☑ ExportControl

Create report



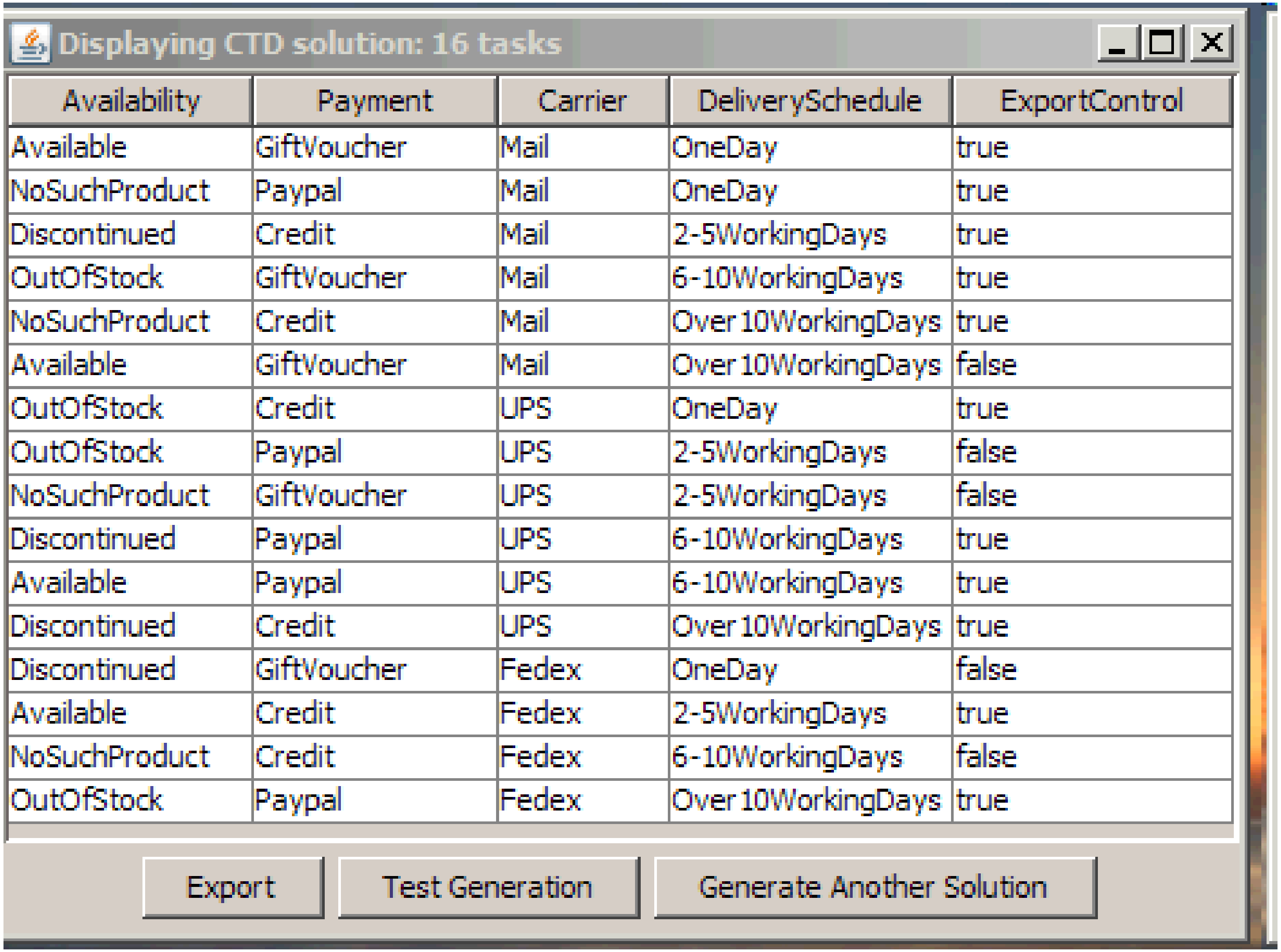
Prepare Combinatorial Test Design report

Interaction Coverage Requirements

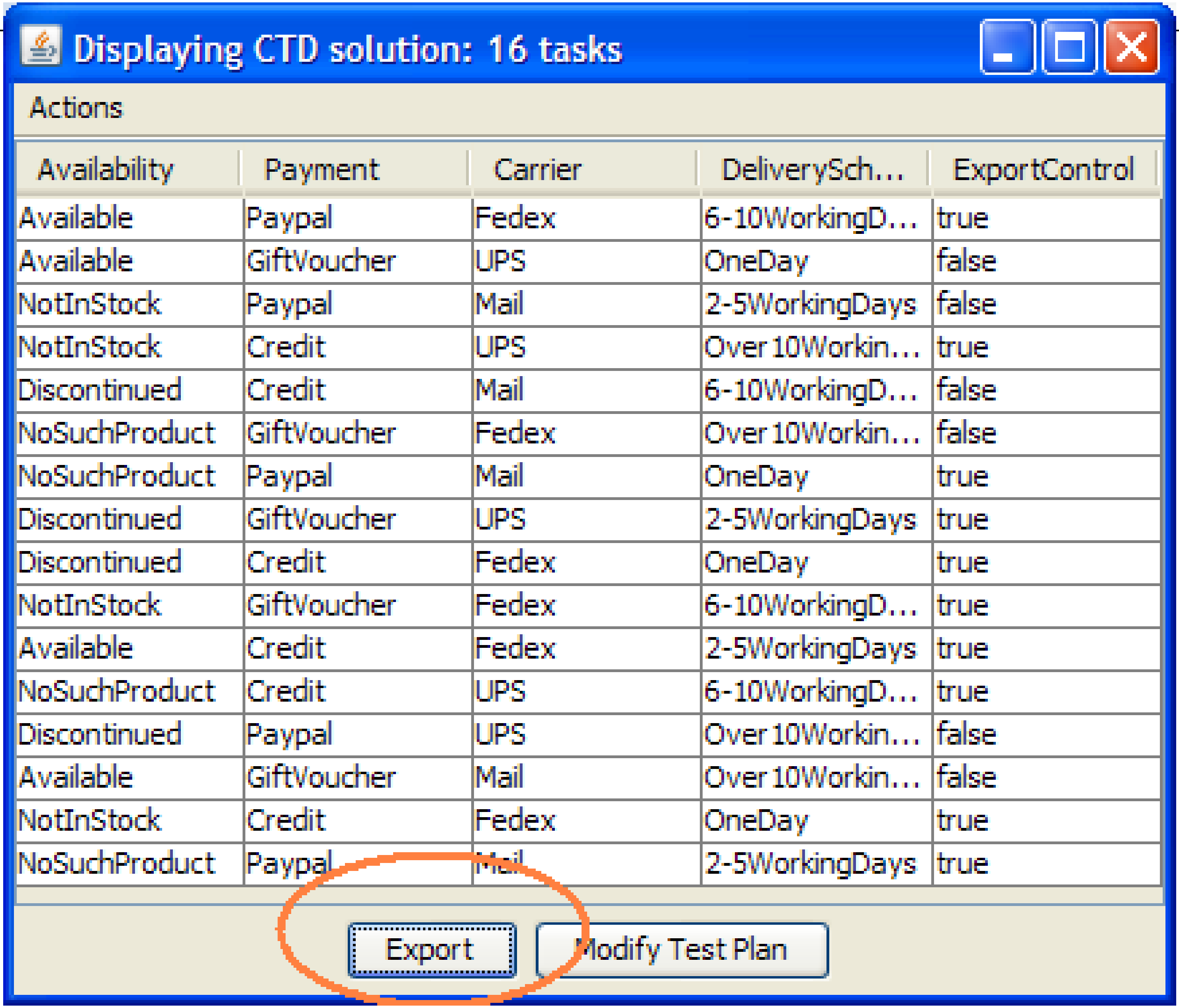
Specify interaction coverage requirements

Attributes	Coverage Goal	Number of legal tuples to cover
Availability Payment Carrier DeliverySchedule ExportControl	every 2 attributes	101

Complete pairwise coverage (one of many)



Availability	Payment	Carrier	DeliverySchedule	ExportControl
Available	GiftVoucher	Mail	OneDay	true
NoSuchProduct	Paypal	Mail	OneDay	true
Discontinued	Credit	Mail	2-5WorkingDays	true
OutOfStock	GiftVoucher	Mail	6-10WorkingDays	true
NoSuchProduct	Credit	Mail	Over10WorkingDays	true
Available	GiftVoucher	Mail	Over10WorkingDays	false
OutOfStock	Credit	UPS	OneDay	true
OutOfStock	Paypal	UPS	2-5WorkingDays	false
NoSuchProduct	GiftVoucher	UPS	2-5WorkingDays	false
Discontinued	Paypal	UPS	6-10WorkingDays	true
Available	Paypal	UPS	6-10WorkingDays	true
Discontinued	Credit	UPS	Over10WorkingDays	true
Discontinued	GiftVoucher	Fedex	OneDay	false
Available	Credit	Fedex	2-5WorkingDays	true
NoSuchProduct	Credit	Fedex	6-10WorkingDays	false
OutOfStock	Paypal	Fedex	Over10WorkingDays	true



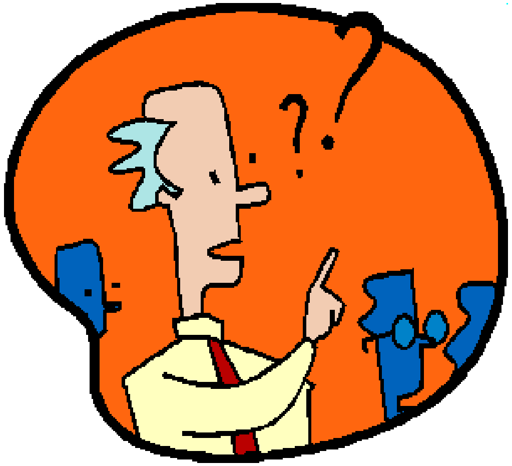
Availability	Payment	Carrier	DeliverySch...	ExportControl
Available	Paypal	Fedex	6-10WorkingD...	true
Available	GiftVoucher	UPS	OneDay	false
NotInStock	Paypal	Mail	2-5WorkingDays	false
NotInStock	Credit	UPS	Over 10Workin...	true
Discontinued	Credit	Mail	6-10WorkingD...	false
NoSuchProduct	GiftVoucher	Fedex	Over 10Workin...	false
NoSuchProduct	Paypal	Mail	OneDay	true
Discontinued	GiftVoucher	UPS	2-5WorkingDays	true
Discontinued	Credit	Fedex	OneDay	true
NotInStock	GiftVoucher	Fedex	6-10WorkingD...	true
Available	Credit	Fedex	2-5WorkingDays	true
NoSuchProduct	Credit	UPS	6-10WorkingD...	true
Discontinued	Paypal	UPS	Over 10Workin...	false
Available	GiftVoucher	Mail	Over 10Workin...	false
NotInStock	Credit	Fedex	OneDay	true
NoSuchProduct	Paypal	Mail	2-5WorkingDays	true

Note: Test Plans vs. Actual Tests

- The CTD tool generates a test plan, not actual tests
- Extracting actual tests from the generated test plan may be a laborious task – generate data, generate test environments, etc.
- However, the model must be built in a way that a row in the test plan corresponds to an actual test
  - When looking at the test plan, it should be clear to you what test each row represents

How do I determine coverage requirements?

- Consider important (= risky) interactions
- Examine different possibilities
  - Easy to generate using CTD
- Balance cost and risk
- Consult SMEs !
- Different levels may be defined for different sets of attributes
- Coverage requirements may overlap



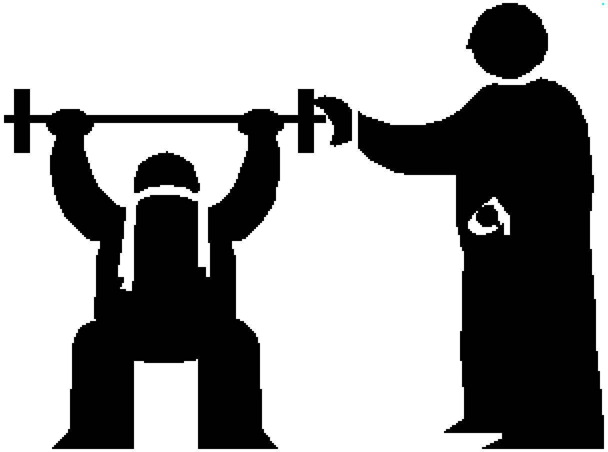
Determining coverage requirements – example

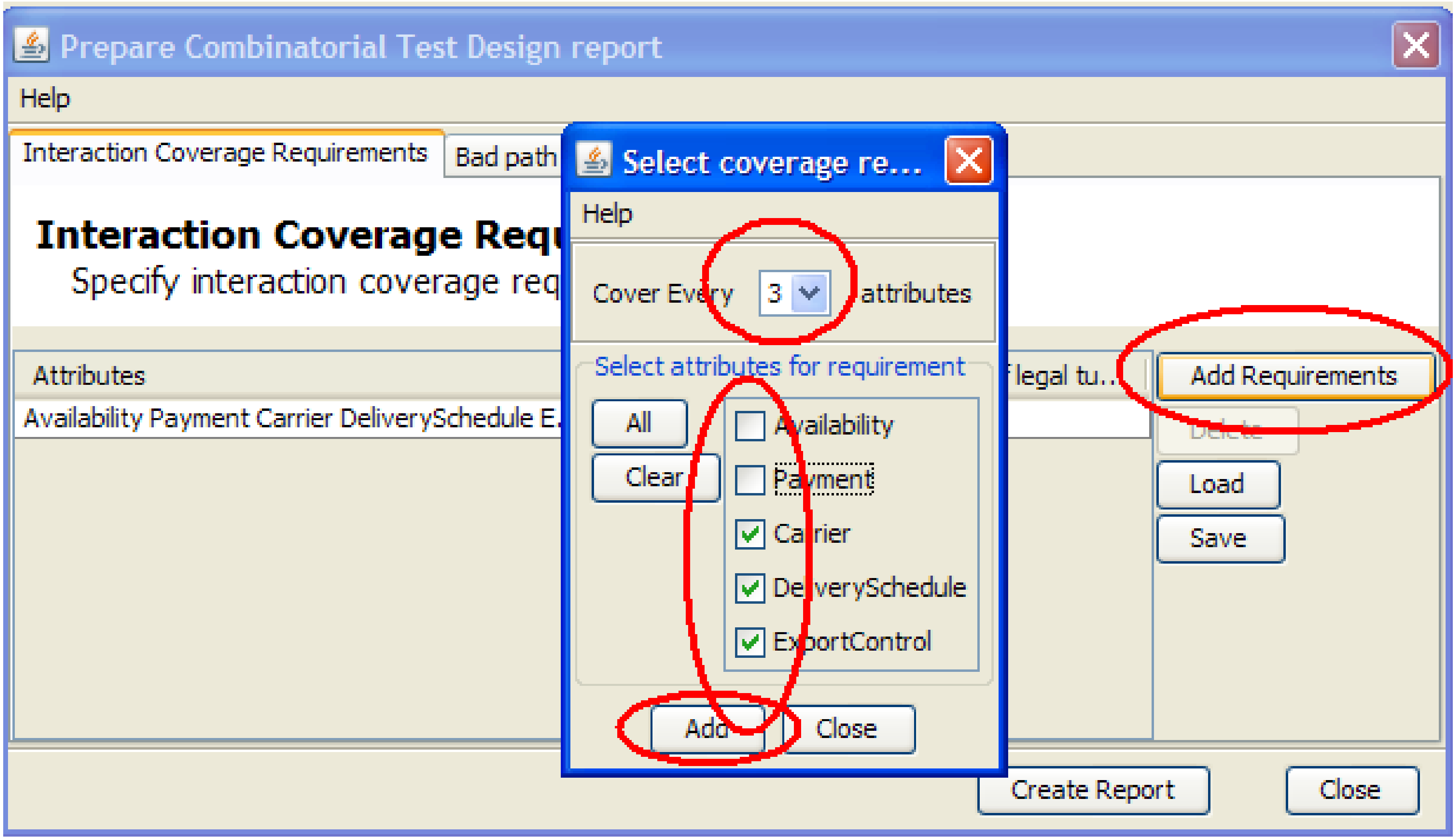
- A SME might point out (or more commonly – you might suspect and verify with a SME) that the combination of carrier, delivery schedule and export control is risky –
  - Intuition is that export control may cause delivery delays
  - Some combinations between the three might cause problems
- For these attributes, three-way coverage should be considered
- Review sets of attributes (pairs, triplets..) and consider whether interactions between them is important

Typical definition of coverage requirements

- We will perform the following:
  - Define interaction level 2 for all attributes + level 3 for carrier, delivery schedule and export control
- Note:
  - Change in test plan size
  - Change in coverage:
    Some combinations of carrier, delivery schedule and export control were not covered before, and are now, e.g.:

    Mail, 2-5 days, no export control





Prepare Combinatorial Test Design report

Interaction Coverage Requ

Specify interaction coverage req

Attributes	legal tu...
Availability Payment Carrier DeliverySchedule E...	

Select coverage re...

Cover Every 3 attributes

Select attributes for requirement

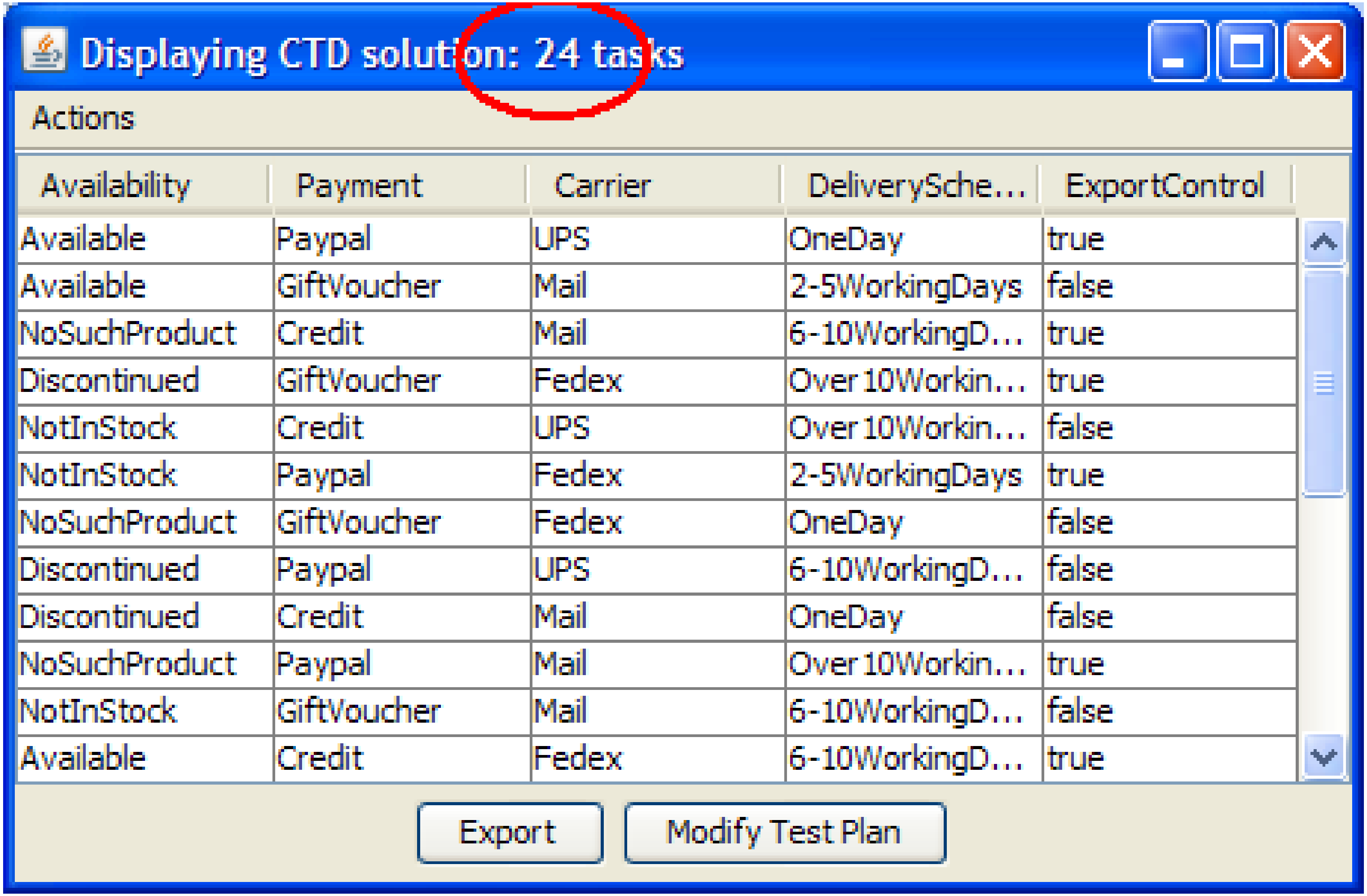
□ Availability

□ Payment

☑ Carrier

☑ DeliverySchedule

☑ ExportControl

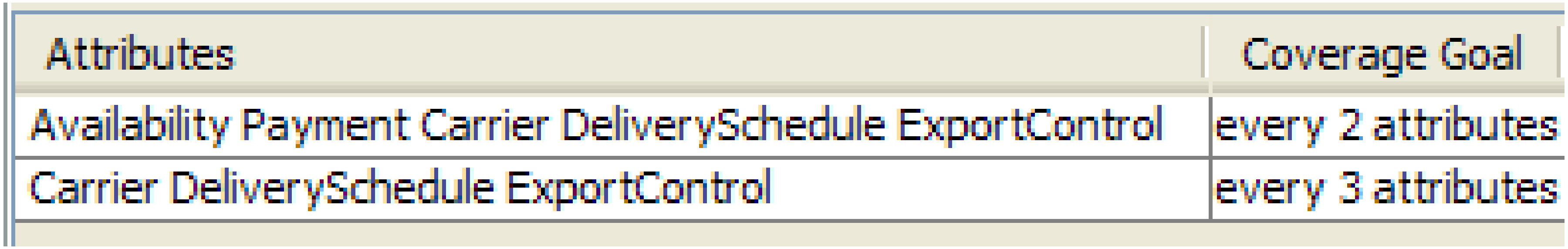


Displaying CTD solution: 24 tasks

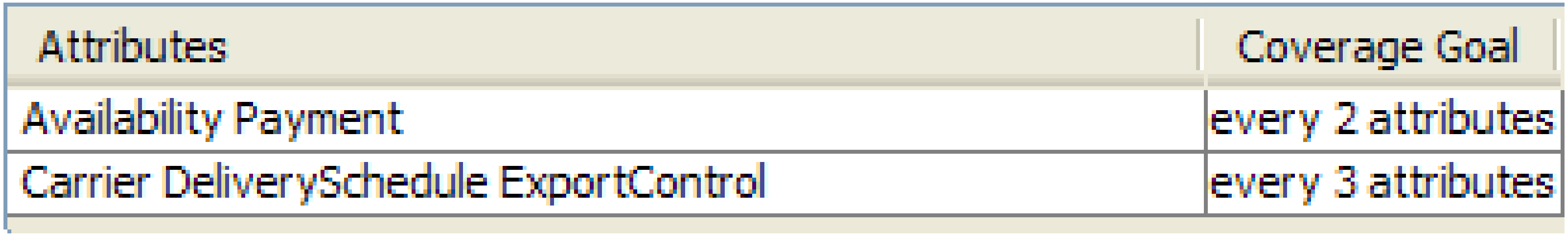
Availability	Payment	Carrier	DeliverySche…	ExportControl
Available	Paypal	UPS	OneDay	true
Available	GiftVoucher	Mail	2-5WorkingDays	false
NoSuchProduct	Credit	Mail	6-10WorkingD…	true
Discontinued	GiftVoucher	Fedex	Over 10Workin…	true
NotInStock	Credit	UPS	Over 10Workin…	false
NotInStock	Paypal	Fedex	2-5WorkingDays	true
NoSuchProduct	GiftVoucher	Fedex	OneDay	false
Discontinued	Paypal	UPS	6-10WorkingD…	false
Discontinued	Credit	Mail	OneDay	false
NoSuchProduct	Paypal	Mail	Over 10Workin…	true
NotInStock	GiftVoucher	Mail	6-10WorkingD…	false
Available	Credit	Fedex	6-10WorkingD…	true

Discussion

- Is there a difference between the following two requirements?

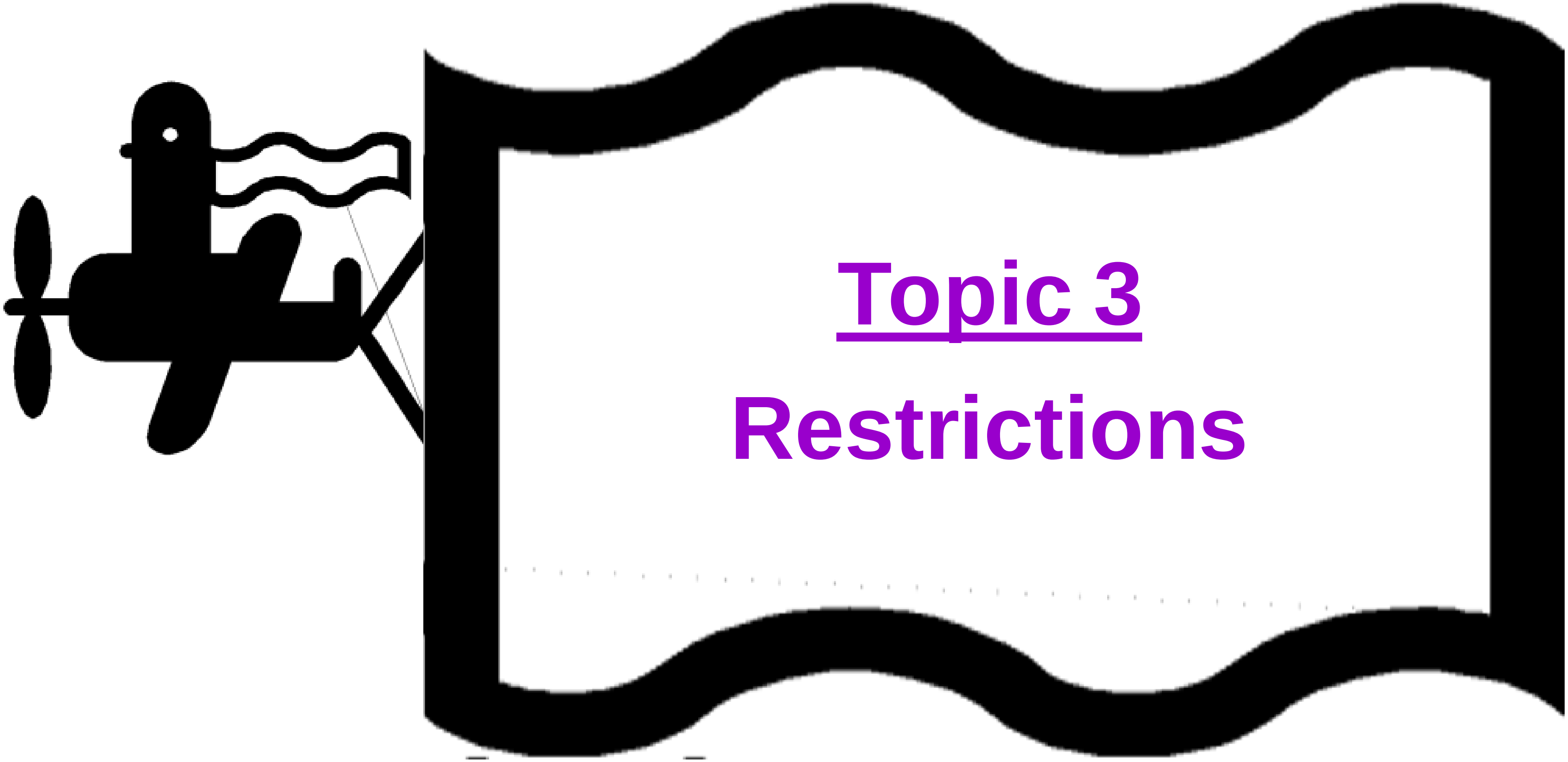


Attributes	Coverage Goal
Availability Payment Carrier DeliverySchedule ExportControl	every 2 attributes
Carrier DeliverySchedule ExportControl	every 3 attributes



Attributes	Coverage Goal
Availability Payment	every 2 attributes
Carrier DeliverySchedule ExportControl	every 3 attributes

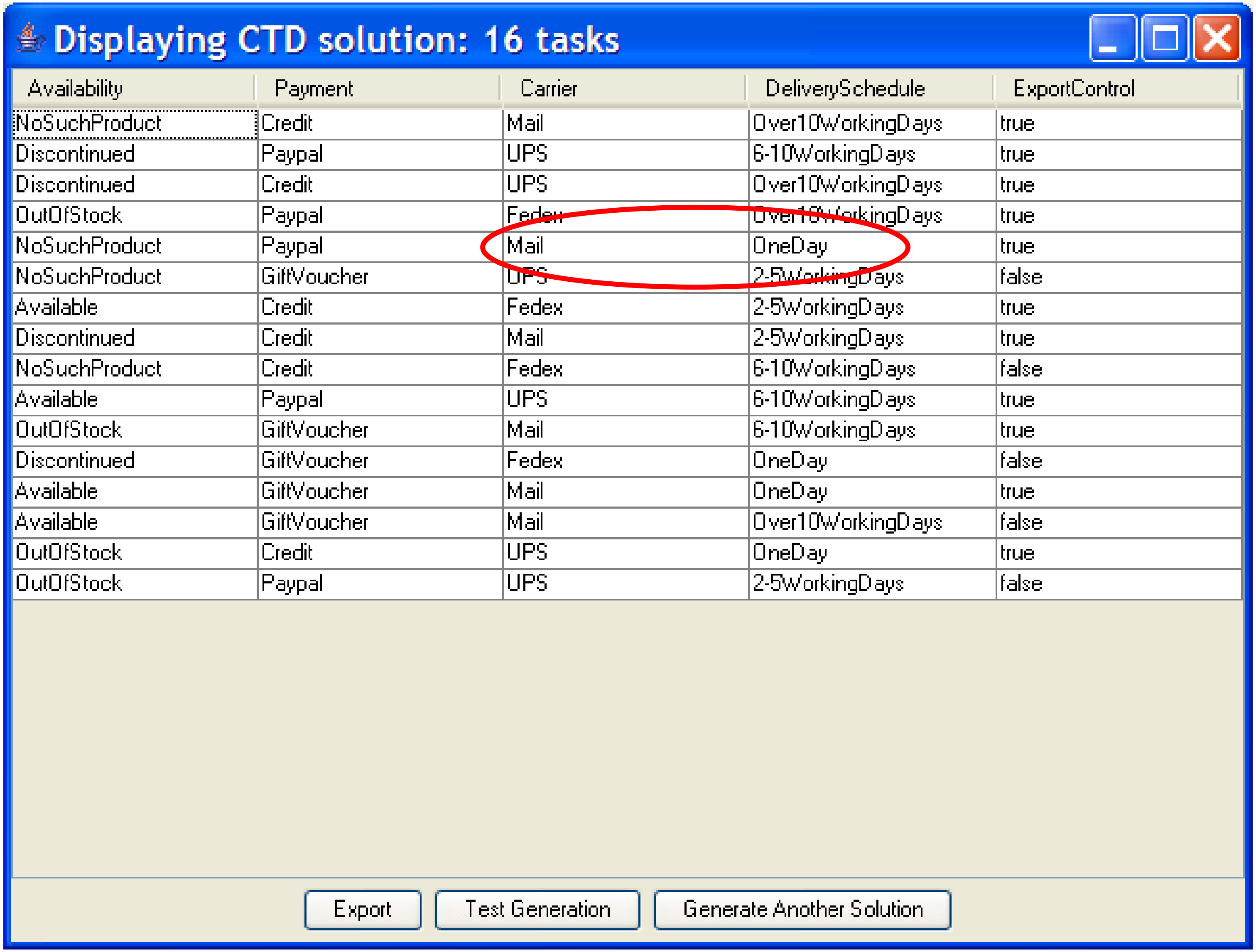
Answer: Yes, there is a difference. For example, interaction between availability and delivery schedule is not required to be covered by the second option.



Topic 3

Restrictions

Complete pairwise coverage



Availability	Payment	Carrier	DeliverySchedule	ExportControl
NoSuchProduct	Credit	Mail	Over10WorkingDays	true
Discontinued	Paypal	UPS	6-10WorkingDays	true
Discontinued	Credit	UPS	Over10WorkingDays	true
OutOfStock	Paypal	Fedex	Over10WorkingDays	true
NoSuchProduct	Paypal	Mail	OneDay	true
NoSuchProduct	GiftVoucher	UPS	2-5WorkingDays	false
Available	Credit	Fedex	2-5WorkingDays	true
Discontinued	Credit	Mail	2-5WorkingDays	true
NoSuchProduct	Credit	Fedex	6-10WorkingDays	false
Available	Paypal	UPS	6-10WorkingDays	true
OutOfStock	GiftVoucher	Mail	6-10WorkingDays	true
Discontinued	GiftVoucher	Fedex	OneDay	false
Available	GiftVoucher	Mail	OneDay	true
Available	GiftVoucher	Mail	Over10WorkingDays	false
OutOfStock	Credit	UPS	OneDay	true
OutOfStock	Paypal	UPS	2-5WorkingDays	false

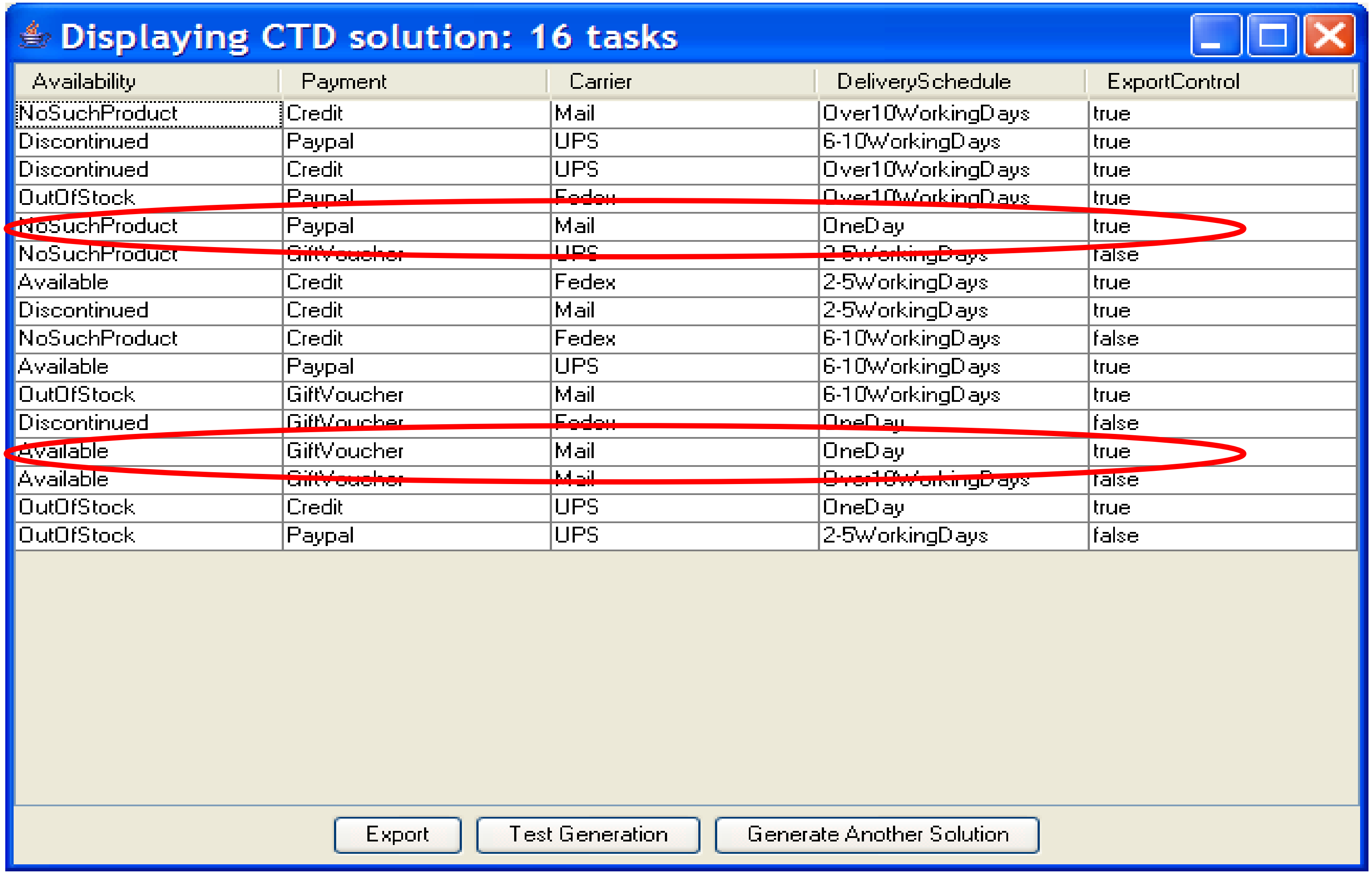
Why do we need restrictions?

- Impossible or irrelevant combinations, for example:
  - Mail Carrier with One Day Delivery Schedule
  - Fedex Carrier with Over 10 Working Days Delivery Schedule
  - and more..

- Naturally we cannot create and run actual tests that contain impossible combinations, so we need to state in advance what should be excluded

Why not just skip tests that contain impossible/irrelevant combinations?

- Assume we skip all tests with mail carrier in one day:



Availability	Payment	Carrier	DeliverySchedule	ExportControl
NoSuchProduct	Credit	Mail	Over10WorkingDays	true
Discontinued	Paypal	UPS	6-10WorkingDays	true
Discontinued	Credit	UPS	Over10WorkingDays	true
OutOfStock	Paypal	Fedex	Over10WorkingDays	true
NoSuchProduct	Paypal	Mail	OneDay	true
NoSuchProduct	GiftVoucher	UPS	2-5WorkingDays	false
Available	Credit	Fedex	2-5WorkingDays	true
Discontinued	Credit	Mail	2-5WorkingDays	true
NoSuchProduct	Credit	Fedex	6-10WorkingDays	false
Available	Paypal	UPS	6-10WorkingDays	true
OutOfStock	GiftVoucher	Mail	6-10WorkingDays	true
Discontinued	GiftVoucher	Fedex	OneDay	false
Available	GiftVoucher	Mail	OneDay	true
Available	GiftVoucher	Mail	Over10WorkingDays	false
OutOfStock	Credit	UPS	OneDay	true
OutOfStock	Paypal	UPS	2-5WorkingDays	false

Why not just skip tests that contain impossible/irrelevant combinations?

5 legal pairs are now uncovered, in addition to the excluded pair !

Availability=NoSuchProduct, Payment=Paypal

Payment=Paypal, Carrier=Mail

Availability=Available, DeliverySchedule=OneDay

Availability=NoSuchProduct, DeliverySchedule=OneDay

Payment=Paypal, DeliverySchedule=OneDay

(We will learn how to perform this analysis later in the course)

Why not just skip tests that contain impossible/irrelevant combinations?

- Each test in the CTD test plan may cover multiple unique legal combinations

- By skipping a test we will lose all these combinations, and potentially no longer have 100% interaction coverage

What are restrictions?

- Restrictions are rules that determine which combinations are included and which are excluded from the model
- Combinations that are excluded from the model will never appear in the test plan nor in coverage analyses
  - So it is important to define them carefully

How do I define restrictions?

By marking and excluding combinations in the Cartesian product report

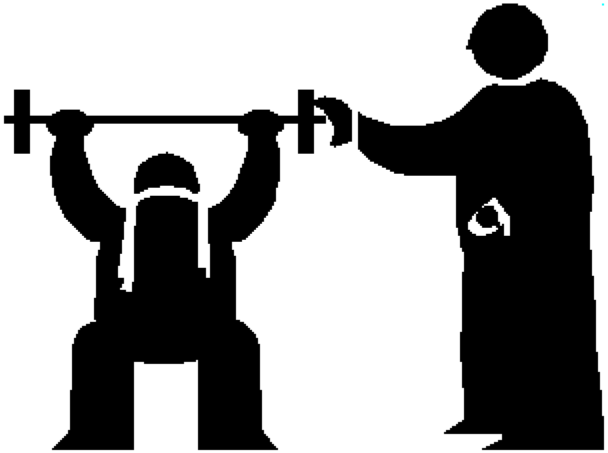
Or

By writing explicit conditions on what combinations should be included/excluded

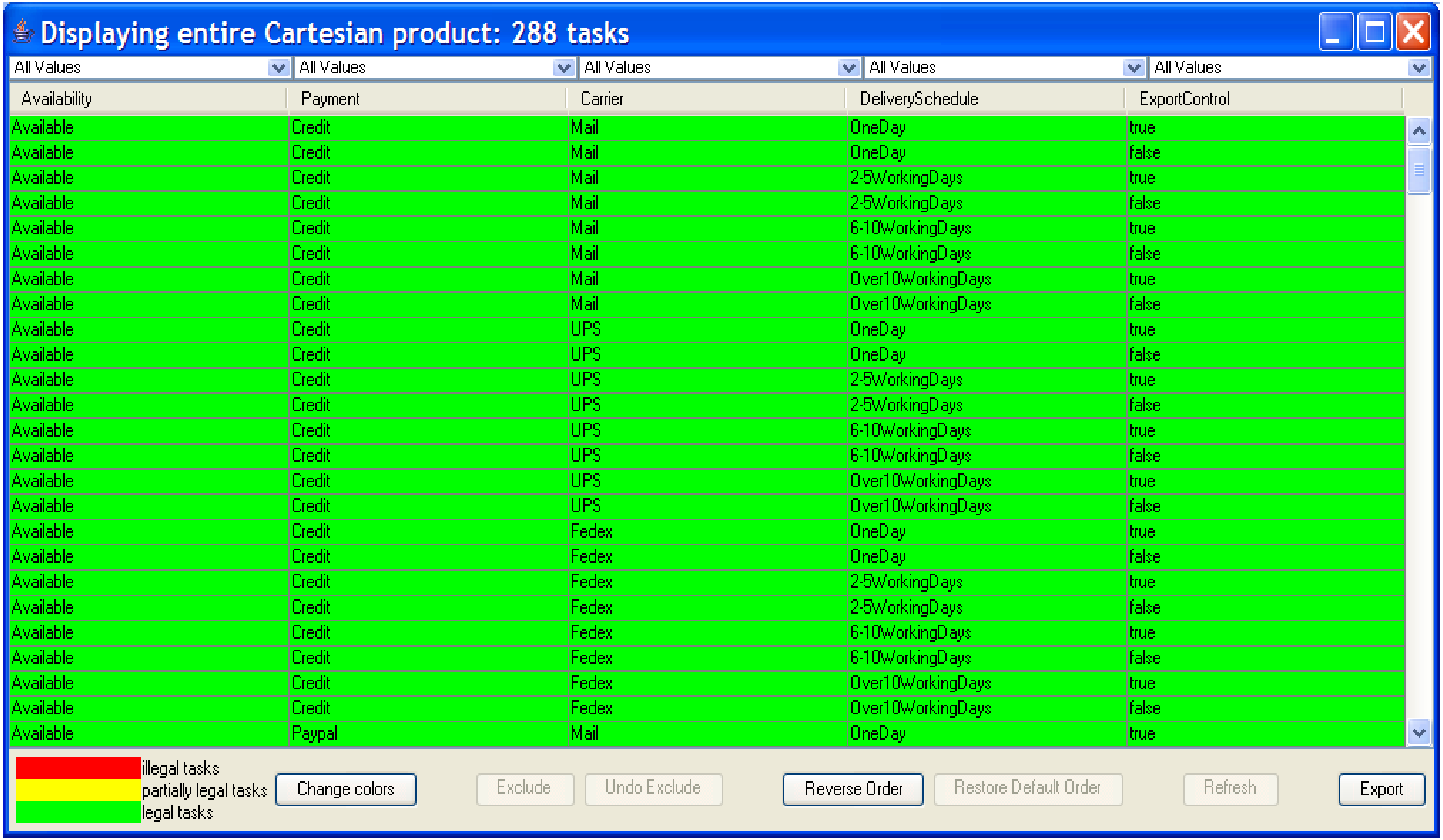
Typical restriction addition

- We will perform the following:
  - Create a Cartesian product report
  - Create a projection
  - Exclude combinations
  - Add a restriction explicitly
  - Generate a new CTD test plan

- Observe the changes in the number of legal and illegal combinations



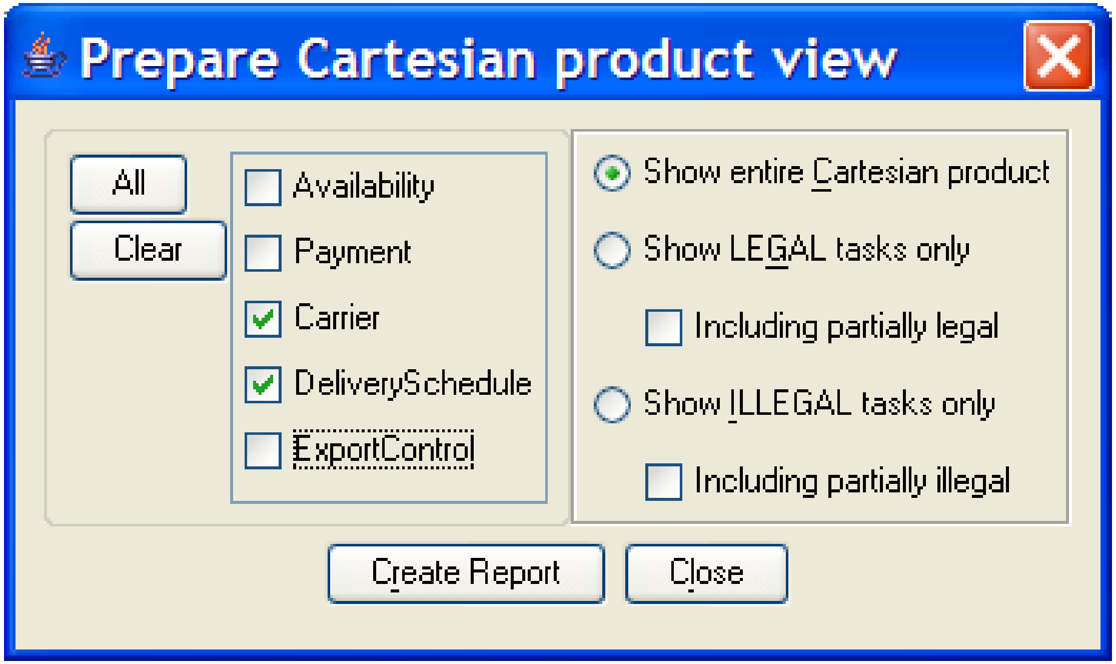
Cartesian product report – all 288 combinations are legal



Availability	Payment	Carrier	DeliverySchedule	ExportControl
Available	Credit	Mail	OneDay	true
Available	Credit	Mail	OneDay	false
Available	Credit	Mail	2-5WorkingDays	true
Available	Credit	Mail	2-5WorkingDays	false
Available	Credit	Mail	6-10WorkingDays	true
Available	Credit	Mail	6-10WorkingDays	false
Available	Credit	Mail	Over10WorkingDays	true
Available	Credit	Mail	Over10WorkingDays	false
Available	Credit	UPS	OneDay	true
Available	Credit	UPS	OneDay	false
Available	Credit	UPS	2-5WorkingDays	true
Available	Credit	UPS	2-5WorkingDays	false
Available	Credit	UPS	6-10WorkingDays	true
Available	Credit	UPS	6-10WorkingDays	false
Available	Credit	UPS	Over10WorkingDays	true
Available	Credit	UPS	Over10WorkingDays	false
Available	Credit	Fedex	OneDay	true
Available	Credit	Fedex	OneDay	false
Available	Credit	Fedex	2-5WorkingDays	true
Available	Credit	Fedex	2-5WorkingDays	false
Available	Credit	Fedex	6-10WorkingDays	true
Available	Credit	Fedex	6-10WorkingDays	false
Available	Credit	Fedex	Over10WorkingDays	true
Available	Credit	Fedex	Over10WorkingDays	false
Available	Paypal	Mail	OneDay	true

🟥 illegal tasks
🟨 partially legal tasks
🟩 legal tasks

Choose to view only part of the attributes (projection)



□ Availability

□ Payment

☑ Carrier

☑ DeliverySchedule

□ ExportControl

● Show entire Cartesian product

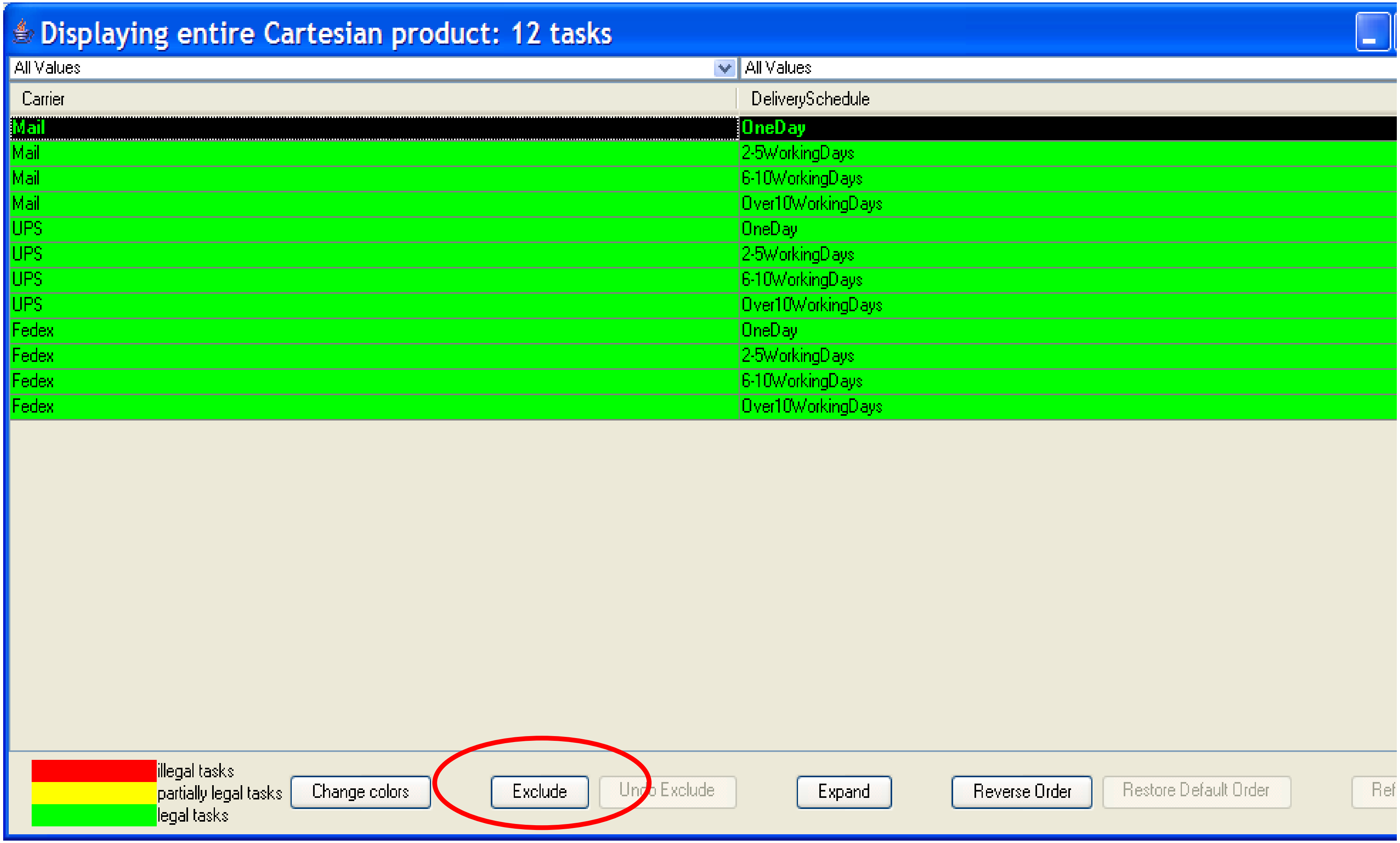
○ Show LEGAL tasks only

□ Including partially legal

○ Show ILLEGAL tasks only

□ Including partially illegal

12 value pairs in the projection of the selected attributes



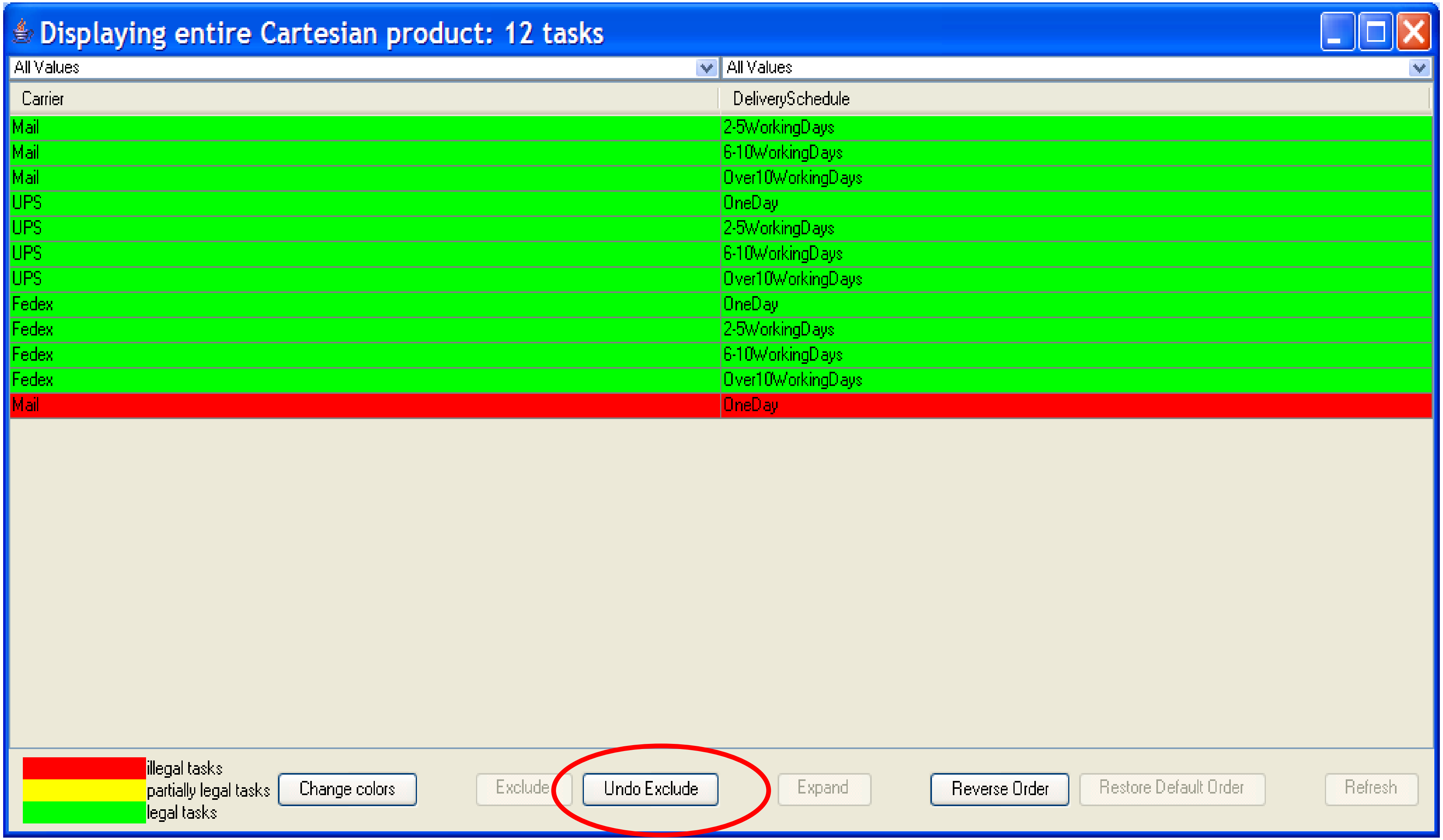
Displaying entire Cartesian product: 12 tasks

All Values | All Values

Carrier	DeliverySchedule
Mail	OneDay
Mail	2-5WorkingDays
Mail	6-10WorkingDays
Mail	Over10WorkingDays
UPS	OneDay
UPS	2-5WorkingDays
UPS	6-10WorkingDays
UPS	Over10WorkingDays
Fedex	OneDay
Fedex	2-5WorkingDays
Fedex	6-10WorkingDays
Fedex	Over10WorkingDays

🟥 illegal tasks
🟨 partially legal tasks
🟩 legal tasks

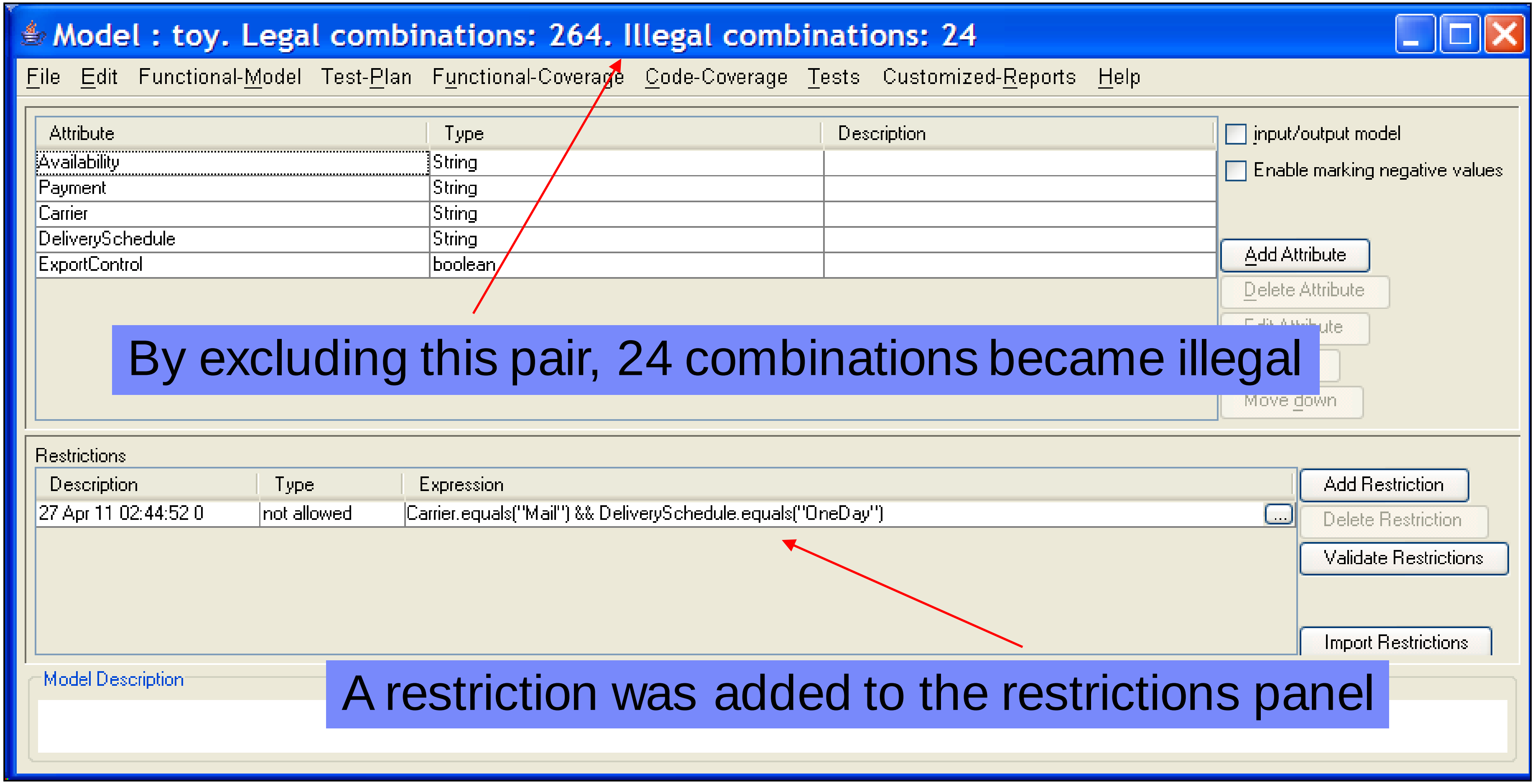
Excluding the invalid combination



Displaying entire Cartesian product: 12 tasks

Carrier	DeliverySchedule
All Values	All Values
Mail	2-5WorkingDays
Mail	6-10WorkingDays
Mail	Over10WorkingDays
UPS	OneDay
UPS	2-5WorkingDays
UPS	6-10WorkingDays
UPS	Over10WorkingDays
Fedex	OneDay
Fedex	2-5WorkingDays
Fedex	6-10WorkingDays
Fedex	Over10WorkingDays
Mail	OneDay

🟥 illegal tasks
🟨 partially legal tasks
🟩 legal tasks



Model : toy. Legal combinations: 264. Illegal combinations: 24

Attribute	Type	Description
Availability	String	
Payment	String	
Carrier	String	
DeliverySchedule	String	
ExportControl	boolean	

☐ input/output model

☐ Enable marking negative values

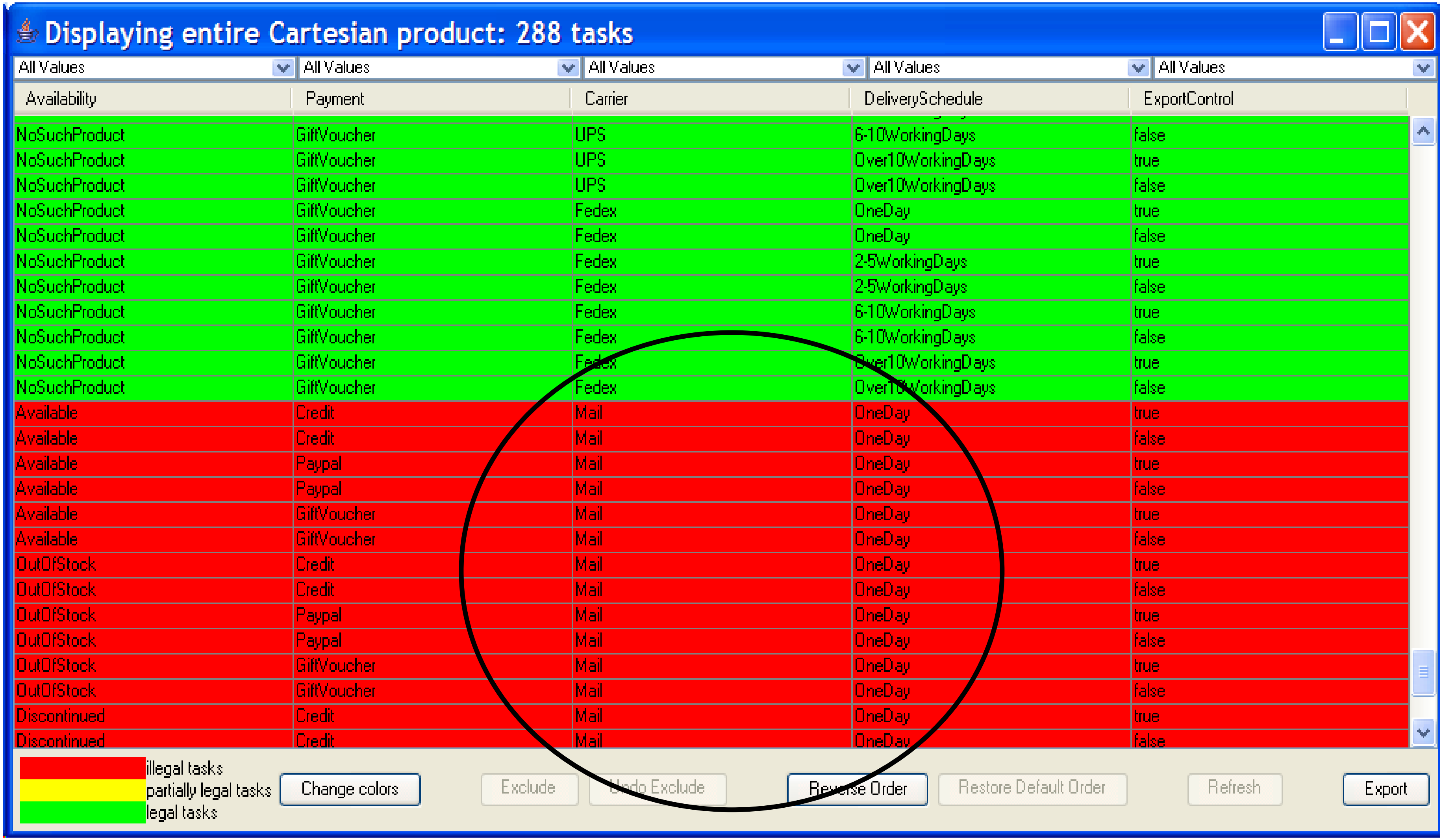
By excluding this pair, 24 combinations became illegal

Restrictions

Description	Type	Expression
27 Apr 11 02:44:52 0	not allowed	Carrier.equals("Mail") && DeliverySchedule.equals("OneDay")

A restriction was added to the restrictions panel

The Cartesian product displays all legal and illegal combinations

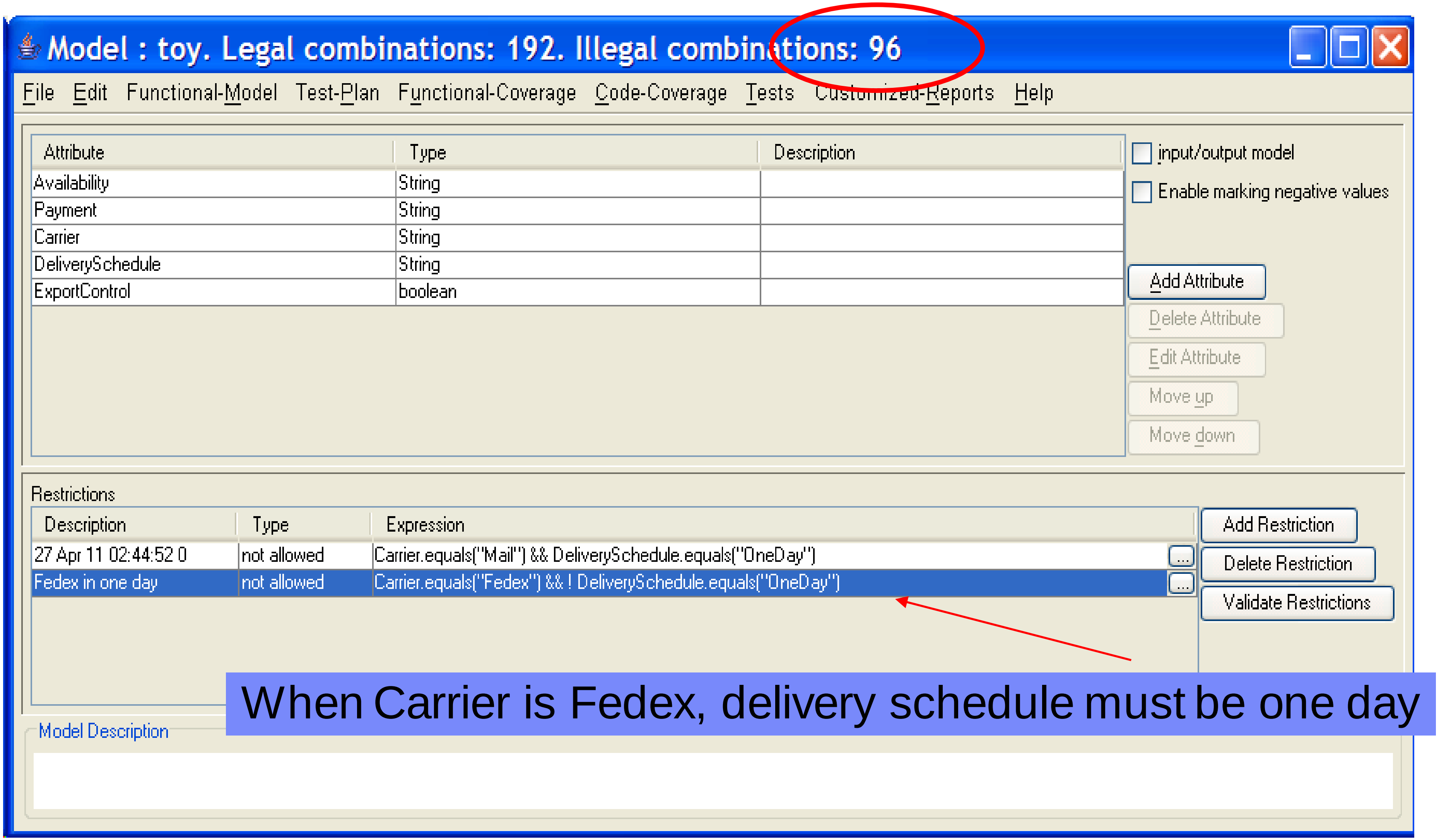


Displaying entire Cartesian product: 288 tasks

Availability	Payment	Carrier	DeliverySchedule	ExportControl
All Values	All Values	All Values	All Values	All Values
NoSuchProduct	GiftVoucher	UPS	6-10WorkingDays	false
NoSuchProduct	GiftVoucher	UPS	Over10WorkingDays	true
NoSuchProduct	GiftVoucher	UPS	Over10WorkingDays	false
NoSuchProduct	GiftVoucher	Fedex	OneDay	true
NoSuchProduct	GiftVoucher	Fedex	OneDay	false
NoSuchProduct	GiftVoucher	Fedex	2-5WorkingDays	true
NoSuchProduct	GiftVoucher	Fedex	2-5WorkingDays	false
NoSuchProduct	GiftVoucher	Fedex	6-10WorkingDays	true
NoSuchProduct	GiftVoucher	Fedex	6-10WorkingDays	false
NoSuchProduct	GiftVoucher	Fedex	Over10WorkingDays	true
NoSuchProduct	GiftVoucher	Fedex	Over10WorkingDays	false
Available	Credit	Mail	OneDay	true
Available	Credit	Mail	OneDay	false
Available	Paypal	Mail	OneDay	true
Available	Paypal	Mail	OneDay	false
Available	GiftVoucher	Mail	OneDay	true
Available	GiftVoucher	Mail	OneDay	false
OutOfStock	Credit	Mail	OneDay	true
OutOfStock	Credit	Mail	OneDay	false
OutOfStock	Paypal	Mail	OneDay	true
OutOfStock	Paypal	Mail	OneDay	false
OutOfStock	GiftVoucher	Mail	OneDay	true
OutOfStock	GiftVoucher	Mail	OneDay	false
Discontinued	Credit	Mail	OneDay	true
Discontinued	Credit	Mail	OneDay	false

🟥 illegal tasks
🟨 partially legal tasks
🟩 legal tasks

Explicitly adding a restriction to the model



Model : toy. Legal combinations: 192. Illegal combinations: 96

Attribute	Type	Description
Availability	String	
Payment	String	
Carrier	String	
DeliverySchedule	String	
ExportControl	boolean	

□ input/output model

□ Enable marking negative values

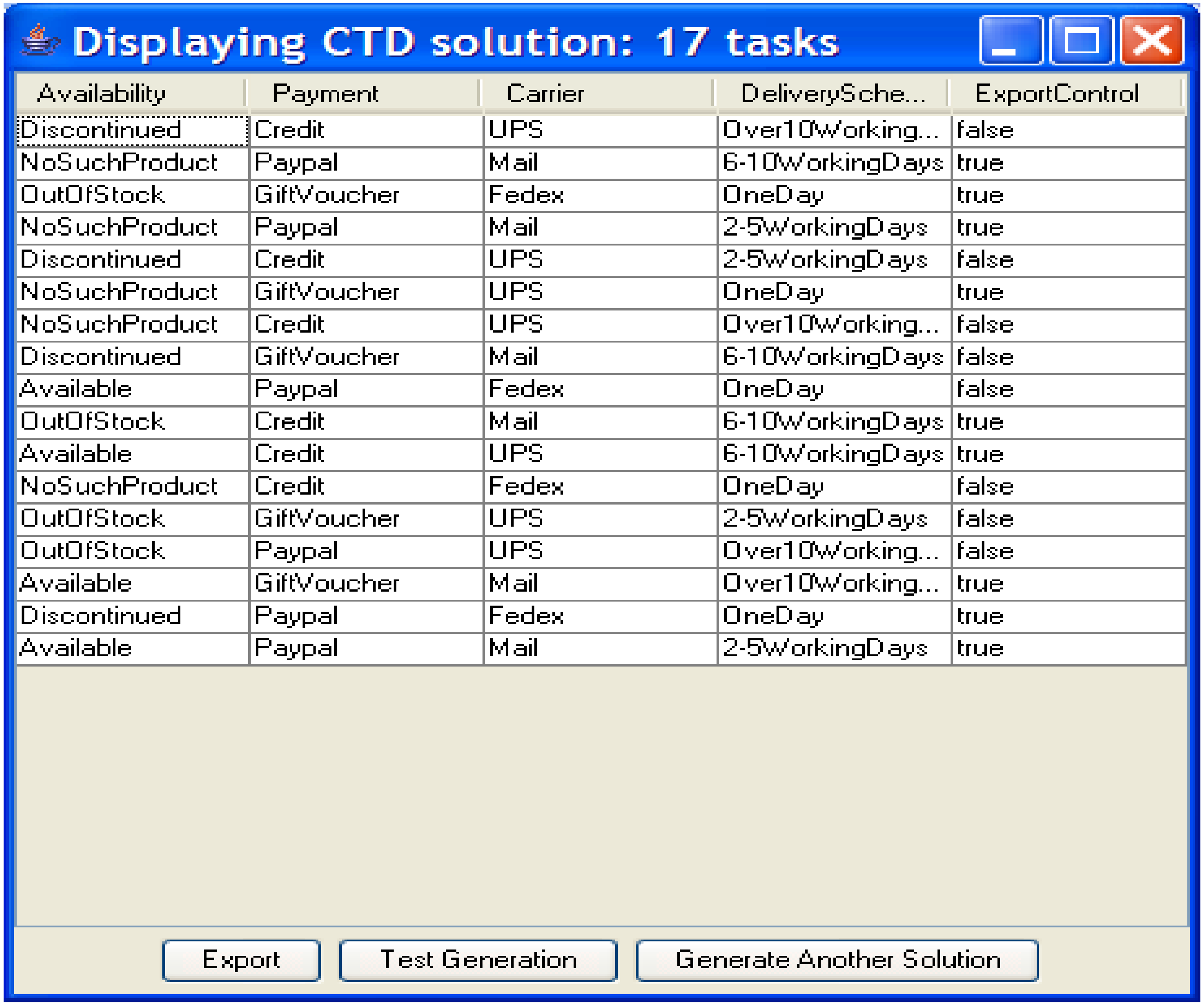
Restrictions

Description	Type	Expression
27 Apr 11 02:44:52 0	not allowed	Carrier.equals("Mail") && DeliverySchedule.equals("OneDay")
Fedex in one day	not allowed	Carrier.equals("Fedex") && ! DeliverySchedule.equals("OneDay")

Model Description

When Carrier is Fedex, delivery schedule must be one day

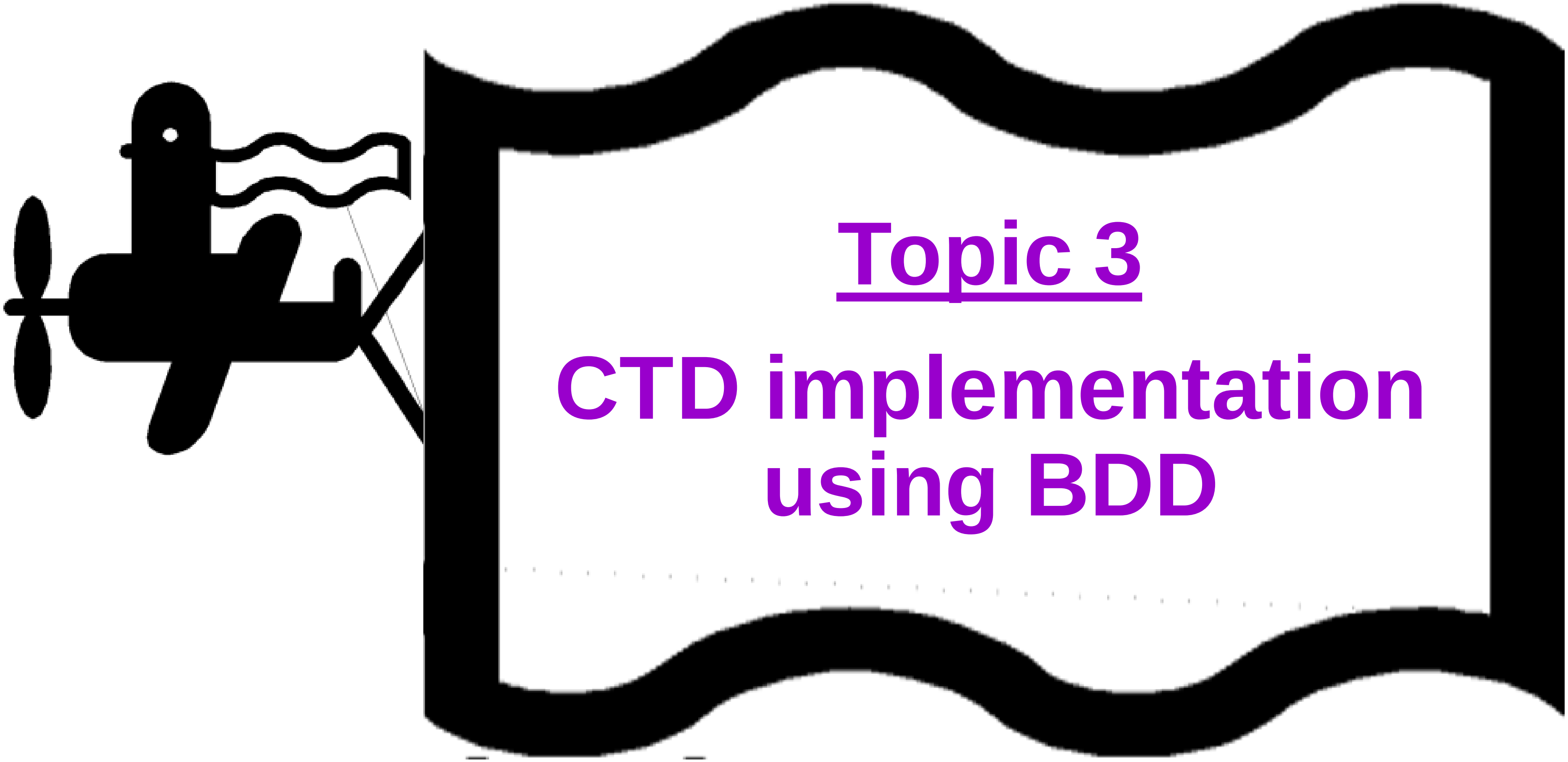
Complete pairwise coverage of the legal pairs



Availability	Payment	Carrier	DeliverySche...	ExportControl
Discontinued	Credit	UPS	Over10Working...	false
NoSuchProduct	Paypal	Mail	6-10WorkingDays	true
OutOfStock	GiftVoucher	Fedex	OneDay	true
NoSuchProduct	Paypal	Mail	2-5WorkingDays	true
Discontinued	Credit	UPS	2-5WorkingDays	false
NoSuchProduct	GiftVoucher	UPS	OneDay	true
NoSuchProduct	Credit	UPS	Over10Working...	false
Discontinued	GiftVoucher	Mail	6-10WorkingDays	false
Available	Paypal	Fedex	OneDay	false
OutOfStock	Credit	Mail	6-10WorkingDays	true
Available	Credit	UPS	6-10WorkingDays	true
NoSuchProduct	Credit	Fedex	OneDay	false
OutOfStock	GiftVoucher	UPS	2-5WorkingDays	false
OutOfStock	Paypal	UPS	Over10Working...	false
Available	GiftVoucher	Mail	Over10Working...	true
Discontinued	Paypal	Fedex	OneDay	true
Available	Paypal	Mail	2-5WorkingDays	true

… but the new test plan is larger!

- After adding restrictions there are less combinations to cover, but also more tests in the test plan (17 instead of 16) …
- This can happen since a test that previously covered many new combinations may now become illegal, and cannot be used
- In other words – we restricted the algorithm’s freedom of choice



Topic 3

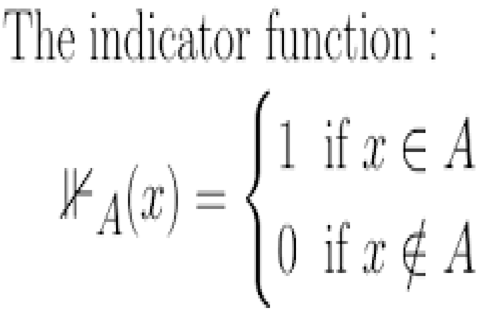
CTD implementation using BDD

An implementation of a CTD tool requires

- Model design
  - Explicit enumeration of the legal and illegal combination
  - Easy adding and removing of restrictions
- Test optimization
  - Finding an optimal set of legal tests for a given required interaction coverage
    - While utilizing existing tests
  - Analyzing the interaction coverage of existing tests
    - Remove redundancy in existing tests given some required interaction coverage

CTD tool design considerations

- The general problem of finding a minimal set of test cases that satisfies t-wise coverage is NP-complete
  - Instead, greedy heuristics are used
- Cartesian product subsets such as
  - Legal tests
  - Illegal tests
  - Projections
  - Interaction coverage requirements
  - Set of tests
  - Set of existing tests
- Can be represented as Boolean functions
  - Using an indicator function that
    - Represents part of the subset as “true”
  - Coding multiple values attributes using a binarity representation



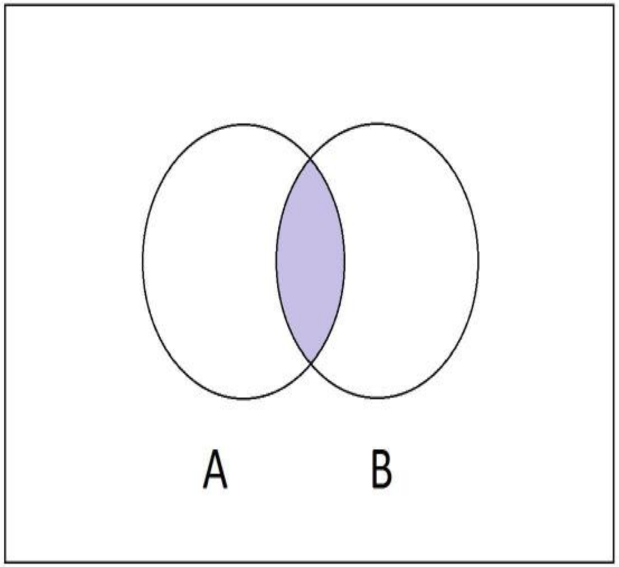
The indicator function :

$$\mathbb{1}_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$$

CTD tool design considerations (continued)

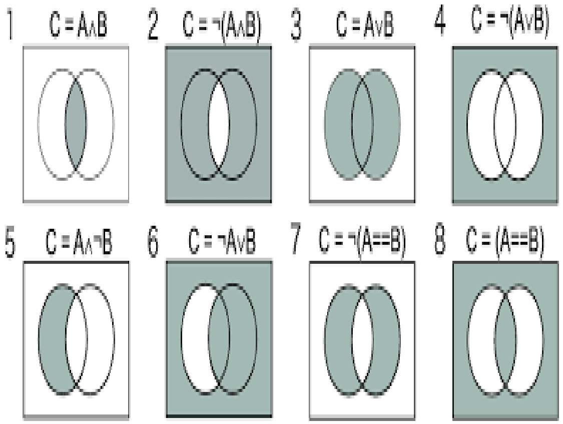
- We should also be able to efficiently preform operations on Boolean functions such as conjunction, negation and enumeration

- Example
  - Consider the Cartesian product of x1,x2, x3, x4 of type Boolean
  - The legal combinations are combinations with at least one 1 which is represented by the Boolean function A
    - E.g., 1011, 1001,…
  - We would like to obtain a legal test that covers the interaction x1 = 0 and x2 = 0 which is represented by the Boolean function B
    - We need to calculate the answer that is
    - 0010, 0001, 0011
    - Which can be obtained by calculating a conjunction of A and B and then enumerating it



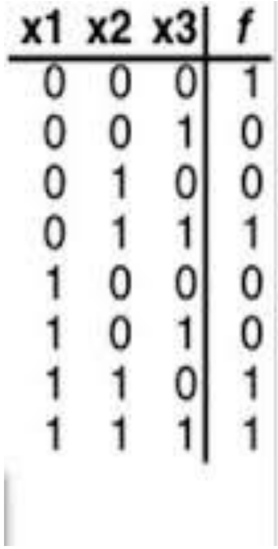
BDDs to the rescue

- BDDs are data structures that represent Boolean functions, and they efficiently support conjunction, negation, enumeration among other things
- Thus, BDDs can be used to provide a good implementation to a CTD tool

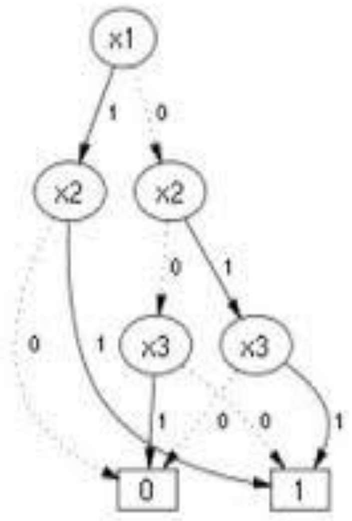


Binary Decision Diagram (BDD)

- Boolean function are functions from X1*…*Xn to X
  - Xi and X get values in {0, 1}
- BDD is a data structure that represents a Boolean function
- The Boolean variables are inspected in some fixed predetermined order
  - Hence the Directed Acyclic Graph (DAG) structure see figure on the right
  - It is customary to denote the order of variables inspection as x1 < x2 < x3



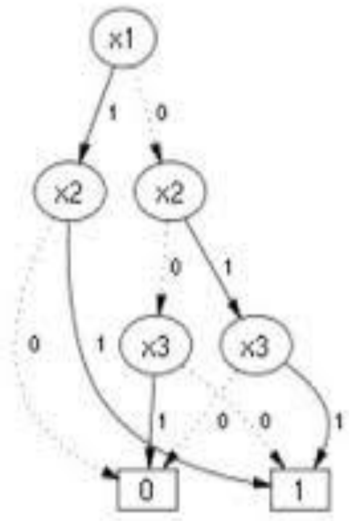
x1	x2	x3	*f*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Binary Decision Diagram (BDD) – continued

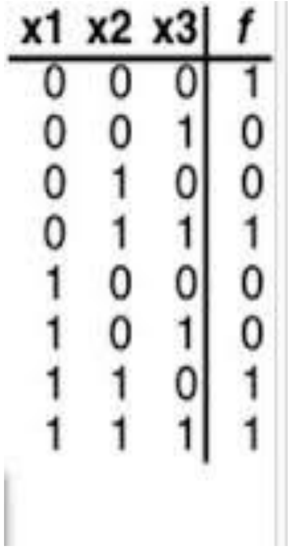
- If the outcome of the function is determined by the prefix of the variables already inspected, we "short cut" to the final answer
  - – See example on the right if x1=1, x2=1 the function outcome is 1
    - • Regardless of the value of x3
- Note the recursive structure of the BDD
  - – Denote by f(x1,…,xn) the Bollean function
  - – Note that the sub DAG reached by a 1 arrow from x1 is a BDD of the Boolean function f(x1,…,xn) in which x1 is known to be 1
    - • We denote that function as f(x1=1, x2,…xn) or f(x1=1) for short
  - – Similarly, the sub DAG reached by the 0 arrow from x1 is f(x1=0)

x1	x2	x3	*f*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

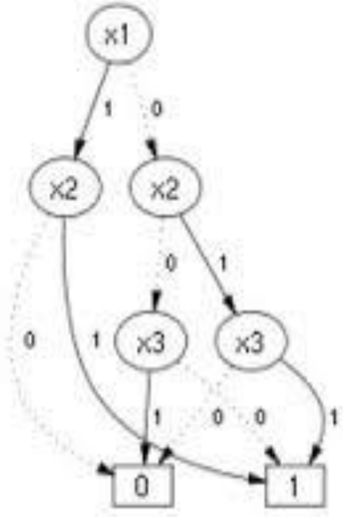


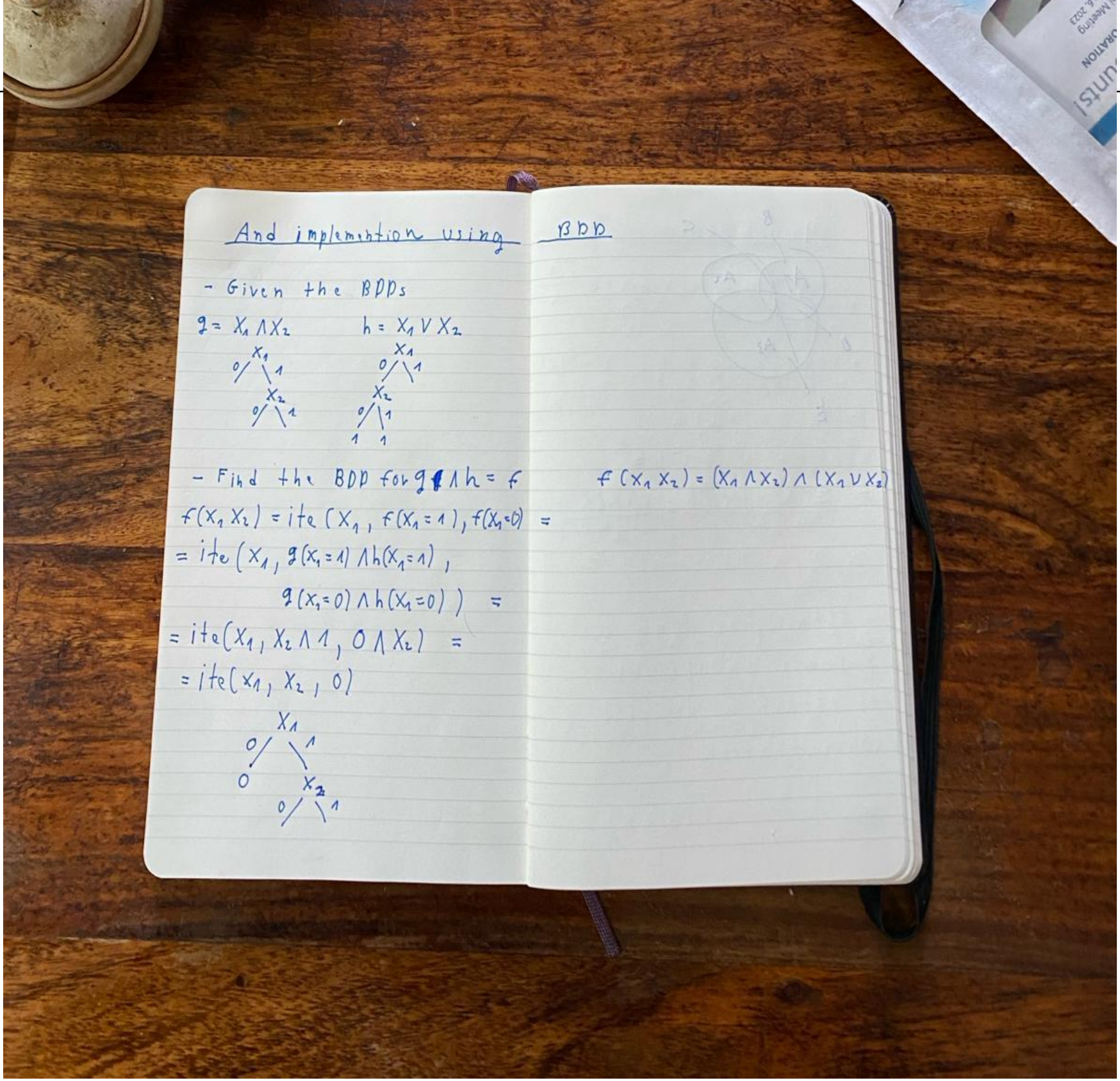
Shannon expansion

- The Shannon expansion highlights this relation algebraically
- f(x1,…,xn) = x1f(x1=1)+(not x1)f(x1=0)
  - If x1 = 1 then (not x1)=0 hence
    - (not x1)f(x1=0) = 0 in addition
  - x1f(x1=1) = 1f(x1=1) = f(x1=1) hence
    - x1f(x1=1)+(not x1)f(x1=0) =
    - f(x1=1)+0 = f(x1=1)
  - Similar proof for the case of x1 = 0 (prove)
- We sometimes refer to
  - x1f(x1=1)+(not x1)f(x1=0) as
  - Ite(x1, f(x1 = 1), f(x1 = 0))
    - Ite stands for “if then else”



x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1





X1
0
1
X2
0
1
X1
0
1
X2
0
1
1
1
X1
0
1
0
X2
0
1

And implemention using BDD

- Given the BDDs

$g = X_1 \wedge X_2$ $h = X_1 \vee X_2$

- Find the BDD for $g \wedge h = f$

$f(X_1 X_2) = (X_1 \wedge X_2) \wedge (X_1 \vee X_2)$

$f(X_1, X_2) = ite(X_1, f(X_1 = 1), f(X_1=0) =$

$= ite(X_1, g(X_1=1) \wedge h(X_1=1),$

$g(X_1=0) \wedge h(X_1=0)) =$

$= ite(X_1, X_2 \wedge 1, 0 \wedge X_2) =$

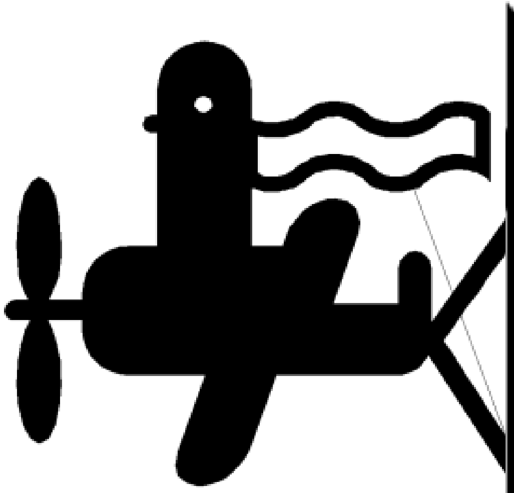
$= ite(x_1, X_2, 0)$

BDD desired features

- An important property of BDDs is that of canonicity: given a function, and an order for the variables, there is only one BDD that represents this function
- Due to the canonic representation, comparison of functions represented by BDDs may be performed in constant time
- Boolean operations such as conjunction, disjunction and negation can also be computed efficiently – negation in constant time, conjunction and disjunction in the worst case in time proportional to the product of the input BDD sizes
- Counting the number of satisfying assignments and iterating over them are other important examples of efficient operations
- Obtaining a satisfying assignment from a BDD is efficient

Deep dive

- See the following papers on CTD tool implantation using BDD Using Binary Decision Diagrams for Combinatorial Test Design.pdf - Google Drive
- See the following lecture for a deep dive on BDDs http://www.lsv.fr/~schwoon/enseignement/verification/ws0910/bdd.pdf



Topic 5

Creating a Model of the Points of Variability in a System, Using Interviews and Reviewing Documents

Creating a model of the points of variability in a system

- Reviewing documents
  - Any documentation can be valuable (e.g., requirement documents, existing test plans)
  - Understand the client's concerns and select which artifacts to focus accordingly
  - Concentrate on a specific component for proof of concept
  - Look for patterns and structured text in the documents (e.g., tables) - might be easier to extract initial models from them that will then be refined
  - When documents are unavailable, use interviews with the client to generate the initial model
    - Essential to get dedicated SMEs time

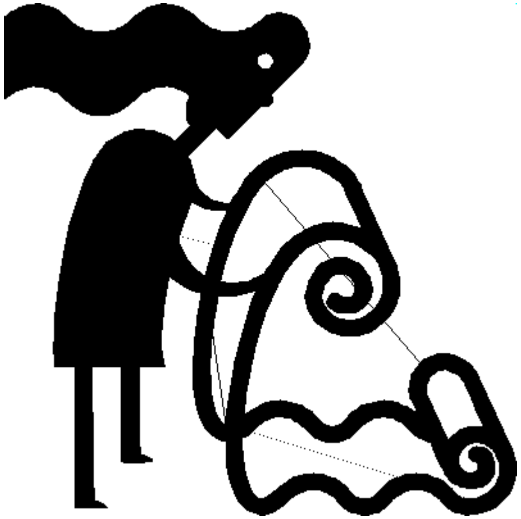
  - Following – some simple examples

- Using interviews
  - We will only point out questions that have to be raised

Common pitfalls in creating models

- Direct representation of parameters as attributes, which produces a huge model
- Modeling mutually exclusive values as separate attributes
- Assumptions on how things are implemented
- Missing attributes resulting in inability to determine whether a test succeeds or not
- Combinations that cannot be mapped into actual tests
- Attributes that cannot be controlled by the tester/user (as above – generate combinations that cannot be mapped into actual tests)

Case 1



Example of a requirements text

The system shall take orders for any valid item, whether it is in stock or not.

The system shall support multiple pricing schemes for an order.

- The first scheme ... [description omitted for demo]
- The second scheme ... [description omitted for demo]
- The third scheme ... [description omitted for demo]

The system shall validate the current credit status of the purchaser, when known.

The purchaser can select one of the following timeframes for order delivery: immediate, within one working week, and within one month. Ground shipping is default, while sea shipping is allowed for orders being delivered in a week or a month, and air shipping is allowed for immediate or one-week orders.

When an item is classified as export controlled, the system shall generate the appropriate work items to comply with governmental requirements.

Example of a requirements text

The system shall take orders for any valid i

Please spend a few minutes on identifying attributes, values, and restrictions

The system sh

c

c

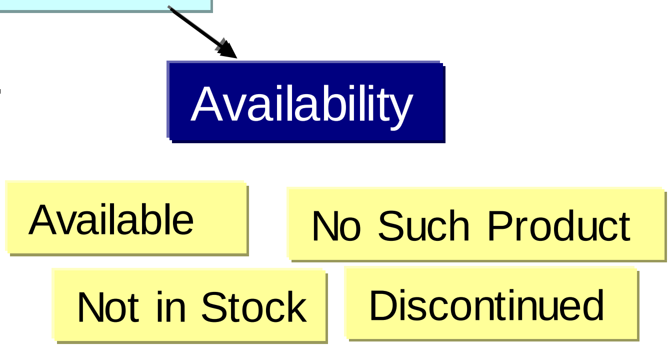
The ail validate the wn.

The purchaser can select one c nediate, within one
working week, and within one m ing is allowed for orders
being delivered in a week or a m one-week orders.

When an item is classified as expor priate work items to
comply with governmental requirem

Example of a requirements text – Availability Attribute and Values

1 The system shall take orders for any valid item, whether it is in stock or not.



The system shall support multiple pricing schemes for an order.

- The first scheme ... [description omitted for demo]
- The second scheme ... [description omitted for demo]
- The third scheme ... [description omitted for demo]

The system shall validate the current credit status of the purchaser, when known.

The purchaser can select one of the following timeframes for order delivery: immediate, within one working week, and within one month. Ground shipping is default, while sea shipping is allowed for orders being delivered in a week or a month, and air shipping is allowed for immediate or one-week orders.

When an item is classified as export controlled, the system shall generate the appropriate work items to comply with governmental requirements.

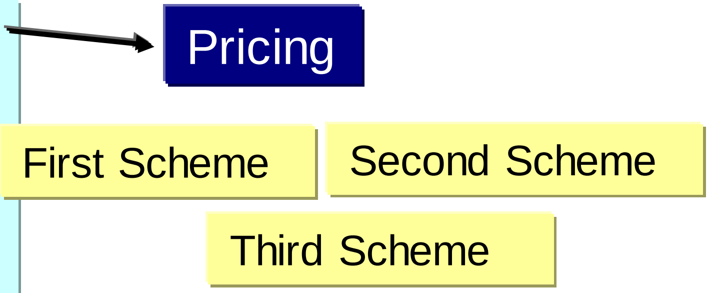
Example of a requirements text – Pricing Attribute and Values

The system shall take orders for any valid item, whether it is in stock or not.

2

The system shall support multiple pricing schemes for an order.

- o The first scheme ... [description omitted for demo]
- o The second scheme ... [description omitted for demo]
- o The third scheme ... [description omitted for demo]



The system shall validate the current credit status of the purchaser, when known.

The purchaser can select one of the following timeframes for order delivery: immediate, within one working week, and within one month. Ground shipping is default, while sea shipping is allowed for orders being delivered in a week or a month, and air shipping is allowed for immediate or one-week orders.

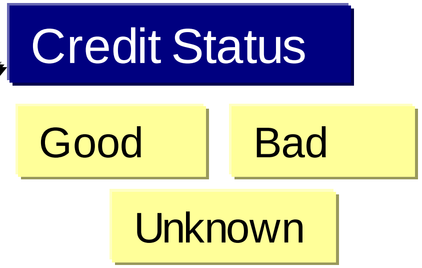
When an item is classified as export controlled, the system shall generate the appropriate work items to comply with governmental requirements.

Example of a requirements text – Credit Status Attribute and Values

The system shall take orders for any valid item, whether it is in stock or not.

The system shall support multiple pricing schemes for an order.

- o The first scheme ... [description omitted for demo]
- o The second scheme ... [description omitted for demo]
- o The third scheme ... [description omitted for demo]



3

The system shall validate the current credit status of the purchaser, when known.

The purchaser can select one of the following timeframes for order delivery: immediate, within one working week, and within one month. Ground shipping is default, while sea shipping is allowed for orders being delivered in a week or a month, and air shipping is allowed for immediate or one-week orders.

When an item is classified as export controlled, the system shall generate the appropriate work items to comply with governmental requirements.

Example of a requirements text – Additional Attributes and Values

The system shall take orders for any valid item, whether it is in stock or not.

The system shall support multiple pricing schemes for an order.

- The first scheme ... [description omitted for demo]
- The second scheme ... [description omitted for demo]
- The third scheme ... [description omitted for demo]

The system shall validate the current credit status of the purchaser, when known.

4

The purchaser can select one of the following timeframes for order delivery: immediate, within one working week, and within one month. Ground shipping is default, while sea shipping is allowed for orders being delivered in a week or a month, and air shipping is allowed for immediate or one-week orders.

6

7

5

When an item is classified as export controlled, the system shall generate the appropriate work items to comply with governmental requirements.

Restrictions

- Sea shipping is allowed for orders being delivered in a week or a month
- Air shipping is allowed for immediate or one-week orders

Checkpoint

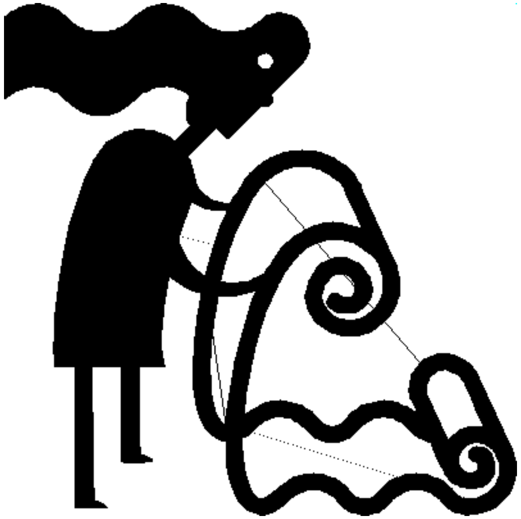
Ask yourself:
Do you know how to do this in a CTD tool?

Discussion – Questions to Subject Matter Experts (SMEs)

- What would you consult with an SME in this document / model?

- Some examples:
  - Is the text we used up to date? Are there other attributes or values?
  - What time frame is valid for an item that is not in stock?
  - Does the credit status have an effect on anything? If not, why is it there?

Case 2



An example of a real requirement

The NPIV WWPN Assignment Table (WAT) can become corrupted.

Two situations exist:

- The WAT in Hardware Storage Area (HSA) has become corrupted.
- The WAT on the Service Element (SE) has become corrupted.

A repair operation identifies a non corrupted WAT and copies it to overwrite the corrupted WAT.

When a repair operation occurs:

- No other repair operation is allowed
- No I/O operation is allowed
- No WAT save operation is allowed

An example of a real requirement

The NPIV WWPN Assignment Tab

Two situ

- corrupted.
- T pted.

A repa eration identifi to overwrite the corrupted WAT.

When a repair operation occ

- No other repair operation
- No I/O operation is allowed
- No WAT save operation is a

Please spend a few minutes on identifying attributes, values, and restrictions

An example of a real requirement

The NPIV WWPN Assignment Table (WAT) can become corrupted.

Two situations exist:

- The WAT in Hardware Storage Area (HSA) has become corrupted.
- The WAT on the Service Element (SE) has become corrupted.

A repair operation identifies a non corrupted WAT and copies it to overwrite the corrupted WAT.

When a repair operation occurs:

- No other repair operation is allowed
- No I/O operation is allowed
- No WAT save operation is allowed

Discussion – Questions to Subject Matter Experts (SMEs)

- Are these requirements well defined?

An example of a real requirement – Questions for SMEs

The NPIV WWPN Assignment Table (WAT) can become corrupted.

Two situations exist:

- The WAT in Hardware Storage Area (HSA) has become corrupted.
- The WAT on the Service Element (SE) has become corrupted.

A repair operation identifies a non corrupted WAT and copies it to overwrite the corrupted WAT.

- What is the expected result if repair is invoked when both WATs are good?

When a repair operation occurs:

- No other repair operation is allowed
  - 2nd repair operation should fail, or both?
- No I/O operation is allowed
  - I/O should fail, or repair should fail, or both?
- No WAT save operation is allowed
  - save should fail, or repair should fail, or both?

We Model a Given Repair Operation

Attributes and Values:

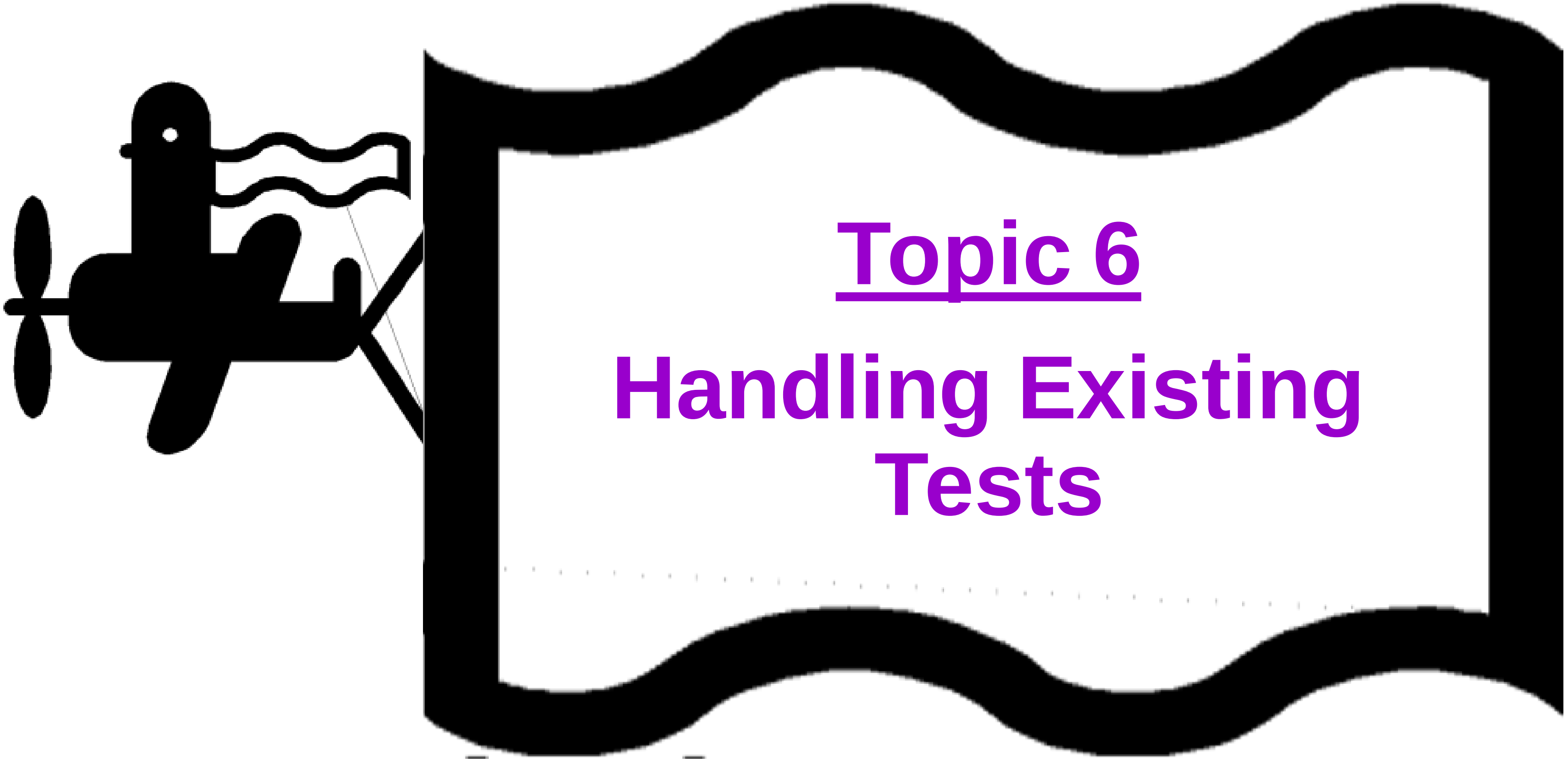
- WAT_HSA: Corrupted/OK
- WAT_SE: Corrupted/OK
- DynamicIOOngoing: No/Yes or StartedBefore/StartedDuring/None
- SaveWATOngoing: No/Yes or StartedBefore/StartedDuring/None
- WATRepair: No/Yes (another parallel repair) or StartedBefore/StartedDuring/None
- ReturnStatus (for the Repair Operation): S/F (success/fail)

Use restrictions to correctly correlate to the other attributes

- Which of the values is negative?
  - Depends on SME answers!

Checkpoint

Ask yourself:
Do you know how to do this in a CTD tool?



Topic 6

Handling Existing Tests

A common customer reaction is:

“ Are you kidding? I have so many tests already. Just throw them away? ”

Another common reaction is:

“ It is impossible to generate new test data, you must use only existing tests ”

Existing Tests

- Some customers have trusted tests suits in which they invested heavily
- In all likelihood, there is much redundancy in these test suites
- On the other side, there are usually large coverage gaps in them

Example

- Features:
  - X with values (1, 2, 3),
  - Y with values (4, 5, 6),
  - Z with values (7, 8, 9)
- All pairs required
- Existing test suite:
  a) 1 4 7
  b) 2 4 7
  c) 2 4 9
  d) 1 4 9
  e) 1 5 9
- What is covered?
- What is not covered?
- What is redundant?

Example

- Features:
  - X with values (1, 2, 3),
  - Y with values (4, 5, 6),
  - Z with values (7, 8, 9)
- All pairs required
- Existing test suite:

  a) 1 4 7

  b) 2 4 7

  c) 2 4 9

  d) 1 4 9 ← redundant: (1,4) from (a), (1,9) from (e), (4,9) from (c)

  e) 1 5 9
- Covered pairs (1,4), (1,5), (1,7), (1, 9) (2,4), (2,7), (2,9), (4,7), (4,9), (5,9)
- Not all pairs are covered

We will learn…

- We will learn three (complementary) approaches to handling existing tests:

Hole Analysis
– or –
“Let me show you what you’re not covering”

Augmentation
– or –
“Let me help you cover the gaps I just found for you”

Test Selection
– or –
“You have too many tests, let me choose for you which ones to use”

Traces

- All of the above approaches require representing the existing tests as traces
- A trace is just like a test plan:
  - A list of tests
  - A test is represented by assigning a value to each attribute
- This is the main obstacle in applying these approaches to real-life engagements – existing tests have to be given in a form close enough to traces (or at least machine-readable)

Traces – Format option 1 – trace files

```
Available Paypal Fedex OneDay true
Available GiftVoucher Mail OneDay true
Available GiftVoucher UPS OneDay false
Available GiftVoucher Fedex 2-5WorkingDays true
OutOfStock Credit Mail OneDay false
OutOfStock Credit UPS 2-5WorkingDays true
OutOfStock Credit UPS 6-10WorkingDays false
OutOfStock Credit UPS Over10WorkingDays false
OutOfStock Credit Fedex OneDay true
OutOfStock Credit Fedex Over10WorkingDays true
OutOfStock Paypal Mail OneDay true
OutOfStock Paypal UPS OneDay false
Discontinued Credit Mail OneDay true
Discontinued Credit Fedex 2-5WorkingDays false
Discontinued GiftVoucher Mail OneDay true
NoSuchProduct Credit Mail OneDay true
NoSuchProduct Paypal UPS OneDay false
```

Blanks, not tabs

Booleans are lower case

Traces – Format option 2 – csv files

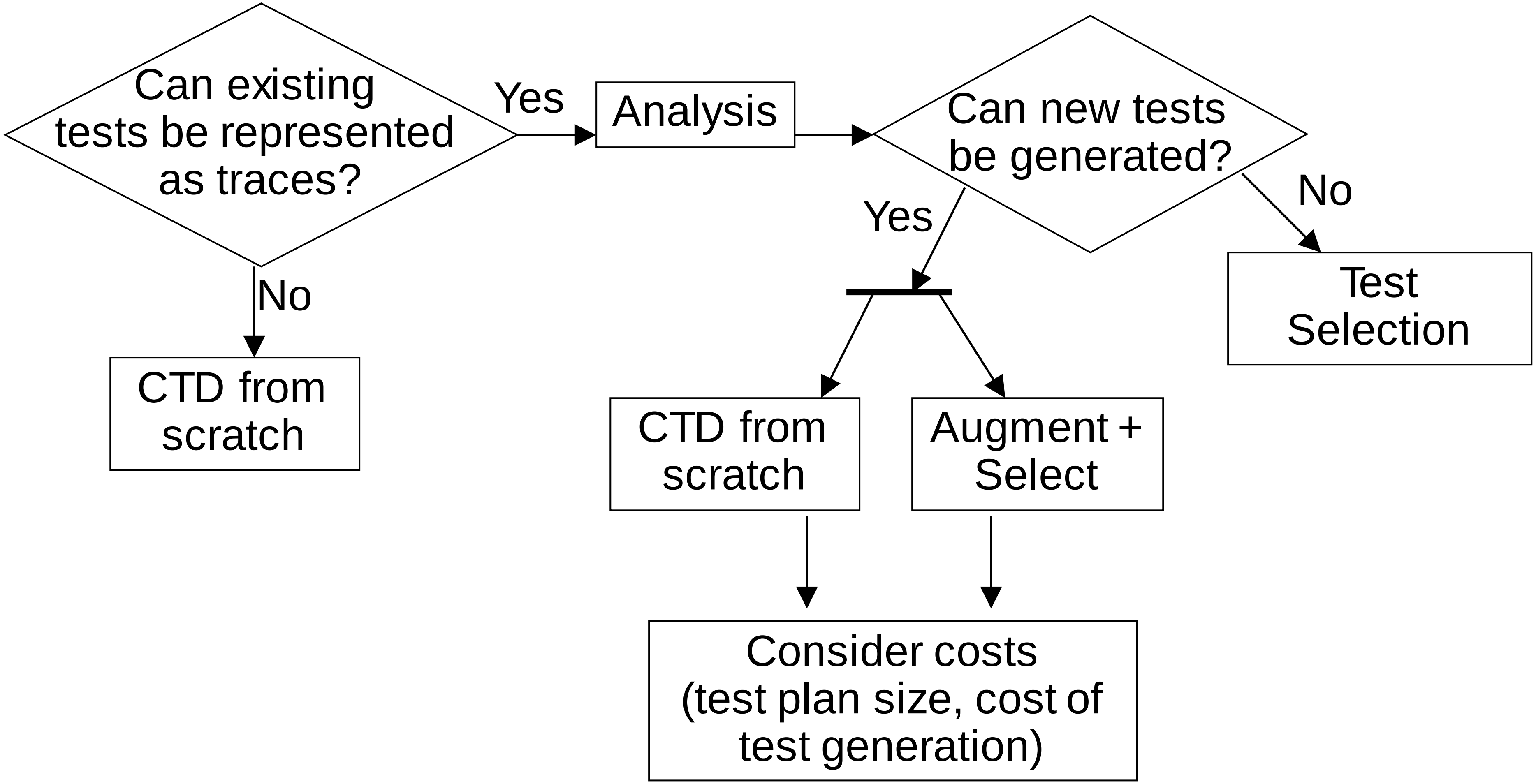
```
Avialability,Payment,Carrier,DeliverySchedule,ExportControl
Available,Paypal,Fedex,OneDay,true
Available,GiftVoucher,Mail,OneDay,true
Available,GiftVoucher,UPS,OneDay,false
Available,GiftVoucher,Fedex,2-5WorkingDays,true
OutOfStock,Credit,Mail,OneDay,false
OutOfStock,Credit,UPS,2-5WorkingDays,true
OutOfStock,Credit,UPS,6-10WorkingDays,false
OutOfStock,Credit,UPS,Over10WorkingDays,false
OutOfStock,Credit,Fedex,OneDay,true
OutOfStock,Credit,Fedex,Over10WorkingDays,true
OutOfStock,Paypal,Mail,OneDay,true
OutOfStock,Paypal,UPS,OneDay,false
Discontinued,Credit,Mail,OneDay,true
Discontinued,Credit,Fedex,2-5WorkingDays,false
Discontinued,GiftVoucher,Mail,OneDay,true
NoSuchProduct,Credit,Mail,OneDay,true
NoSuchProduct,Paypal,UPS,OneDay,false
```

First line is attribute names

Commas between values

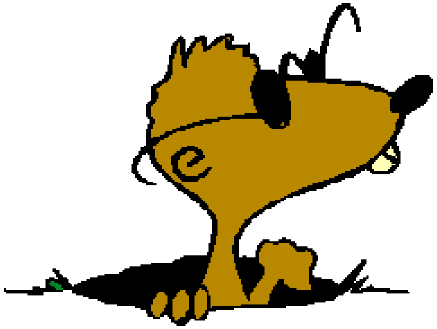
Booleans are lower case

Choosing an Approach

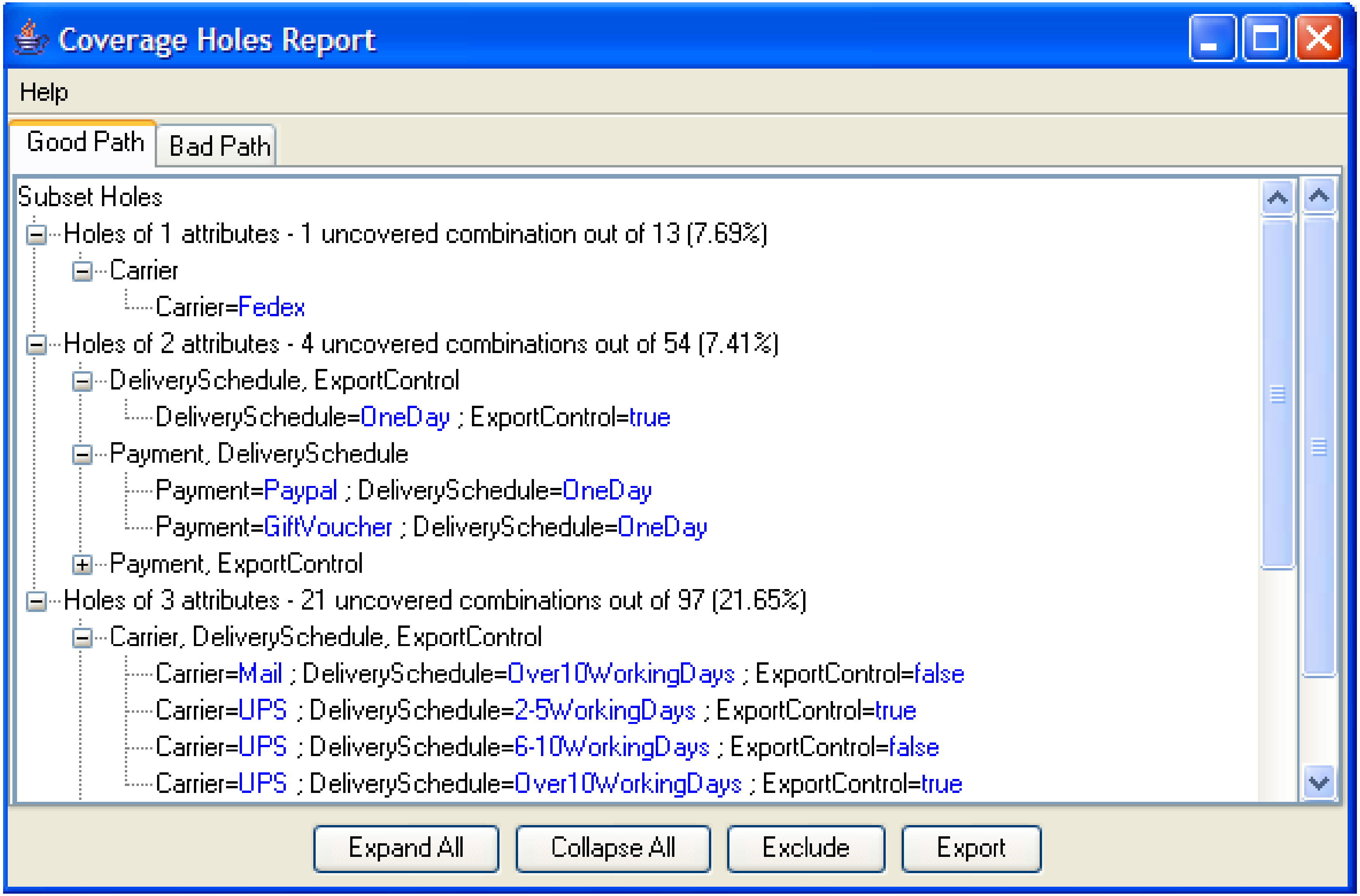

Can existing tests be represented as traces?
Yes
Analysis
Can new tests be generated?
No
CTD from scratch
Yes
No
Test Selection
CTD from scratch
Augment + Select
Consider costs (test plan size, cost of test generation)

Hole Analysis

- Input:
  - Model
  - Trace
  - Interaction Requirements
- Output:
  - Required combinations that are not covered by the tests in the trace



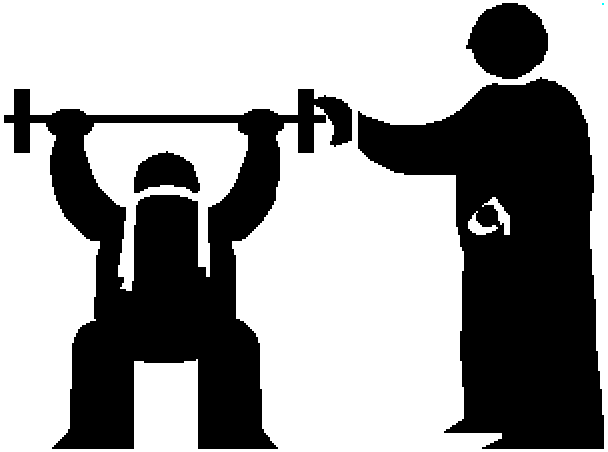
Hole Analysis



```
Subset Holes
⊟ Holes of 1 attributes - 1 uncovered combination out of 13 (7.69%)
    ⊟ Carrier
        Carrier=Fedex
⊟ Holes of 2 attributes - 4 uncovered combinations out of 54 (7.41%)
    ⊟ DeliverySchedule, ExportControl
        DeliverySchedule=OneDay ; ExportControl=true
    ⊟ Payment, DeliverySchedule
        Payment=Paypal ; DeliverySchedule=OneDay
        Payment=GiftVoucher ; DeliverySchedule=OneDay
    ⊞ Payment, ExportControl
⊟ Holes of 3 attributes - 21 uncovered combinations out of 97 (21.65%)
    ⊟ Carrier, DeliverySchedule, ExportControl
        Carrier=Mail ; DeliverySchedule=Over10WorkingDays ; ExportControl=false
        Carrier=UPS ; DeliverySchedule=2-5WorkingDays ; ExportControl=true
        Carrier=UPS ; DeliverySchedule=6-10WorkingDays ; ExportControl=false
        Carrier=UPS ; DeliverySchedule=Over10WorkingDays ; ExportControl=true
```

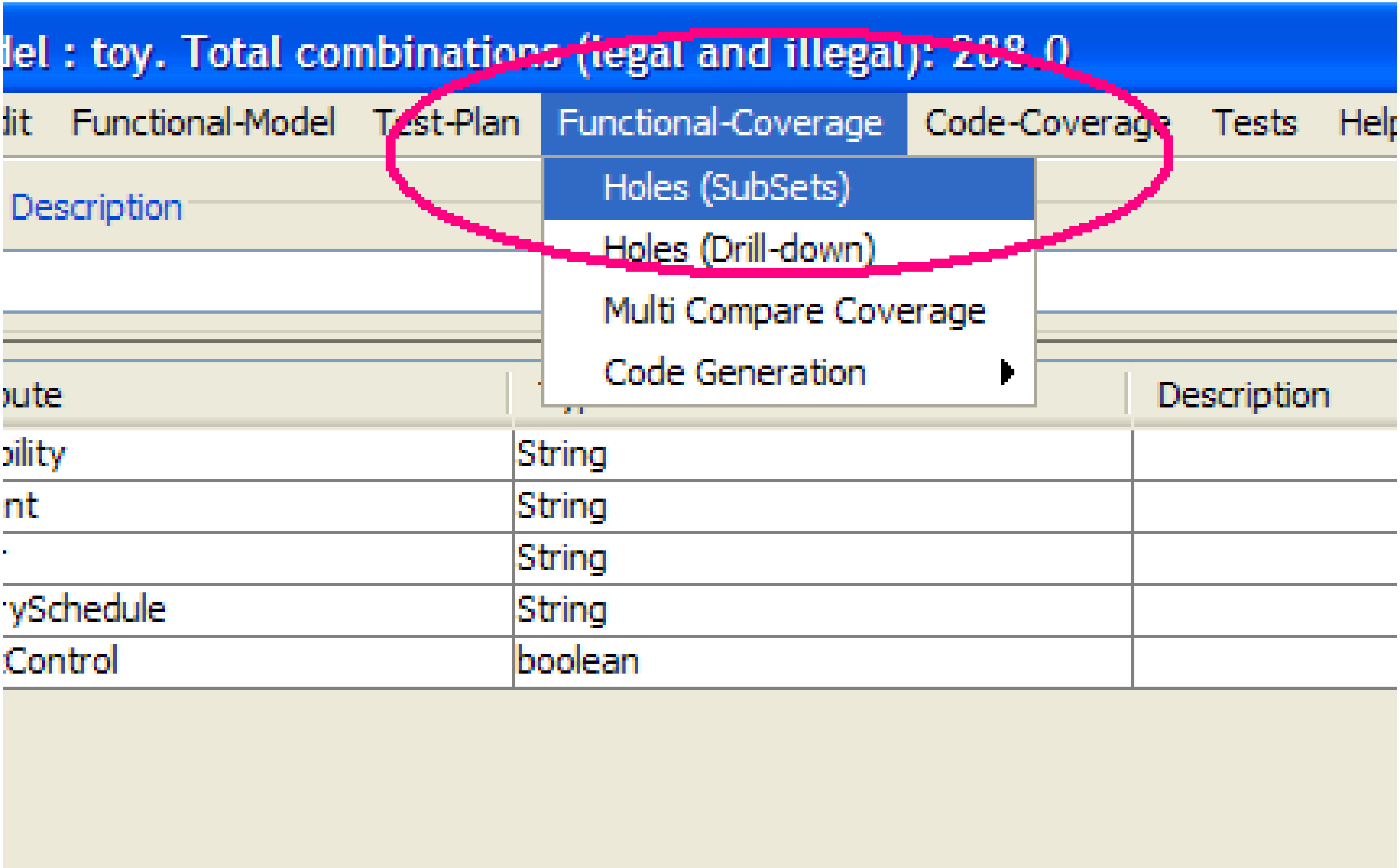
Typical hole analysis generation with CTD tools

- We will perform the following:
  - –Generate a hole analysis for a “customer” trace



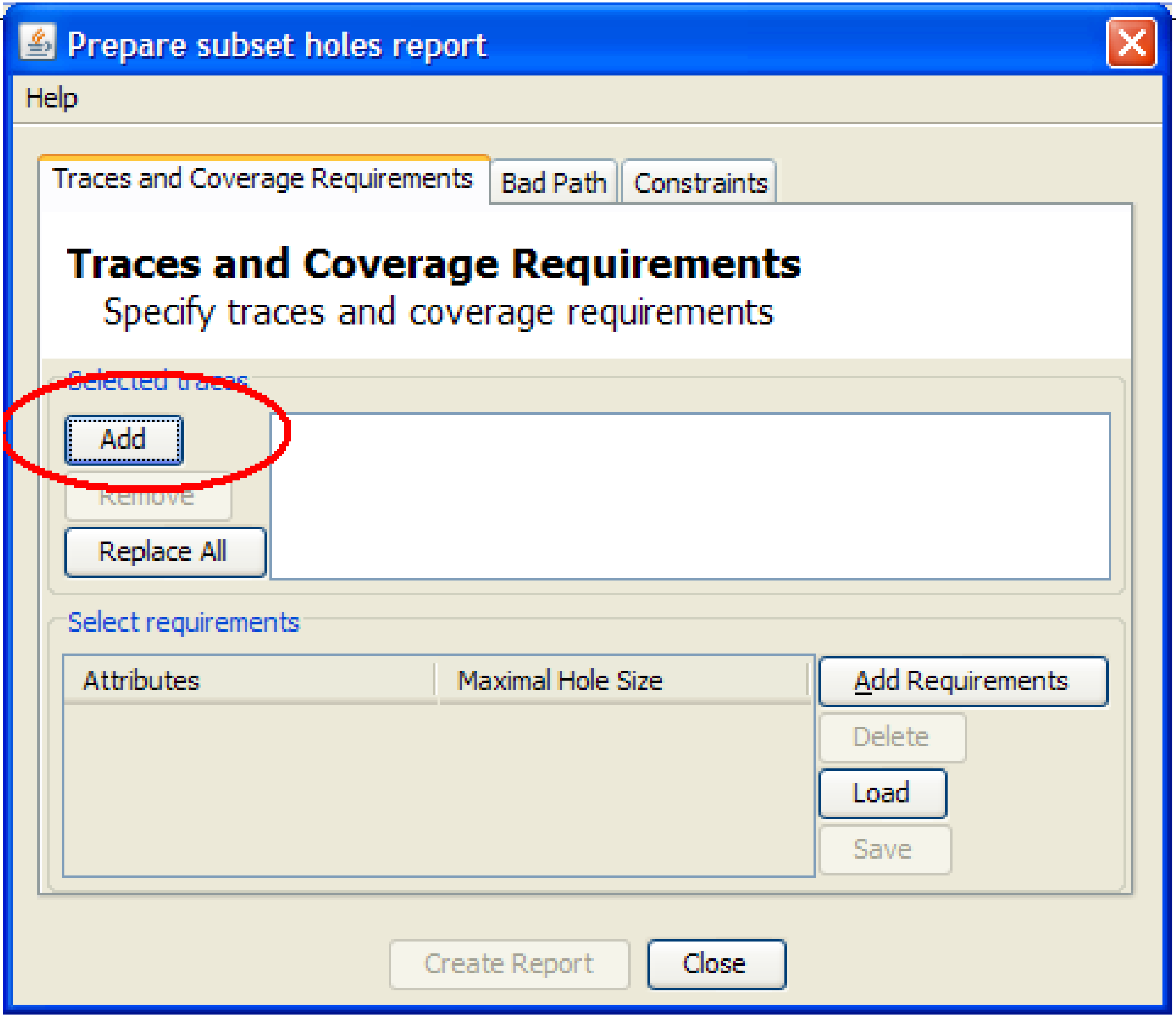
The Trace

```
Available Credit Mail 2-5WorkingDays true
Available Credit UPS OneDay true
Available Credit UPS OneDay false
Available Credit UPS 6-10WorkingDays false
Available Credit Fedex OneDay true
Available Credit Fedex OneDay false
Available Paypal Mail 6-10WorkingDays true
Available Paypal UPS OneDay true
Available Paypal UPS OneDay false
Available Paypal UPS 6-10WorkingDays false
Available GiftVoucher Mail 2-5WorkingDays true
Available GiftVoucher UPS 6-10WorkingDays true
Available GiftVoucher Fedex OneDay false
NotInStock Credit UPS OneDay false
NotInStock Credit UPS 6-10WorkingDays false
NotInStock Paypal Mail Over10WorkingDays true
NotInStock Paypal UPS 2-5WorkingDays false
NotInStock Paypal Fedex OneDay false
NotInStock GiftVoucher UPS OneDay true
NotInStock GiftVoucher UPS Over10WorkingDays true
Discontinued Credit Mail 6-10WorkingDays true
Discontinued Credit UPS 2-5WorkingDays true
Discontinued Credit Fedex OneDay true
Discontinued Paypal Mail Over10WorkingDays true
Discontinued Paypal UPS Over10WorkingDays false
Discontinued GiftVoucher Mail Over10WorkingDays true
NoSuchProduct Credit Mail 2-5WorkingDays false
NoSuchProduct Credit UPS 6-10WorkingDays false
NoSuchProduct Paypal UPS 6-10WorkingDays false
NoSuchProduct GiftVoucher UPS 2-5WorkingDays true
```



lel : toy. Total combinations (legal and illegal): 288.0

ute		Description
ility	String	
nt	String	
·	String	
ySchedule	String	
Control	boolean	

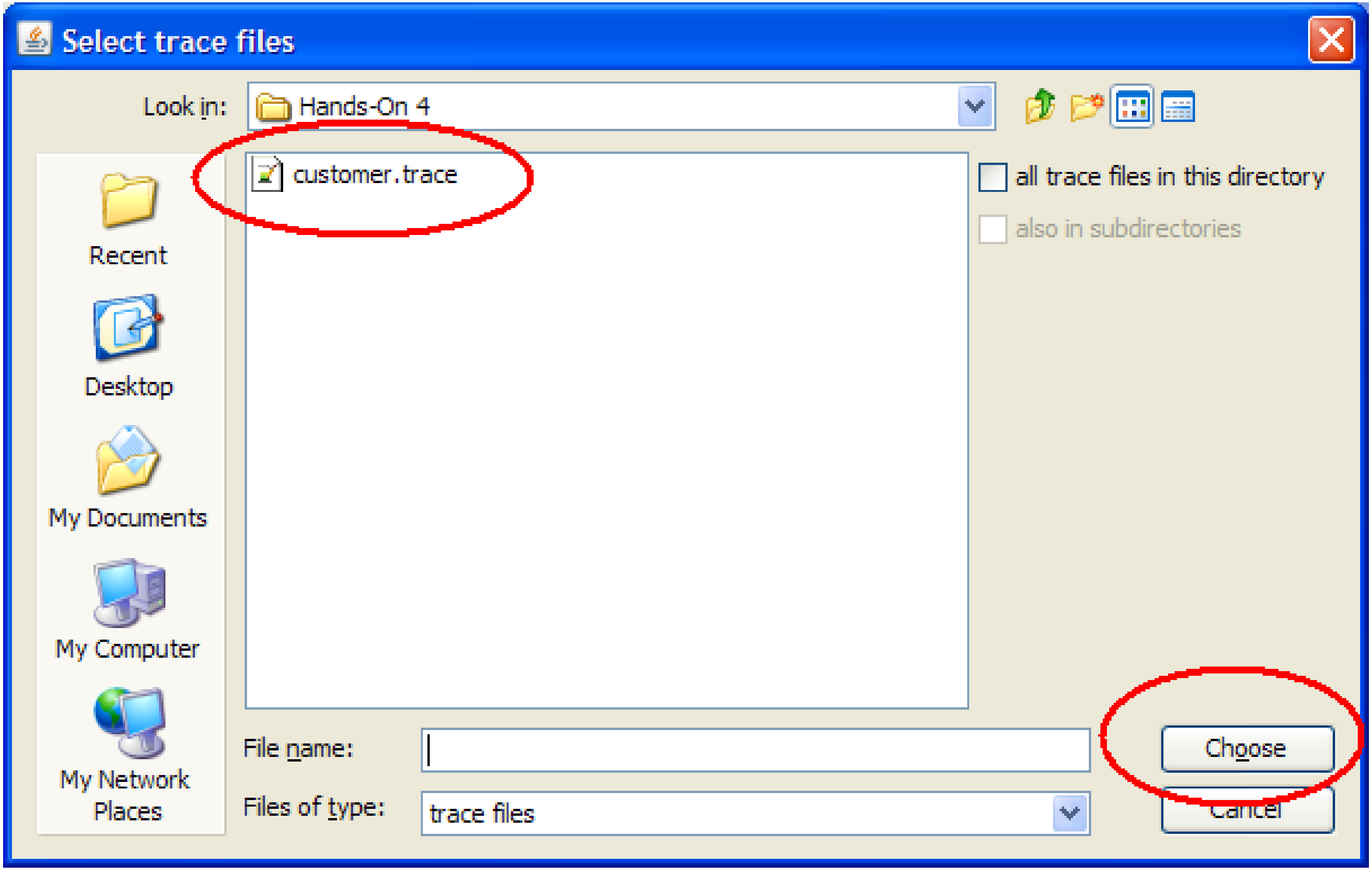

Prepare subset holes report

Help

Traces and Coverage Requirements | Bad Path | Constraints

Traces and Coverage Requirements

Specify traces and coverage requirements



Select trace files

Look in: Hands-On 4

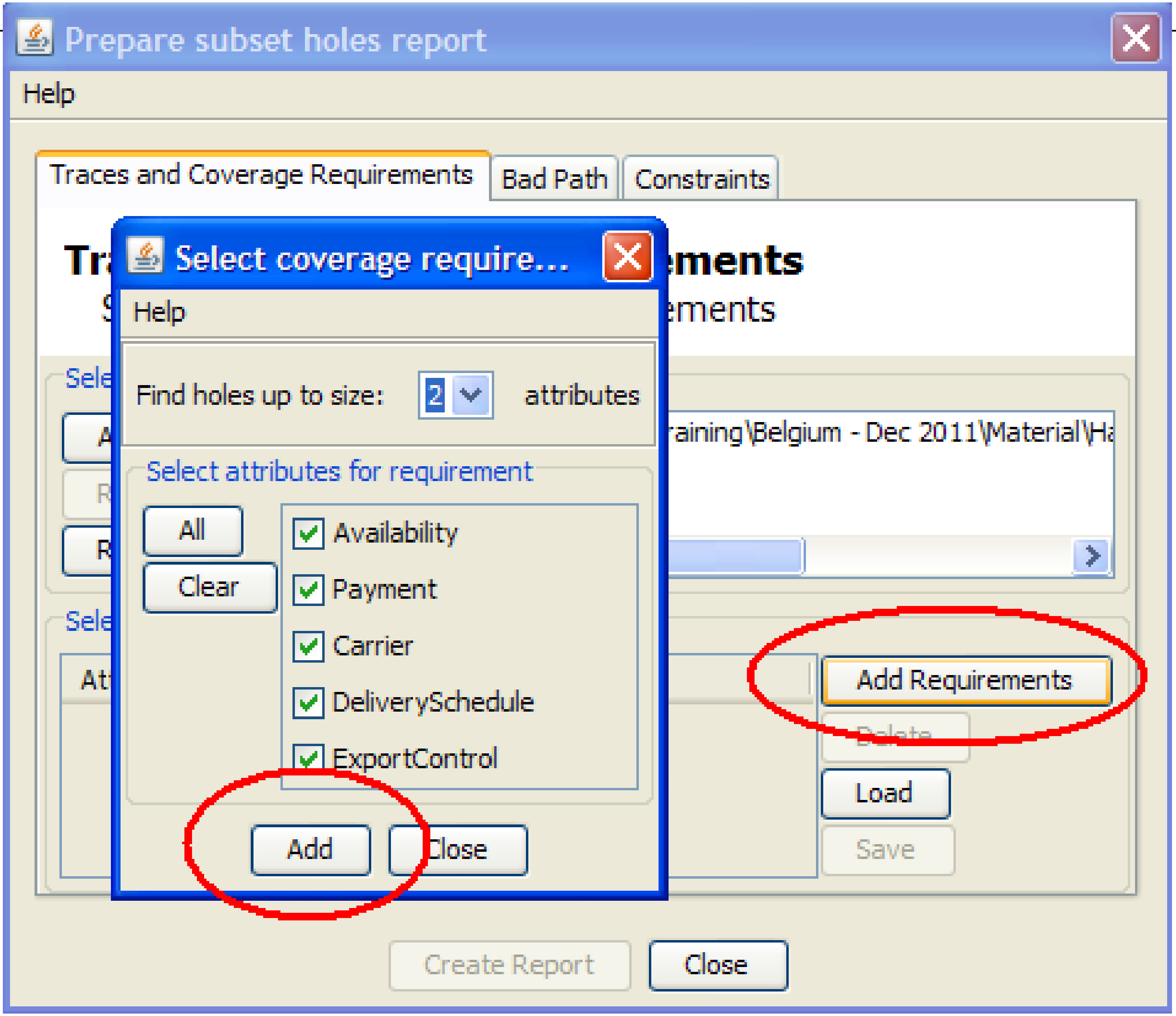
customer.trace

□ all trace files in this directory

□ also in subdirectories

File name:

Files of type: trace files



Prepare subset holes report

Help

Traces and Coverage Requirements | Bad Path | Constraints

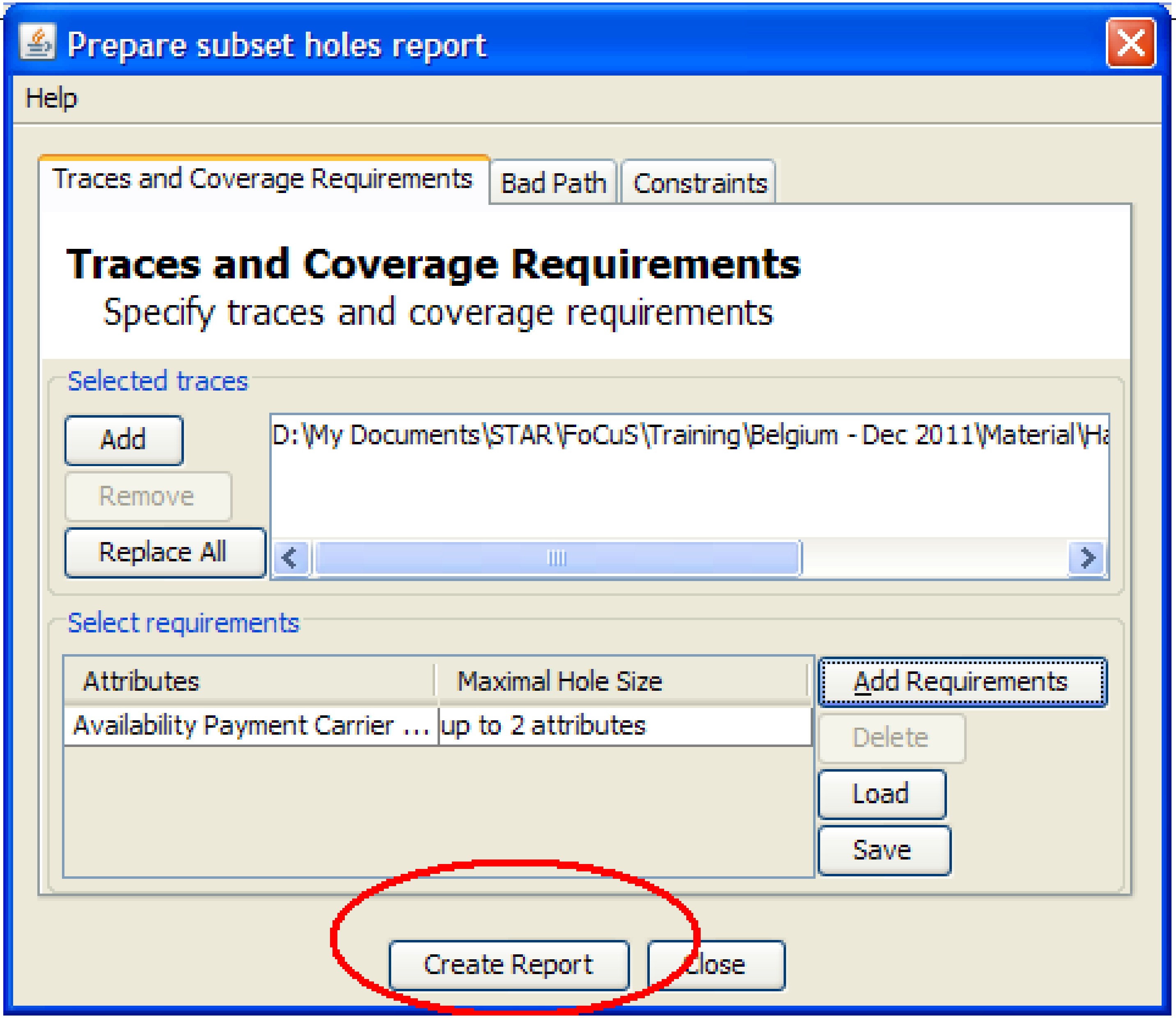
Select coverage require...

Help

Find holes up to size: 2 attributes

Select attributes for requirement

☑ Availability
☑ Payment
☑ Carrier
☑ DeliverySchedule
☑ ExportControl



Traces and Coverage Requirements

Specify traces and coverage requirements

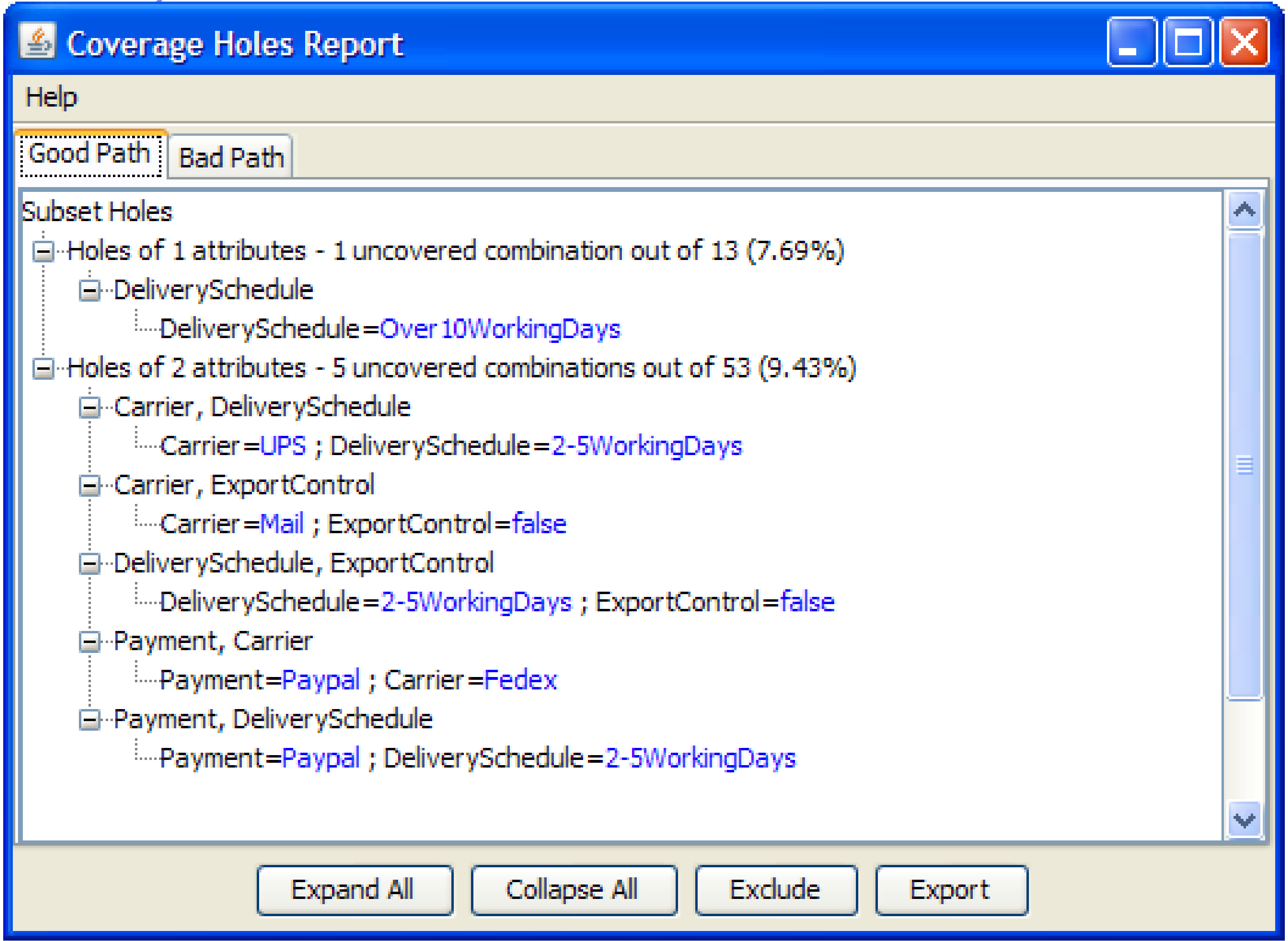
Selected traces

D:\My Documents\STAR\FoCuS\Training\Belgium - Dec 2011\Material\Ha

Select requirements

Attributes	Maximal Hole Size
Availability Payment Carrier ...	up to 2 attributes

Hole Analysis

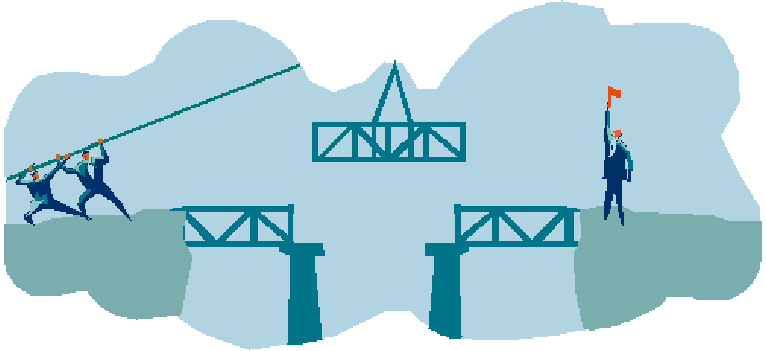


Subset Holes

- Holes of 1 attributes - 1 uncovered combination out of 13 (7.69%)
  - DeliverySchedule
    - DeliverySchedule=Over10WorkingDays
- Holes of 2 attributes - 5 uncovered combinations out of 53 (9.43%)
  - Carrier, DeliverySchedule
    - Carrier=UPS ; DeliverySchedule=2-5WorkingDays
  - Carrier, ExportControl
    - Carrier=Mail ; ExportControl=false
  - DeliverySchedule, ExportControl
    - DeliverySchedule=2-5WorkingDays ; ExportControl=false
  - Payment, Carrier
    - Payment=Paypal ; Carrier=Fedex
  - Payment, DeliverySchedule
    - Payment=Paypal ; DeliverySchedule=2-5WorkingDays

Augmentation

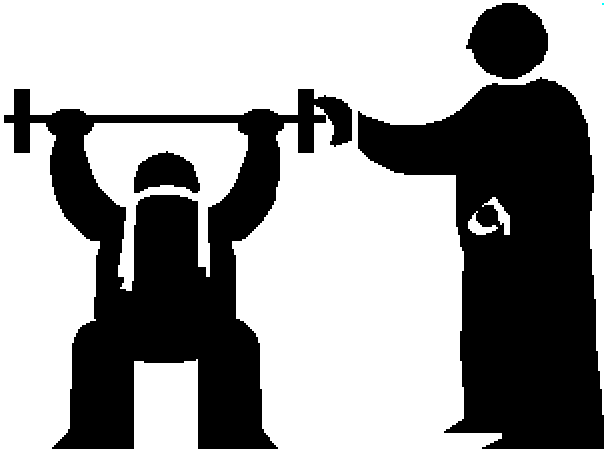
- Input:
  - –Model
  - –Trace
  - –Interaction requirements
- Output:
  - –Further tests to be added, such that:
    - 100% coverage is achieved by the merged test plan (existing + new tests)
    - Smallest possible set

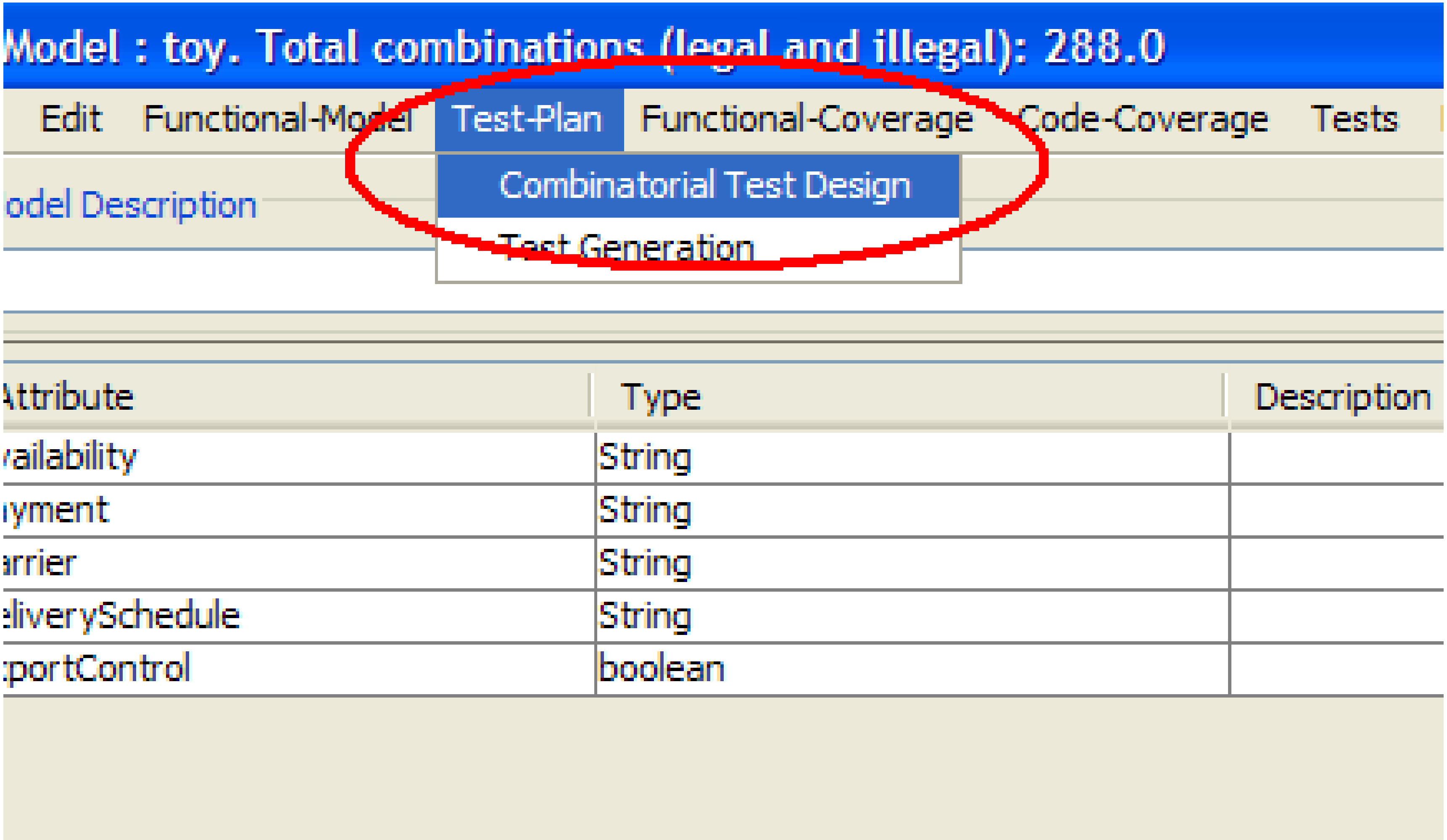


Typical augmentation of a test set using a CTD tool

- We will perform the following:
  - Augment the “customer” test plan to achieve pairwise coverage

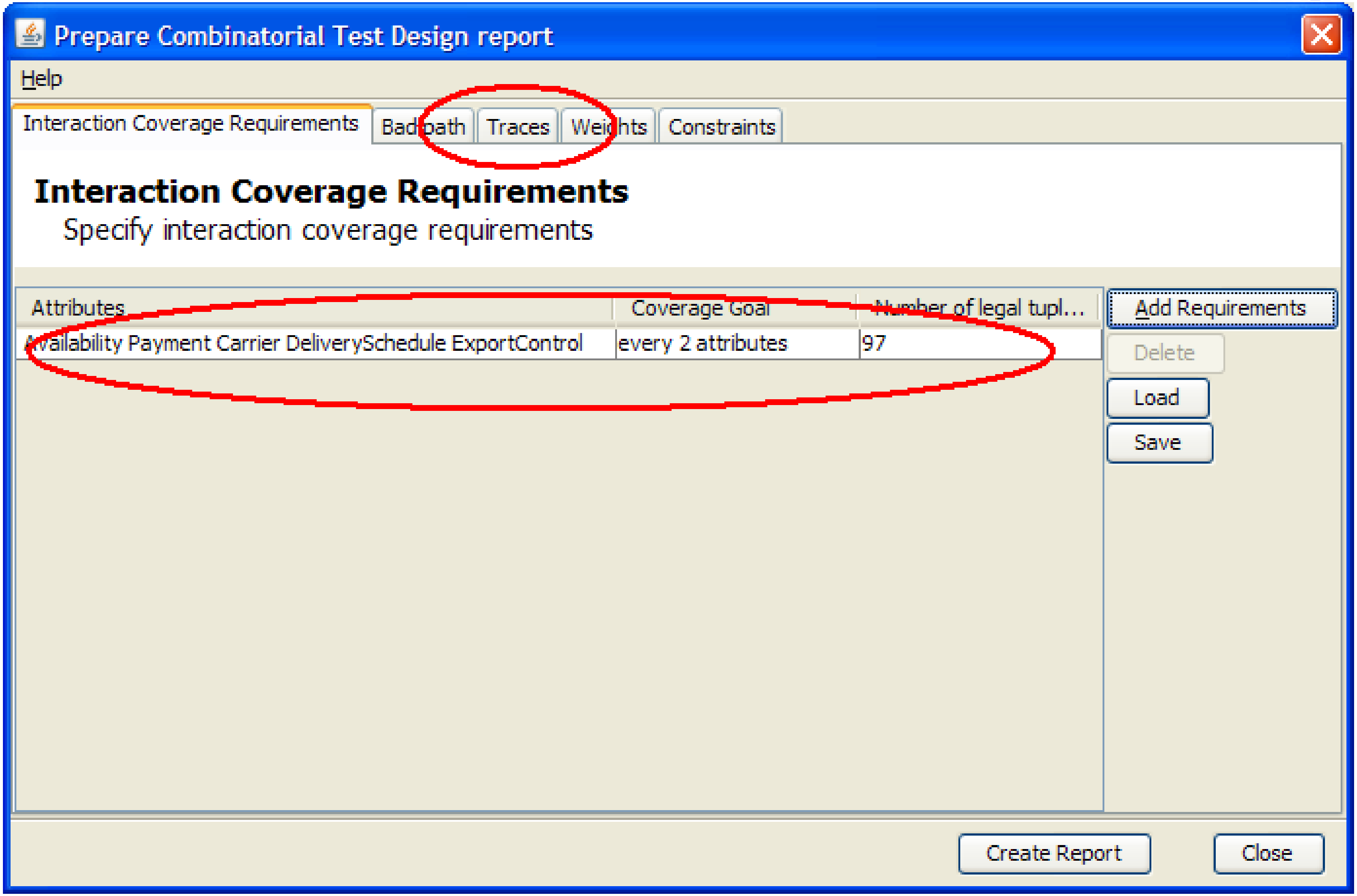
- Note:
  - The size of the augmented result vs. pairwise coverage from scratch
  - The number of new tests to be generated





Model : toy. Total combinations (legal and illegal): 288.0

ttribute	Type	Description
ailability	String	
yment	String	
rrier	String	
liverySchedule	String	
portControl	boolean	

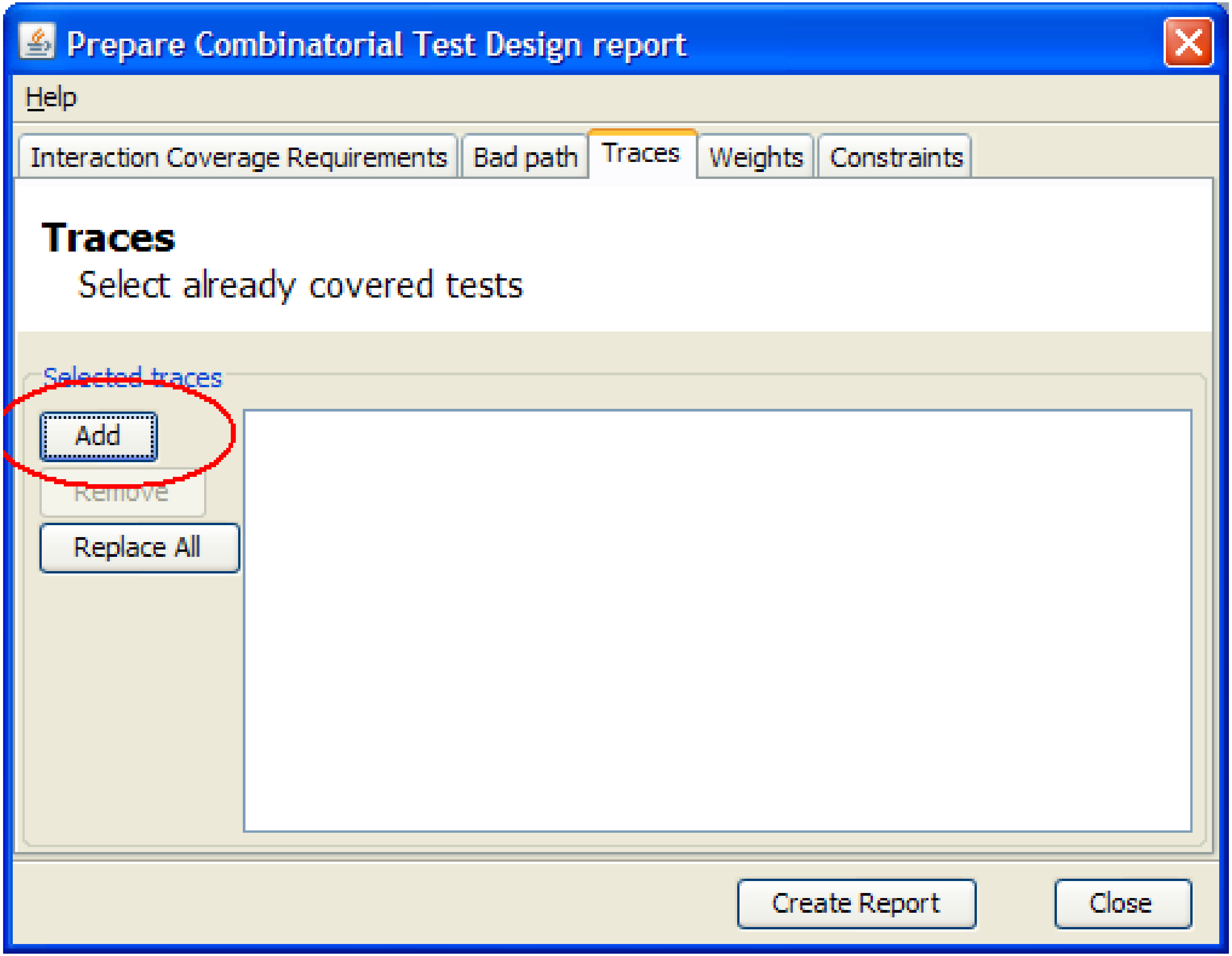


Prepare Combinatorial Test Design report

Interaction Coverage Requirements

Specify interaction coverage requirements

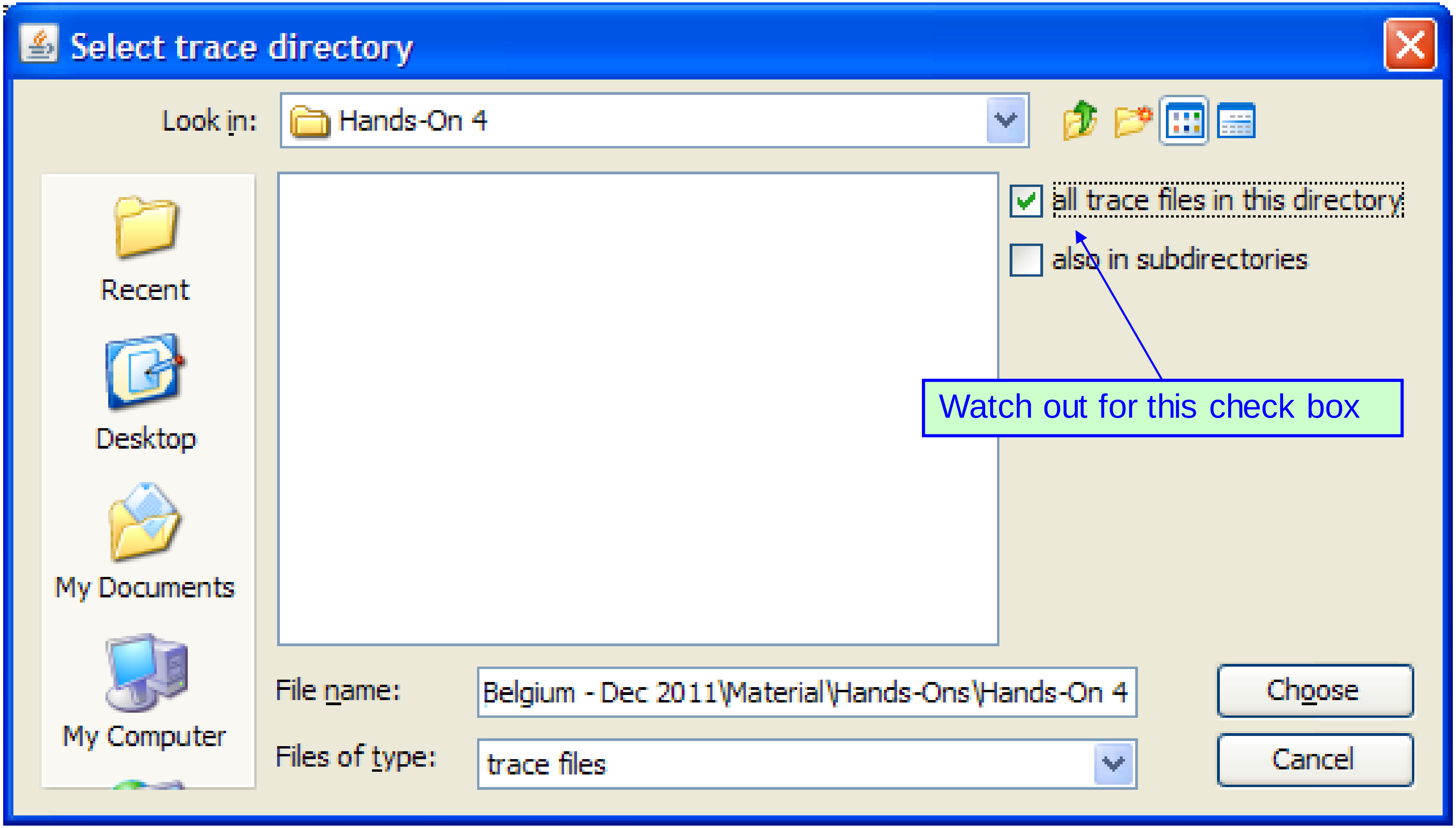
Attributes	Coverage Goal	Number of legal tupl...
Availability Payment Carrier DeliverySchedule ExportControl	every 2 attributes	97



Prepare Combinatorial Test Design report

Traces

Select already covered tests



Select trace directory

Look in: Hands-On 4

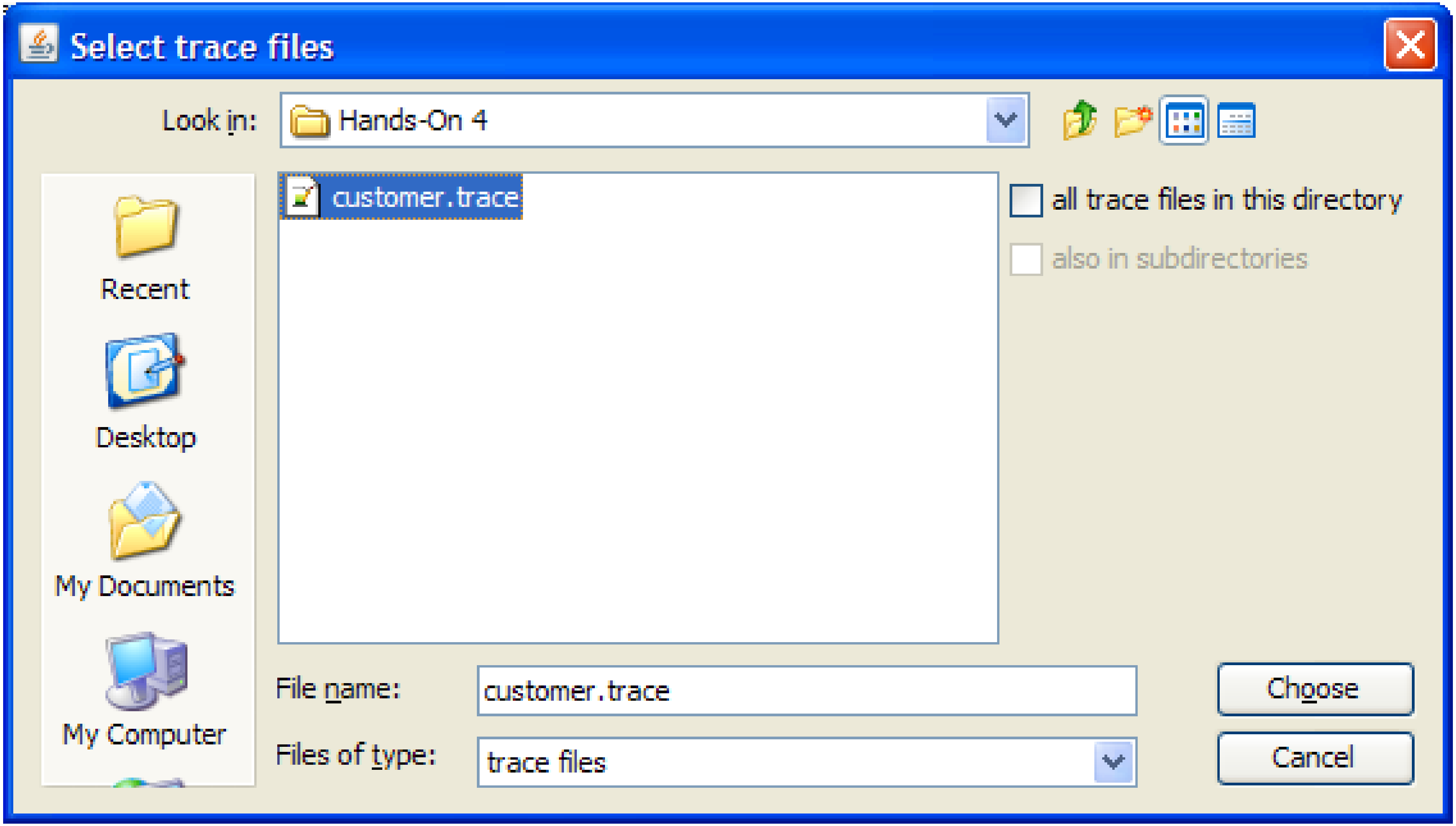
☑ all trace files in this directory

□ also in subdirectories

Watch out for this check box

File name: Belgium - Dec 2011\Material\Hands-Ons\Hands-On 4

Files of type: trace files



Look in: Hands-On 4

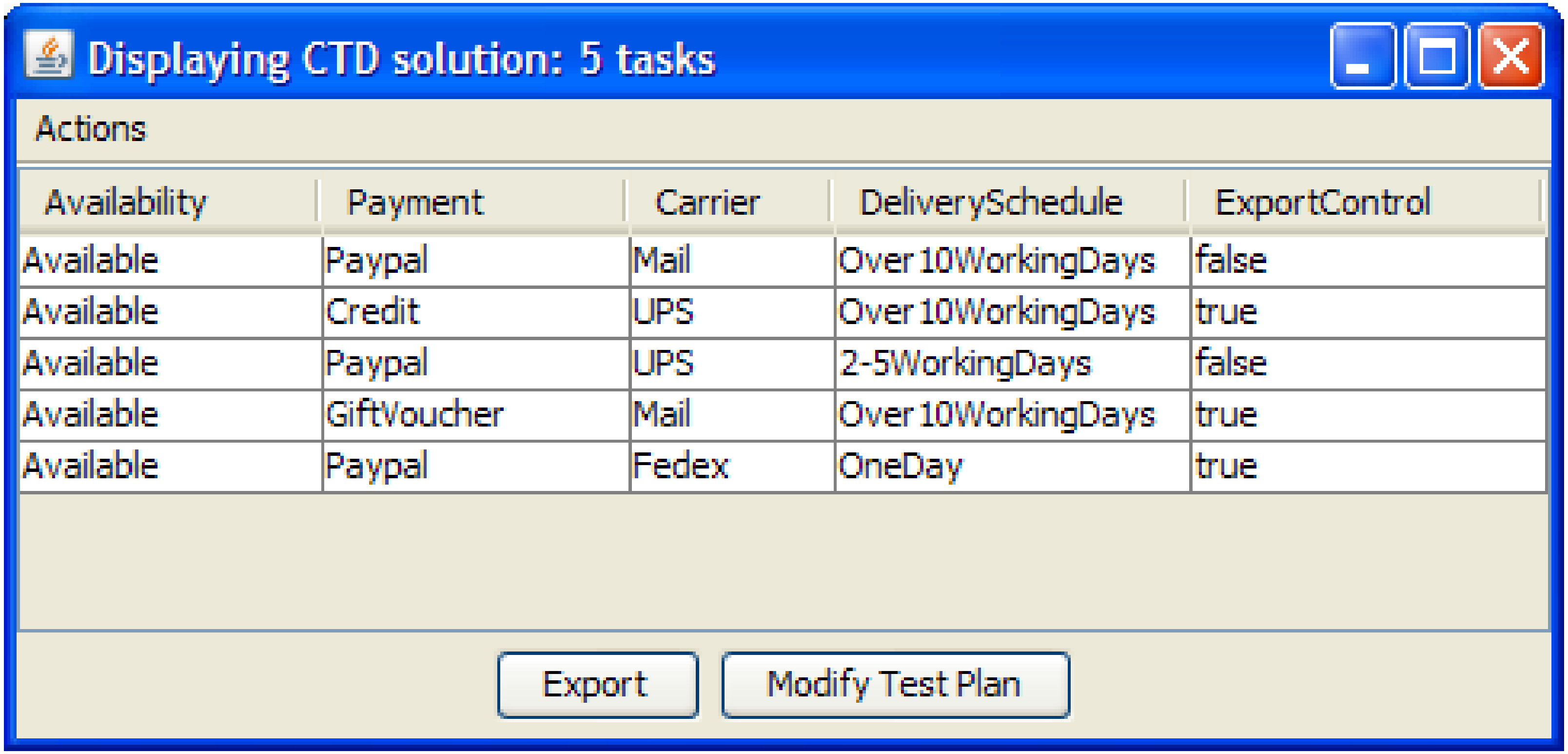
customer.trace

□ all trace files in this directory

□ also in subdirectories

File name: customer.trace

Files of type: trace files



Availability	Payment	Carrier	DeliverySchedule	ExportControl
Available	Paypal	Mail	Over10WorkingDays	false
Available	Credit	UPS	Over10WorkingDays	true
Available	Paypal	UPS	2-5WorkingDays	false
Available	GiftVoucher	Mail	Over10WorkingDays	true
Available	Paypal	Fedex	OneDay	true

Adding these tests to the 30 input tests gives 100% pair coverage

Test Selection

- Input:
  - Model
  - Trace
  - Interaction requirements
- Output:
  - Subset of the trace, that maintains the same interaction coverage
  - Note: a similar technique exists for code coverage, out of scope here

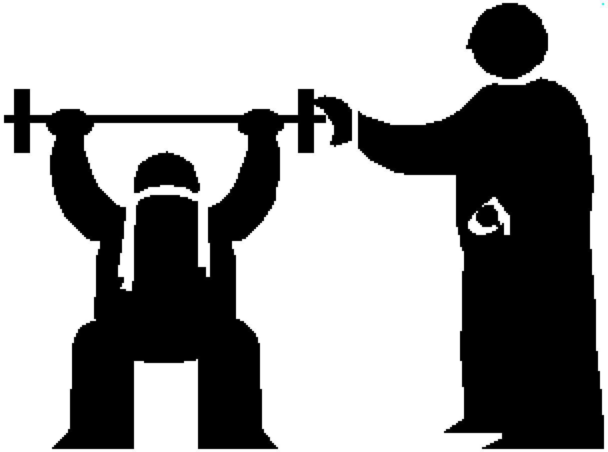
Typical test selection using CTD tools

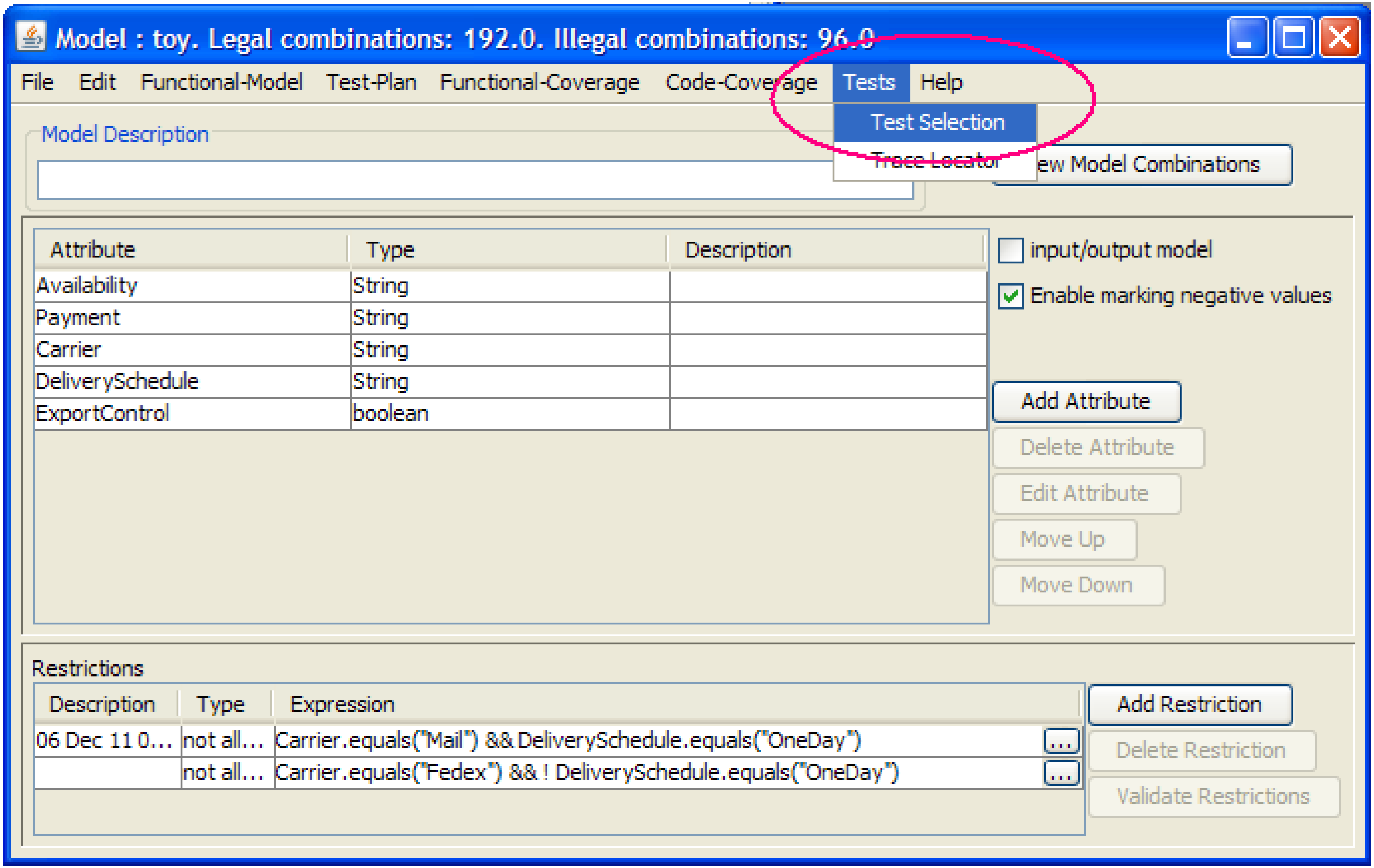
- We will perform the following:
  - Perform test selection on:

    a) the “customer” test plan

    b) the augmented test plan

- Note:
  - The reduction in size
  - The size of the pairwise solution vs. pairwise coverage from scratch
  - The number of reused tests vs. new ones





Model : toy. Legal combinations: 192.0. Illegal combinations: 96.0

Model Description

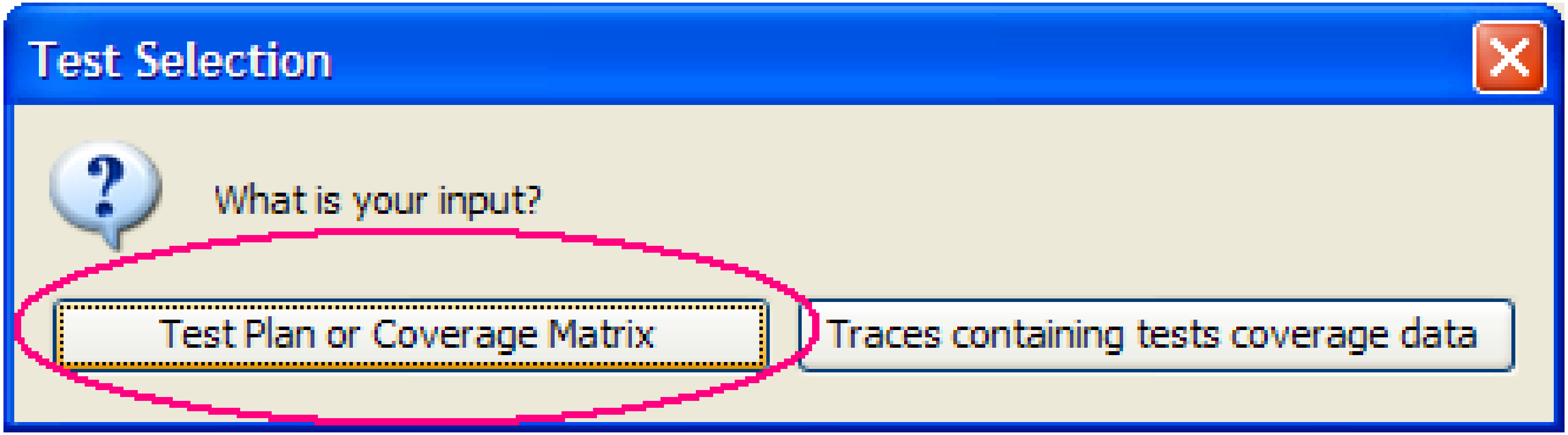
Attribute	Type	Description
Availability	String	
Payment	String	
Carrier	String	
DeliverySchedule	String	
ExportControl	boolean	

□ input/output model

☑ Enable marking negative values

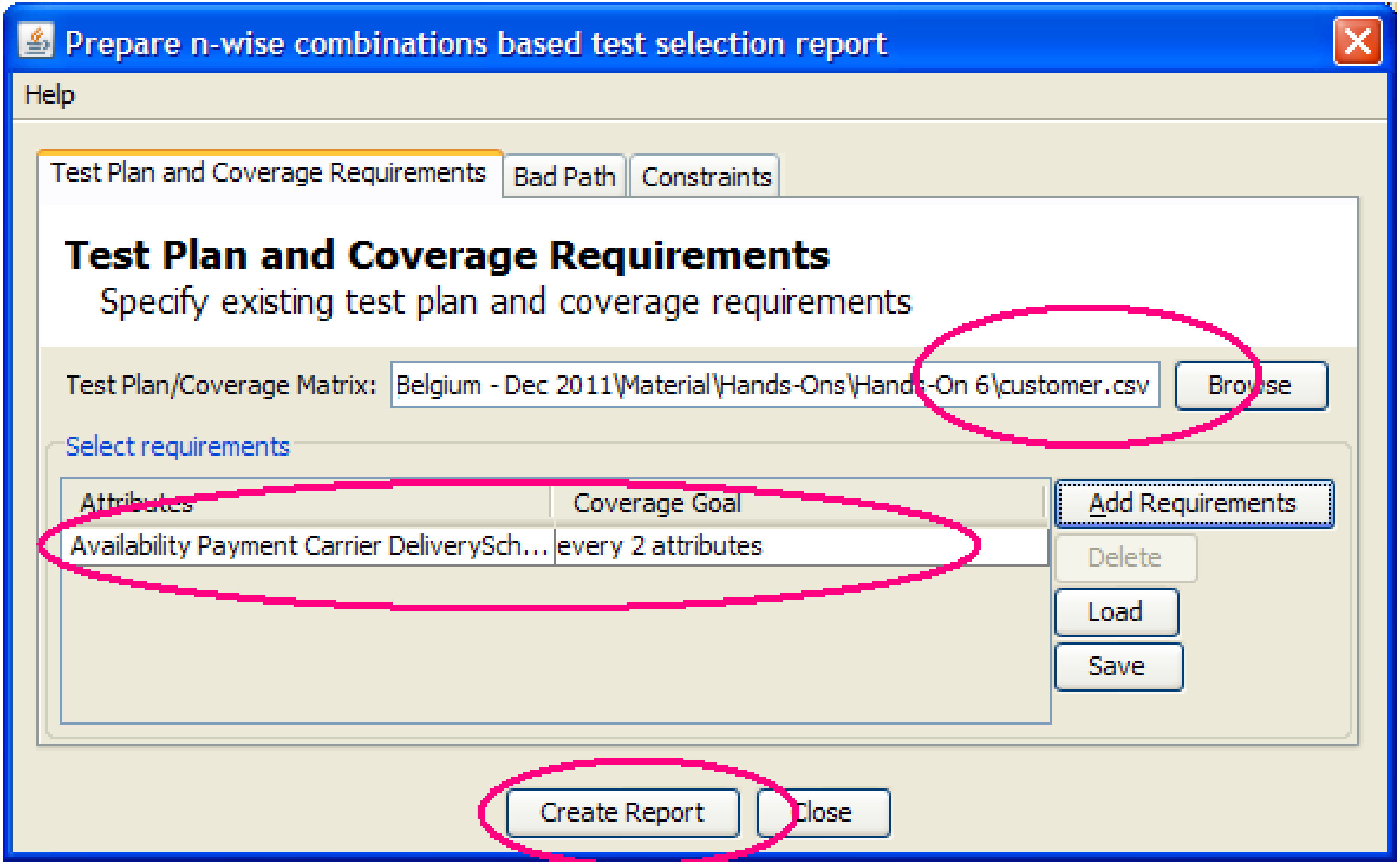
Restrictions

Description	Type	Expression
06 Dec 11 0...	not all...	Carrier.equals("Mail") && DeliverySchedule.equals("OneDay")
	not all...	Carrier.equals("Fedex") && ! DeliverySchedule.equals("OneDay")



What is your input?

Test Plan or Coverage Matrix
Traces containing tests coverage data



Prepare n-wise combinations based test selection report

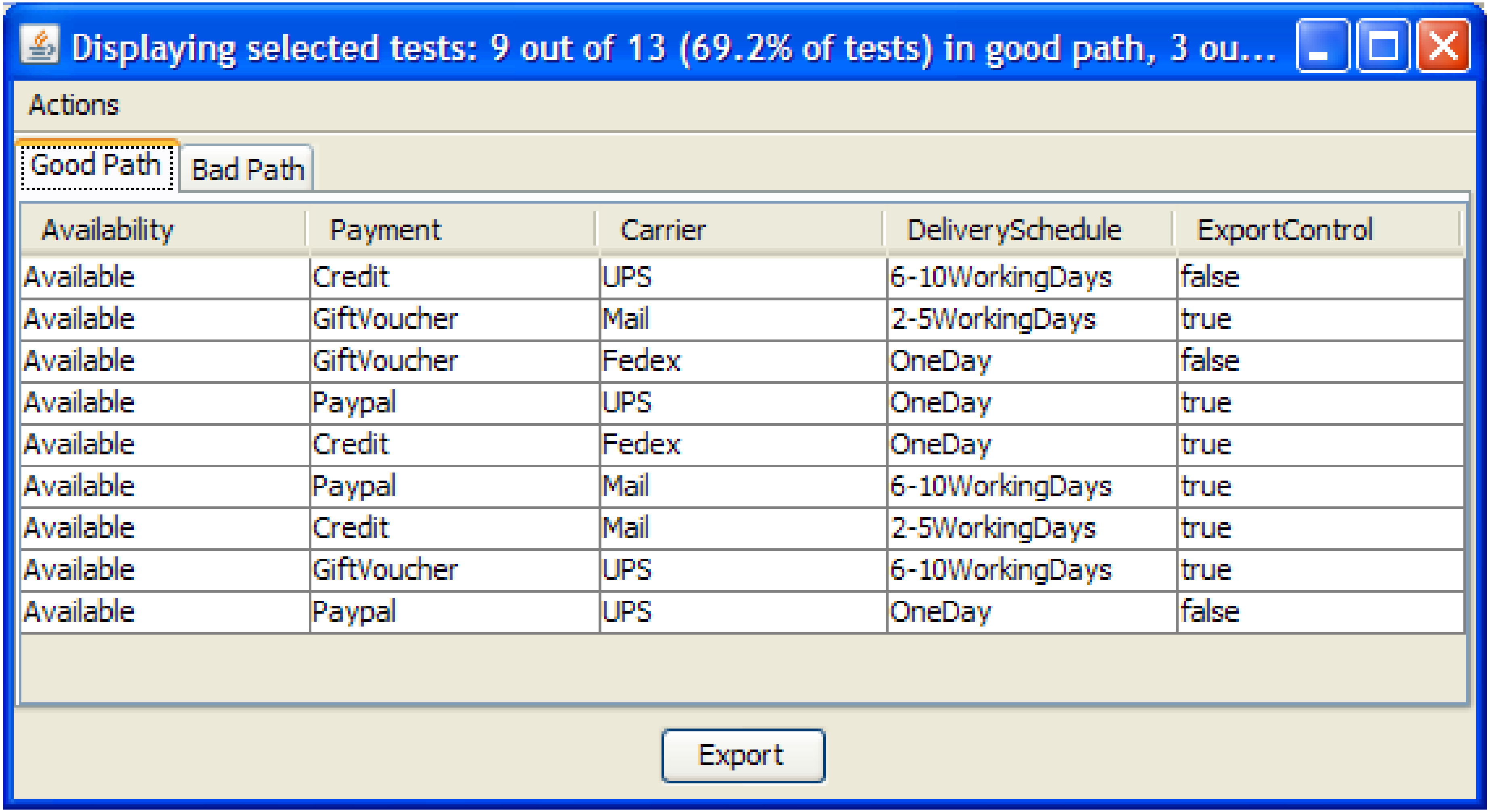
Test Plan and Coverage Requirements

Specify existing test plan and coverage requirements

Test Plan/Coverage Matrix: Belgium - Dec 2011\Material\Hands-Ons\Hands-On 6\customer.csv

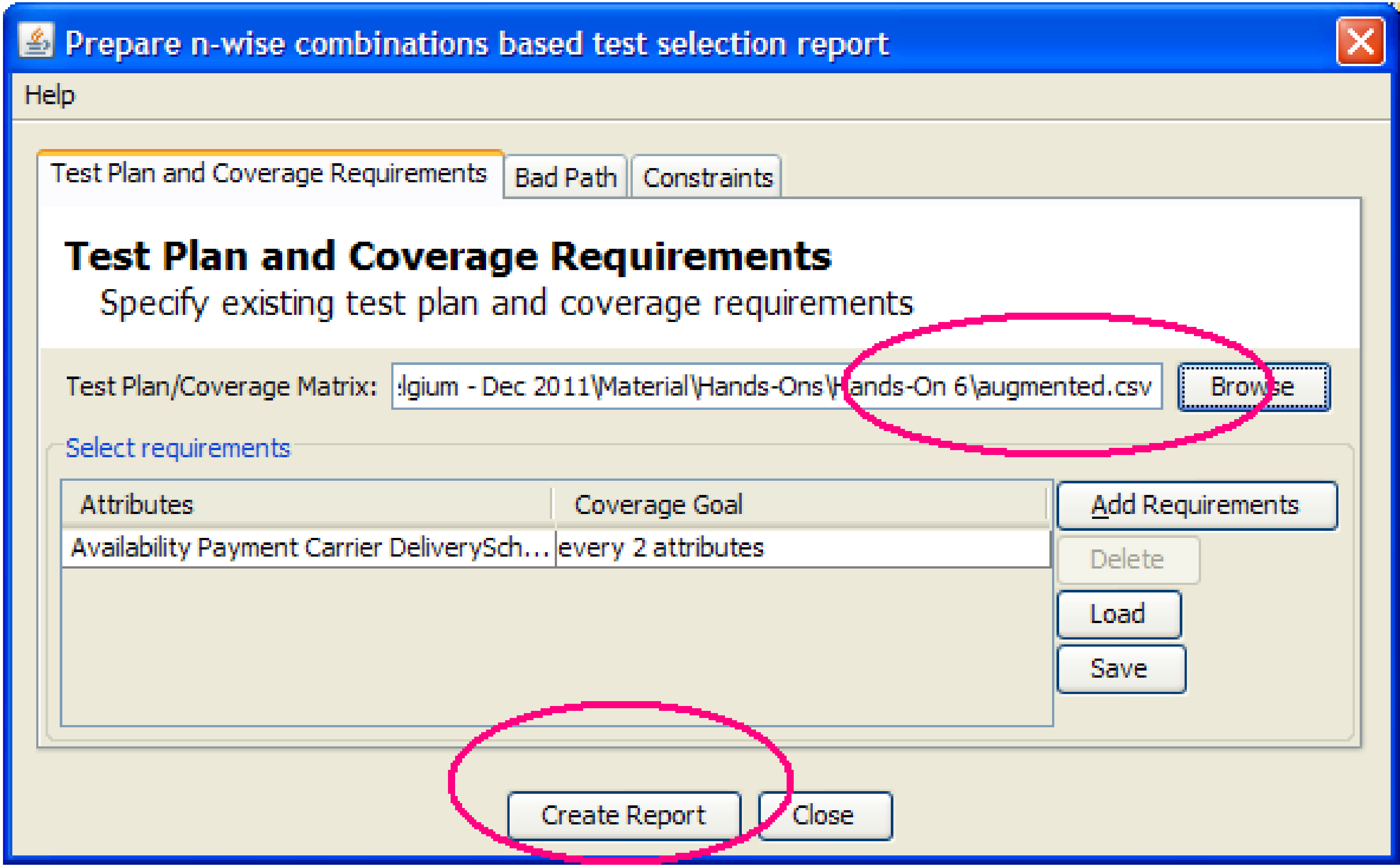
Select requirements

Attributes	Coverage Goal
Availability Payment Carrier DeliverySch…	every 2 attributes



Displaying selected tests: 9 out of 13 (69.2% of tests) in good path, 3 ou...

Availability	Payment	Carrier	DeliverySchedule	ExportControl
Available	Credit	UPS	6-10WorkingDays	false
Available	GiftVoucher	Mail	2-5WorkingDays	true
Available	GiftVoucher	Fedex	OneDay	false
Available	Paypal	UPS	OneDay	true
Available	Credit	Fedex	OneDay	true
Available	Paypal	Mail	6-10WorkingDays	true
Available	Credit	Mail	2-5WorkingDays	true
Available	GiftVoucher	UPS	6-10WorkingDays	true
Available	Paypal	UPS	OneDay	false



Prepare n-wise combinations based test selection report

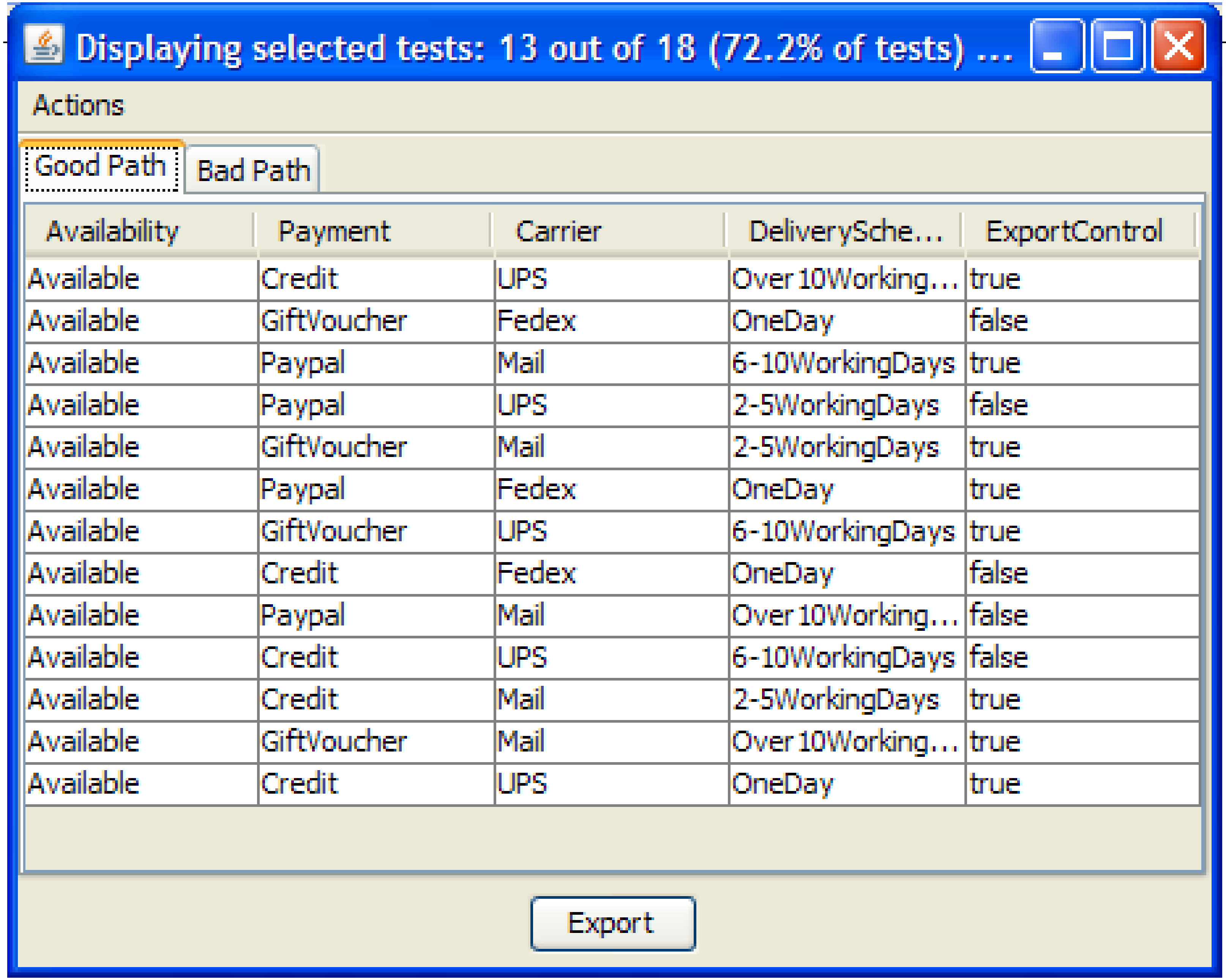
Test Plan and Coverage Requirements

Specify existing test plan and coverage requirements

Test Plan/Coverage Matrix: :lgium - Dec 2011\Material\Hands-Ons\Hands-On 6\augmented.csv

Select requirements

Attributes	Coverage Goal
Availability Payment Carrier DeliverySch…	every 2 attributes



Displaying selected tests: 13 out of 18 (72.2% of tests) ...
Actions
Good Path
Bad Path
Export

Availability	Payment	Carrier	DeliverySche...	ExportControl
Available	Credit	UPS	Over 10Working...	true
Available	GiftVoucher	Fedex	OneDay	false
Available	Paypal	Mail	6-10WorkingDays	true
Available	Paypal	UPS	2-5WorkingDays	false
Available	GiftVoucher	Mail	2-5WorkingDays	true
Available	Paypal	Fedex	OneDay	true
Available	GiftVoucher	UPS	6-10WorkingDays	true
Available	Credit	Fedex	OneDay	false
Available	Paypal	Mail	Over 10Working...	false
Available	Credit	UPS	6-10WorkingDays	false
Available	Credit	Mail	2-5WorkingDays	true
Available	GiftVoucher	Mail	Over 10Working...	true
Available	Credit	UPS	OneDay	true

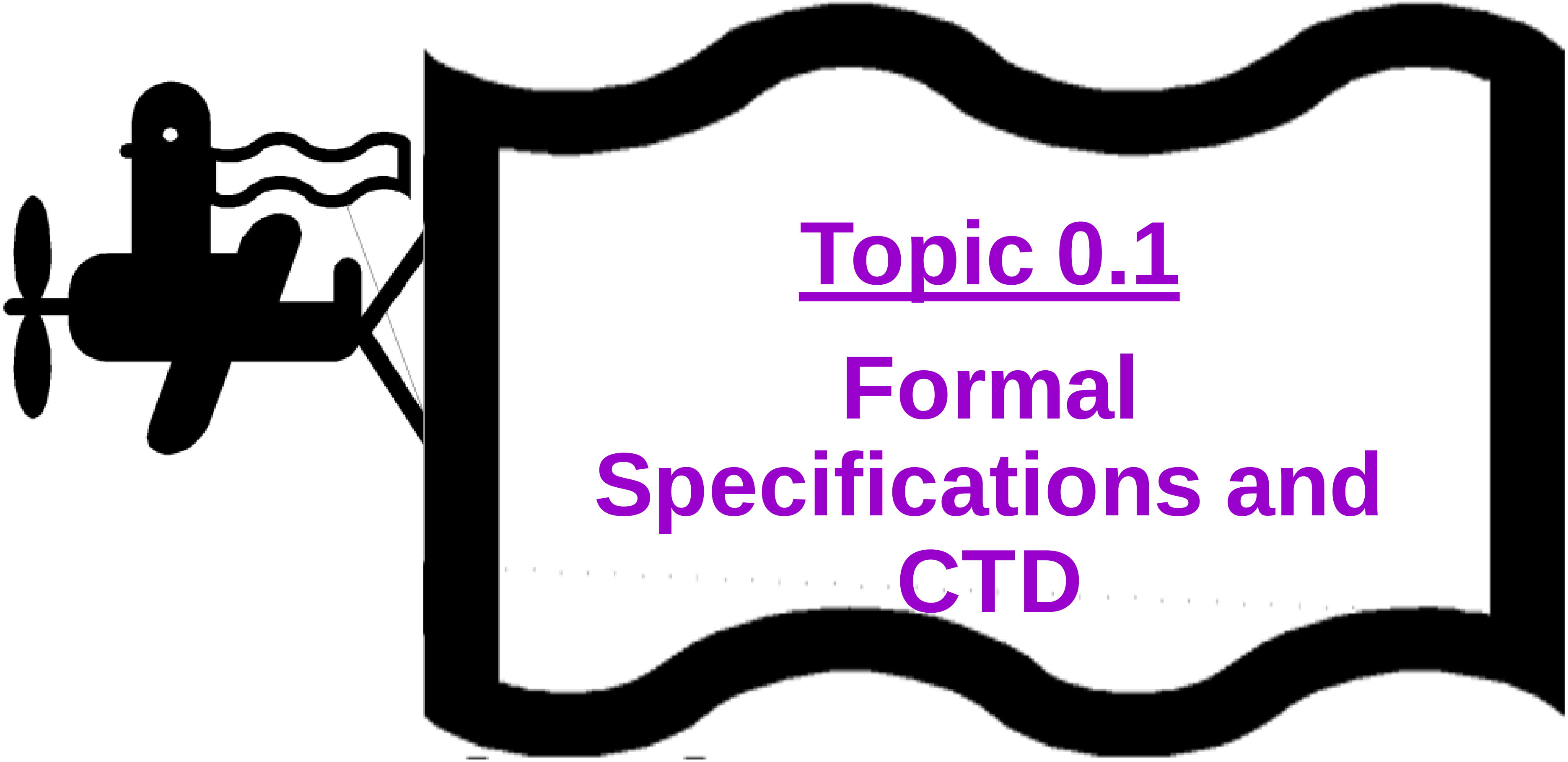
Augmentation and Selection

- Augmentation and selection are independent
- When both are applied – better to augment first, and then select
  - The augmentation may make more existing tests redundant, thus make the selection more effective

CTD from scratch, Augmentation, Selection

	CTD From Scratch	Augmentation	Selection
Existing tests (trace)	Not required	Required (trace file)	Required (csv file)
Number of new tests	High	Medium	None
Total number of tests	Low	Higher	Low
Interaction coverage	Full	Full	As existing tests (typically low)

Appendix



Topic 0.1

Formal Specifications and CTD

Software formal specification

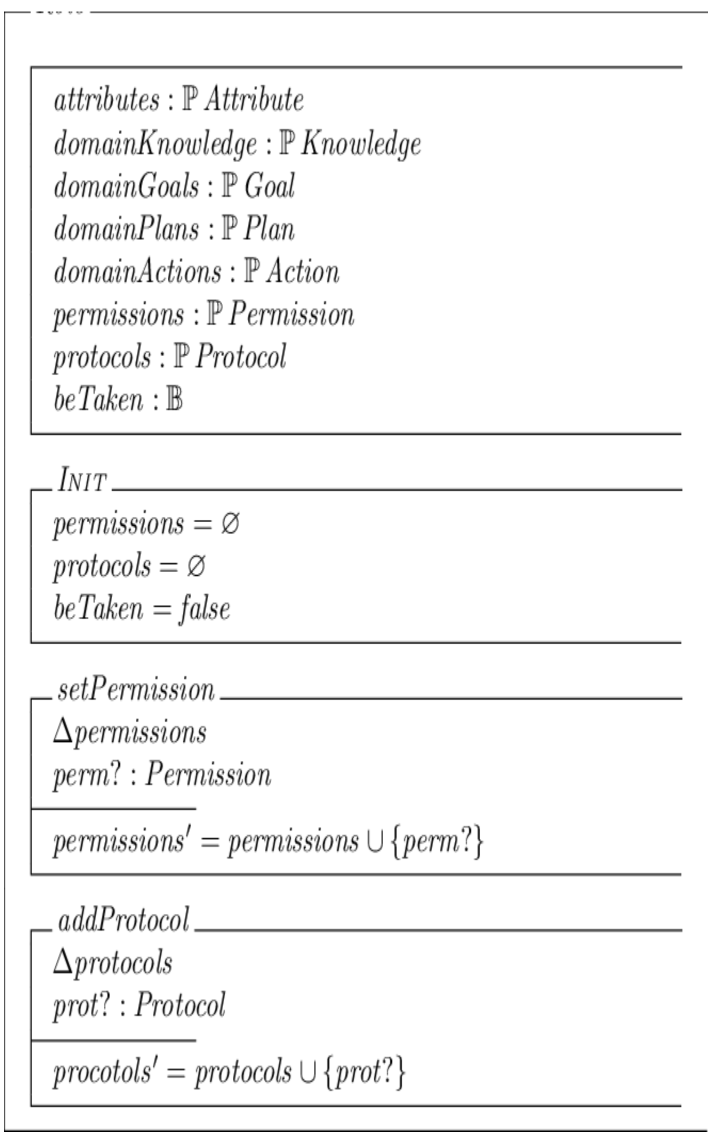
- Specification of aspects of the software or parts of its functionality using a mathematical model that specifies what the system is required to do

- Pros –
  - Removes ambiguity of software specification that is an important factor in specification and design issues
  - Enables formal reasoning making claims and proving that some aspects of the software are correct possible

- Cons –
  - A lot of work
  - Requires mathematical training to apply

A Z specification example

- The requirement in natural language
  - In a group communication setting there are n nodes that participates in the communication. A central leader node holds a communication group, GROUP(central) which is a subset of the set of nodes. In addition, each of the n nodes i in {1,…,n} holds an individual group communication set GROUP(i). As the system evolves in any state S(t) of the system, t=1, 2, 3,… we have that GROUP(i) is included in GROUP(central)
- A Z like specification will state this more formally
  - For all state : N, i in NODES, there exists central in NODES, such that
    - GROUP(i, state) is included in GROUP(central, state)
- More details on the Z formal specification language can be found here
  - [11Z.pdf (kit.edu)](11Z.pdf (kit.edu))



$attributes : \mathbb{P}\ Attribute$
$domainKnowledge : \mathbb{P}\ Knowledge$
$domainGoals : \mathbb{P}\ Goal$
$domainPlans : \mathbb{P}\ Plan$
$domainActions : \mathbb{P}\ Action$
$permissions : \mathbb{P}\ Permission$
$protocols : \mathbb{P}\ Protocol$
$beTaken : \mathbb{B}$

Init

$permissions = \varnothing$
$protocols = \varnothing$
$beTaken = false$

setPermission

$\Delta permissions$
$perm? : Permission$

$permissions' = permissions \cup \{perm?\}$

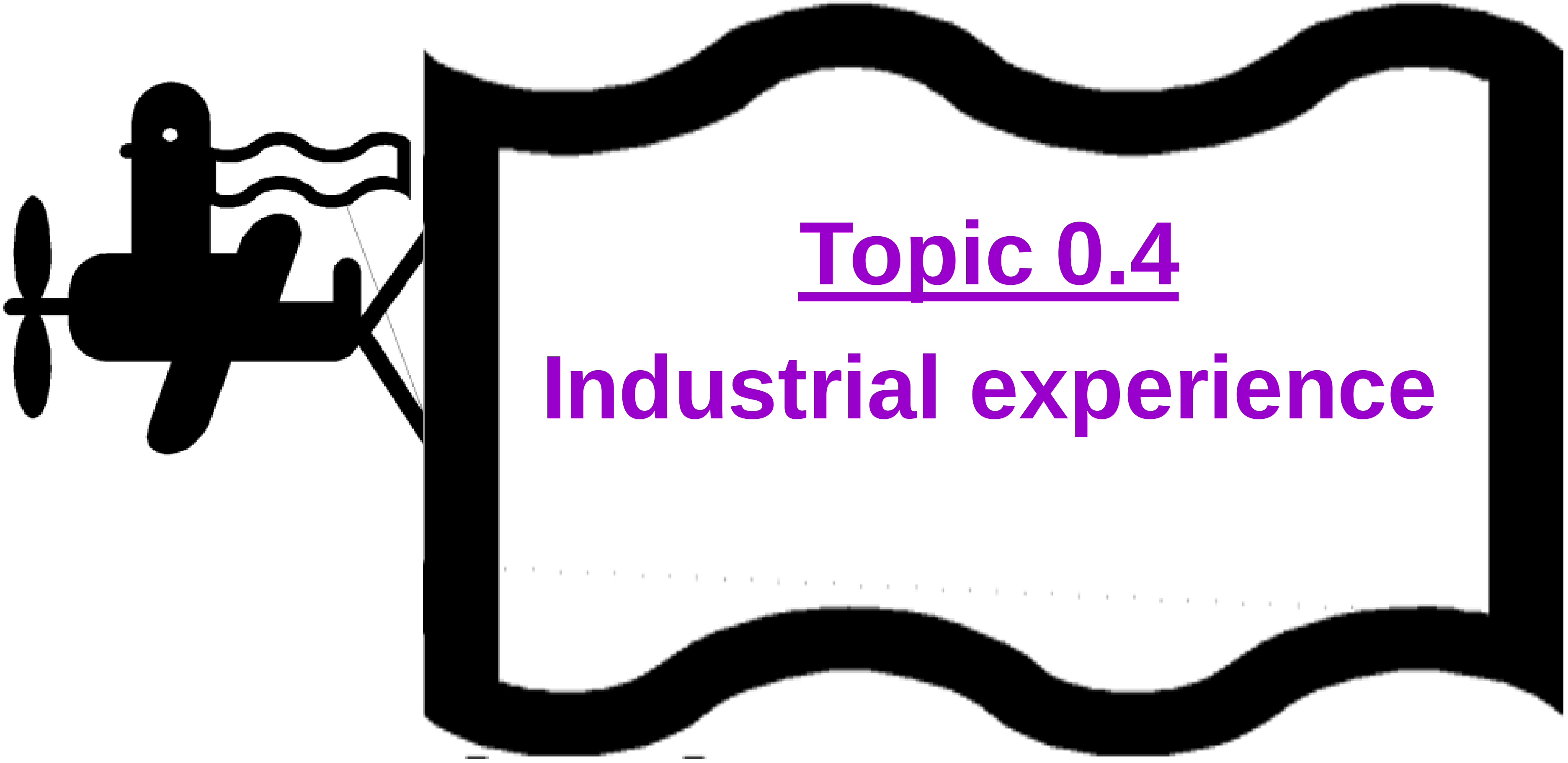
addProtocol

$\Delta protocols$
$prot? : Protocol$

$procotols' = protocols \cup \{prot?\}$

CTD and formal software specification

- Essentially the language of CTD is the language of a relation, i.e., a subset of a Cartesian product
- It is as if we stripped a first order language (more or less Z) from everything but the concept of a relation
- We are thus striking a tradeoff in which
  - The process of modeling using CTD removes ambiguity and finds problems, thus we are still getting that benefit of formal specification
  - The redundant language compared to Z is much less powerful but is more accessible and intuitive to testers



Topic 0.4

Industrial experience

Example 2: Medical claims adjudication

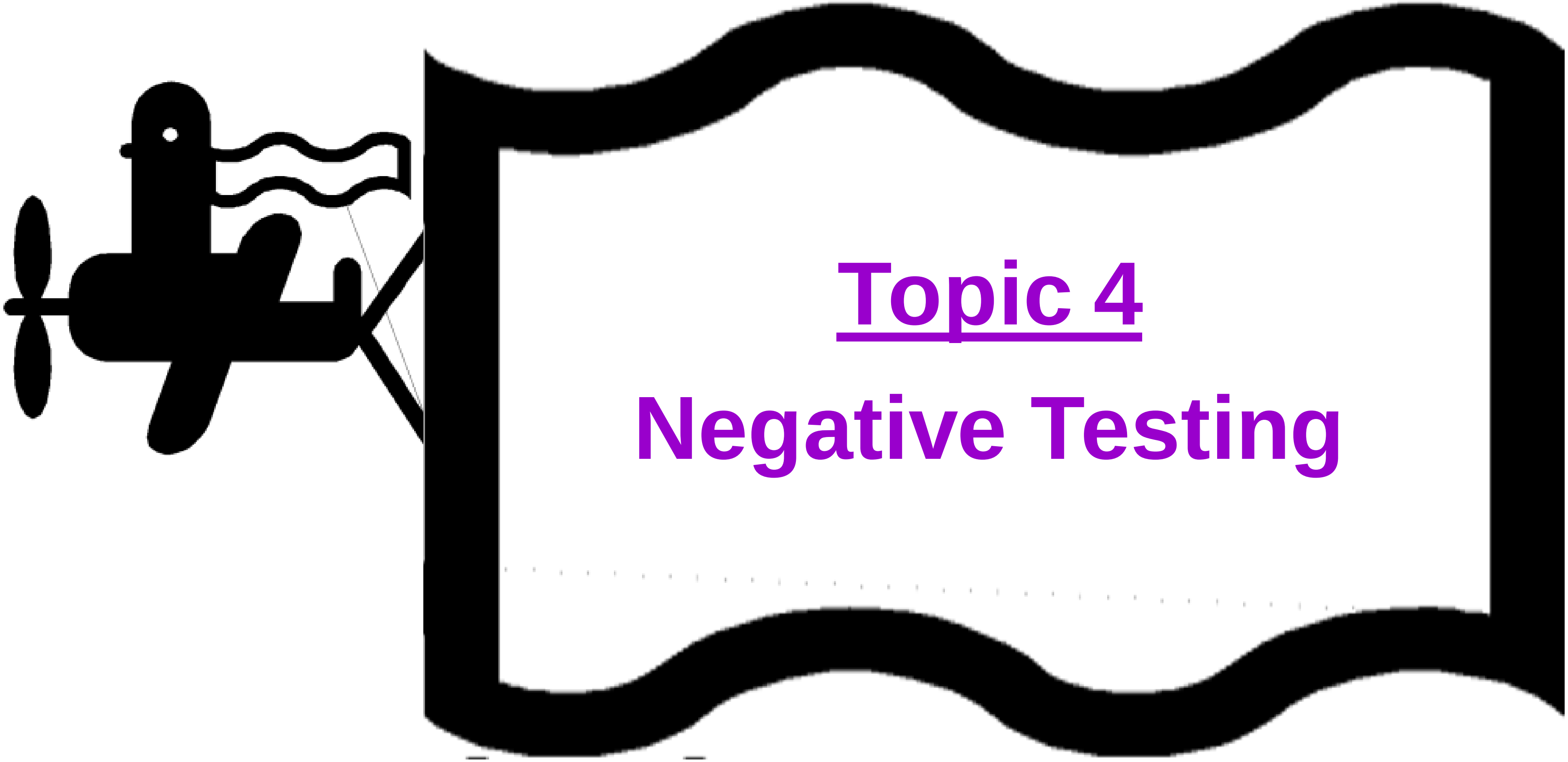
- The testing project was late, with too many tests in plan
- We were given examples of test plans – spreadsheets, but not structured
- We created an initial model that reflected our (incorrect) understanding
- We spent spent a week at the customer site, learning the system and reviewing models
- Several more weeks of iterating on the model were done remotely
- The resulting CTD proposal had about 50 tests with more coverage than the original 6000

Example 3: Financial institute

- Customer has too many tests, of insufficient quality
- We were given a set of Business Requirement Documents, and high level test plans
- We created a preliminary model for demonstrating CTD
  - Later the customer explained that the focus of the model is not of interest, but it was still useful for an introduction
- We spent a week at the customer site, learning the system, reviewing models and learning about existing test cases and the format of the test data
- A client SME dedicated the entire week for this work
  - Most data is SME domain knowledge that doesn't exist in any documentation
- At the end of week 1, the model was almost final, including restrictions and coverage requirements
  - Throughout the following three weeks some minor modifications were performed
- Some (relatively simple) scripts had to be written to extract traces from existing test data
  - Apply significant domain knowledge supplied by customer SME
- The following steps were performed using the model and traces:
  - Hole analysis on existing test plans (256 tests, achieving 45% coverage)
  - Enhancing existing test plans
  - Test selection (suggested 27 tests – 18 reused and 9 new ones)
  - CTD from scratch (suggested 16 tests for good path, 20 for bad path)
-

Example 4: a middleware test project

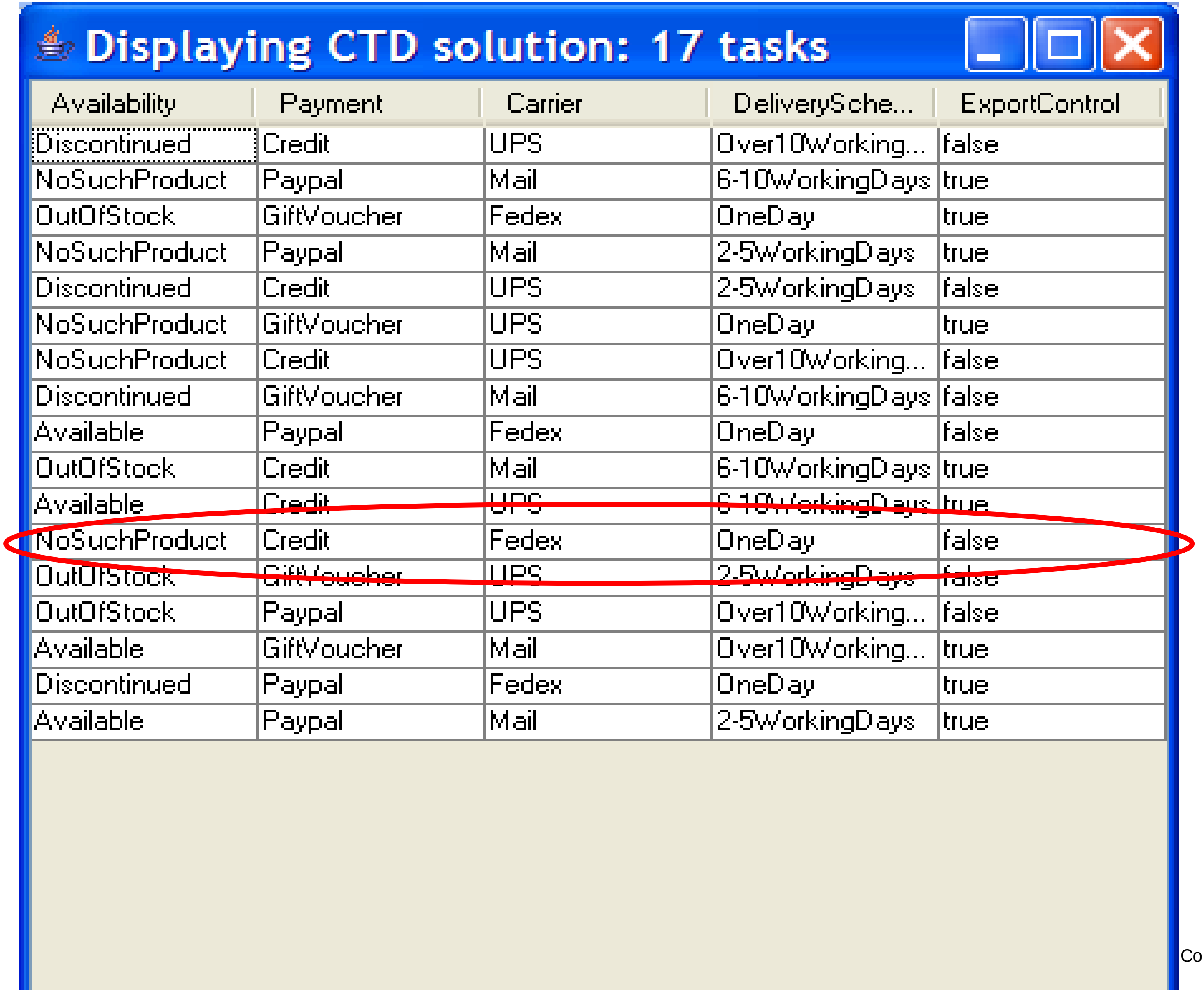
- The testing project has too many tests in plan
- Integrates tests from various sources, duplicates are possible
- Writing new tests is extremely costly, which means that only test selection is feasible
- Domains are huge, the challenge is to cover every value
- We were given examples of very structured test plans in the form of Test Coverage Matrices (TCMs)
- Imported one such TCM into the CTD tool
- Domain 1:
  - Coverage analysis revealed that 20% of the values are uncovered
  - Test selection showed that only 48% of the tests are needed for covering the same values, and only 88% are needed for covering the same pairs
- Domain 2:
  - 71% coverage achieved by existing tests
  - Same coverage can be achieved by selecting 58% of the tests



Topic 4

Negative Testing

A closer look at the resulting test plan



Displaying CTD solution: 17 tasks

Availability	Payment	Carrier	DeliverySche...	ExportControl
Discontinued	Credit	UPS	Over10Working...	false
NoSuchProduct	Paypal	Mail	6-10WorkingDays	true
OutOfStock	GiftVoucher	Fedex	OneDay	true
NoSuchProduct	Paypal	Mail	2-5WorkingDays	true
Discontinued	Credit	UPS	2-5WorkingDays	false
NoSuchProduct	GiftVoucher	UPS	OneDay	true
NoSuchProduct	Credit	UPS	Over10Working...	false
Discontinued	GiftVoucher	Mail	6-10WorkingDays	false
Available	Paypal	Fedex	OneDay	false
OutOfStock	Credit	Mail	6-10WorkingDays	true
Available	Credit	UPS	6-10WorkingDays	true
NoSuchProduct	Credit	Fedex	OneDay	false
OutOfStock	GiftVoucher	UPS	2-5WorkingDays	false
OutOfStock	Paypal	UPS	Over10Working...	false
Available	GiftVoucher	Mail	Over10Working...	true
Discontinued	Paypal	Fedex	OneDay	true
Available	Paypal	Mail	2-5WorkingDays	true

Negative Values

- When no such product exists, the test will terminate prematurely
  - The interactions between the other attributes will not be actually tested by this test
- The combination Payment=Credit, Carrier=Fedex appears only in one test, that is failing
  - It is covered by the test plan, but will not be reached by the executed code
- To really achieve 100% interaction coverage, negative values must be identified and considered
- FoCuS supports testing of bad paths
  - Indication of the failure values in the model
  - Creation of separate test plans for good paths and bad paths

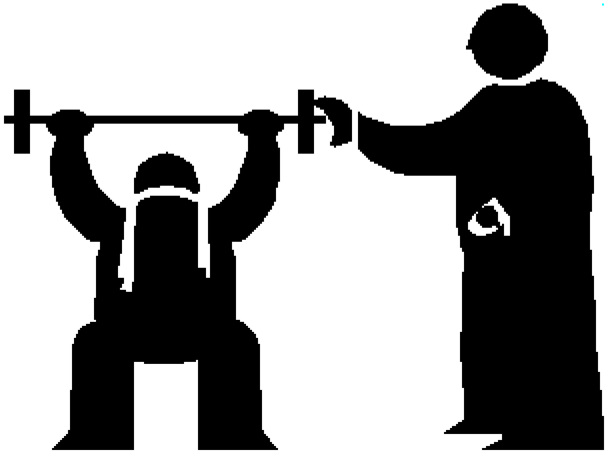
Negative Testing

- Testing of what happens when things go wrong
- There can be many ways to fail
  - Wrong inputs, unexpected conditions, unavailable resources..
- Testers tend to concentrate on the good path, and neglect the bad paths
  - Failure scenarios are less intuitive to consider
  - Bad path tests can be more difficult to implement
  - Results in incomprehensible error messages, unnecessary crashes, and chain reactions of failures..
- Especially important to consider when using CTD, as otherwise might result in false coverage of interactions

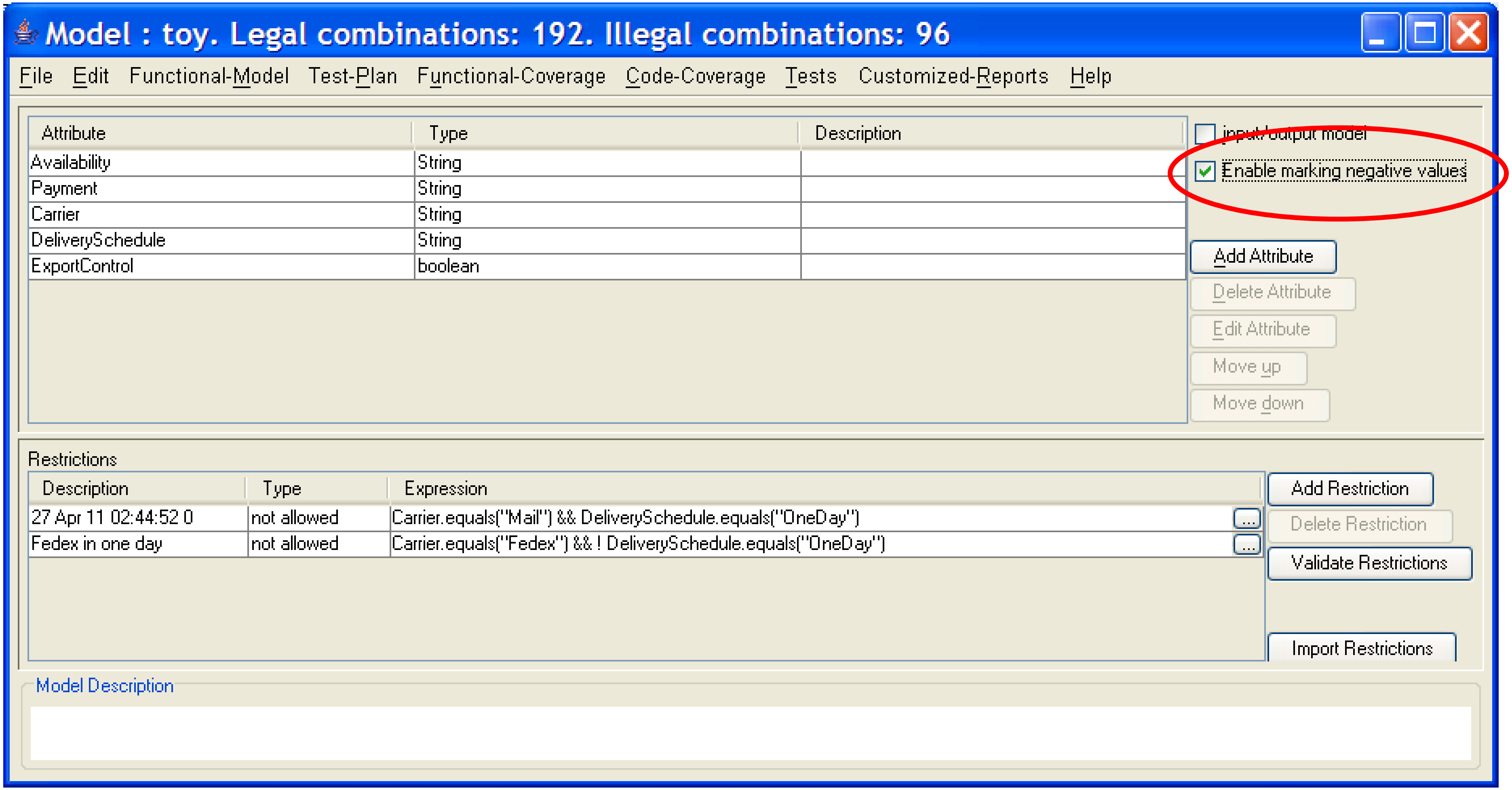
Typical negative testing

- We will perform the following:
  - Mark values as negative
  - Generate a new CTD test plan

- Note the coverage requirements for the bad path test plan
- Observe the two test plans generated



Marking negative values



Model : toy. Legal combinations: 192. Illegal combinations: 96

Attribute	Type	Description
Availability	String	
Payment	String	
Carrier	String	
DeliverySchedule	String	
ExportControl	boolean	

□ input/output model

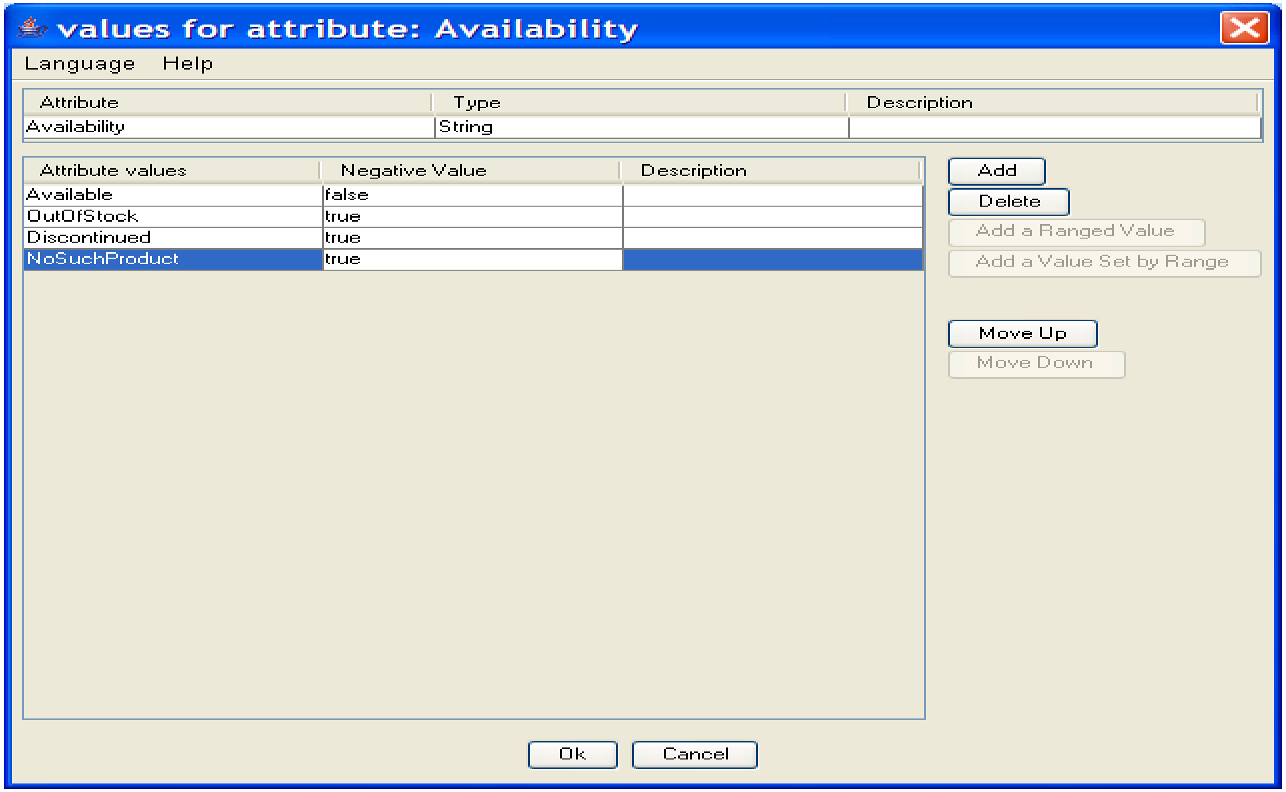
☑ Enable marking negative values

Restrictions

Description	Type	Expression
27 Apr 11 02:44:52 0	not allowed	Carrier.equals("Mail") && DeliverySchedule.equals("OneDay")
Fedex in one day	not allowed	Carrier.equals("Fedex") && ! DeliverySchedule.equals("OneDay")

Model Description

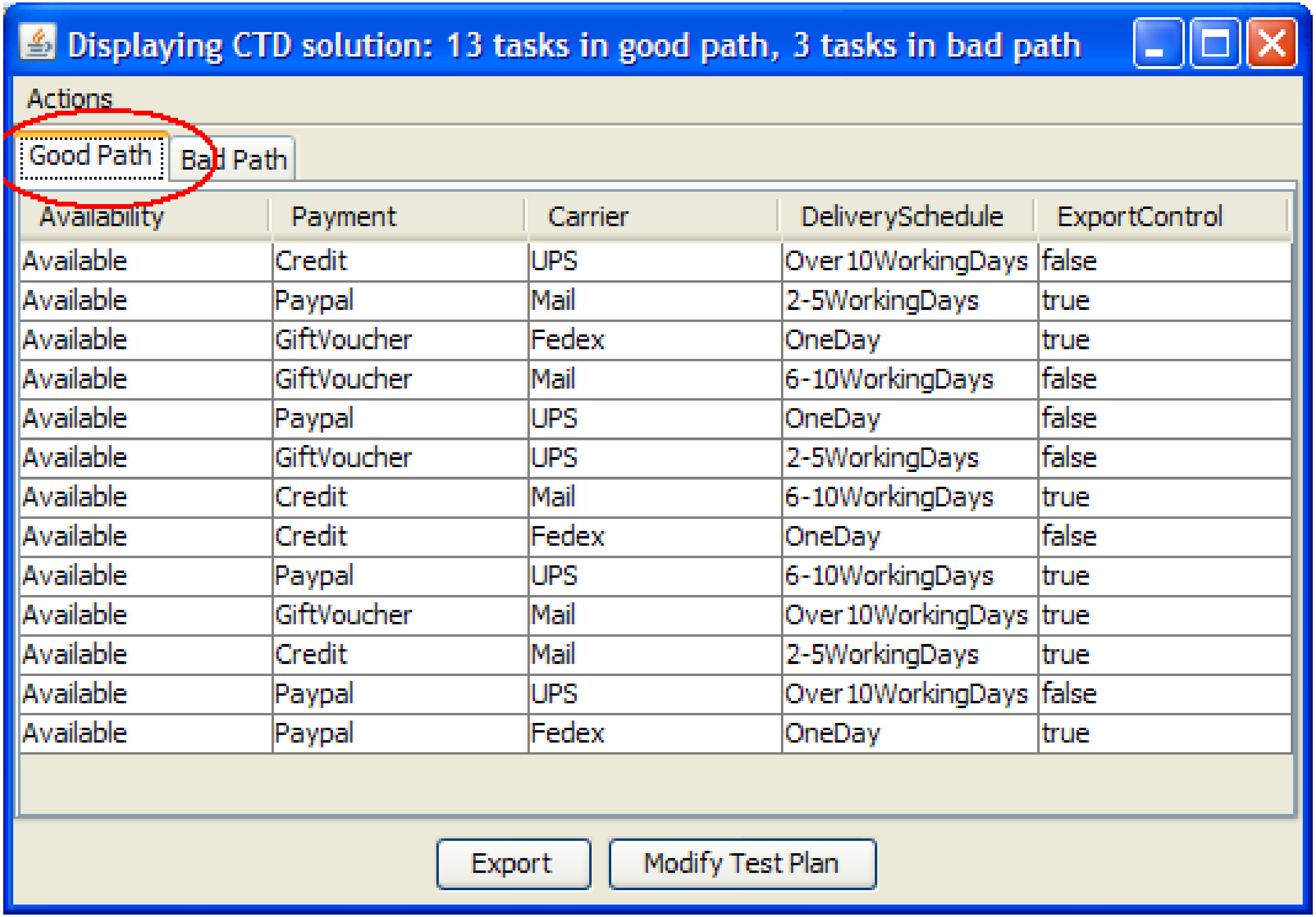
Marking negative values



Attribute	Type	Description
Availability	String	

Attribute values	Negative Value	Description
Available	false	
OutOfStock	true	
Discontinued	true	
NoSuchProduct	true	

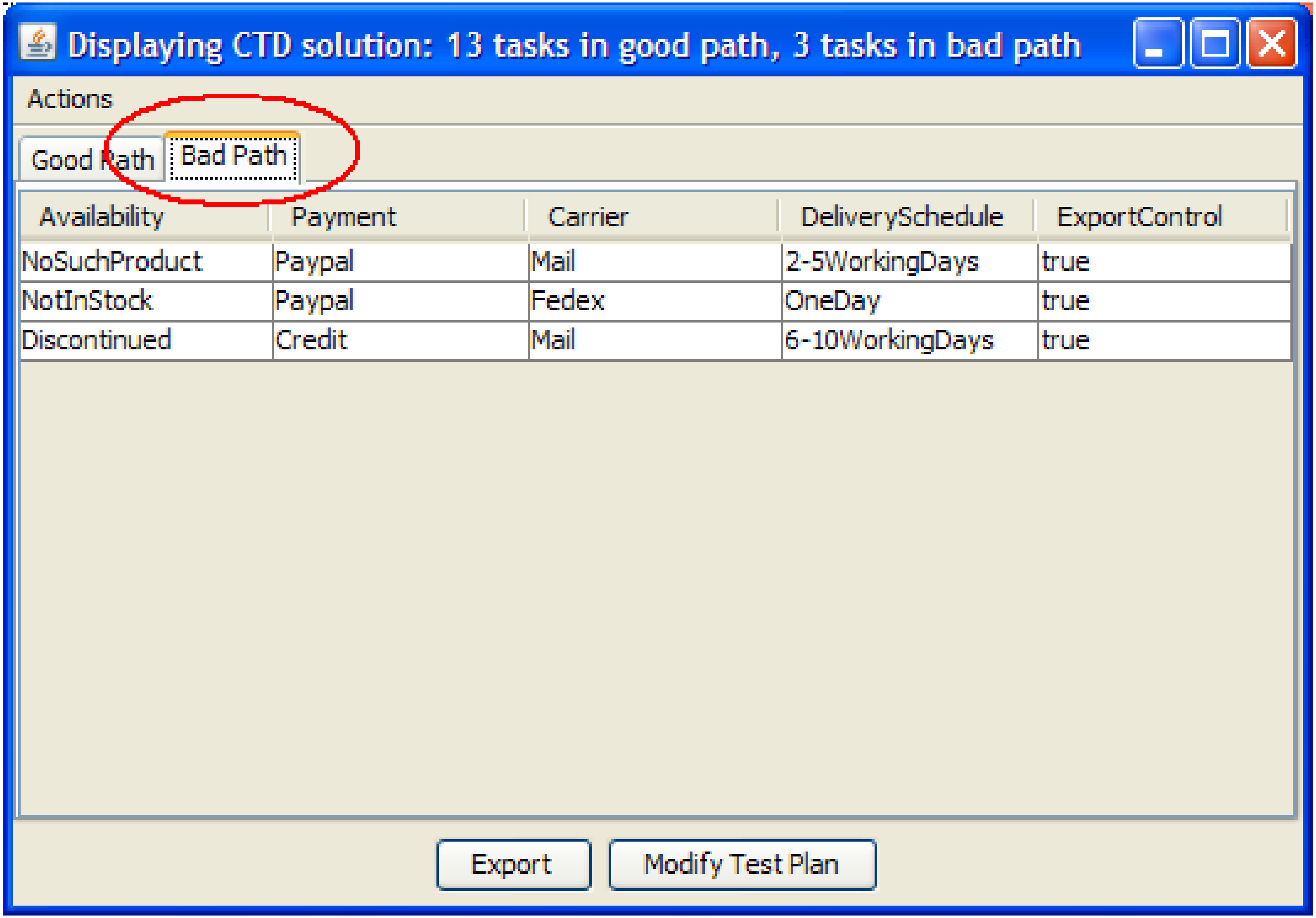
Good path test plan does not contain negative values



Displaying CTD solution: 13 tasks in good path, 3 tasks in bad path

Availability	Payment	Carrier	DeliverySchedule	ExportControl
Available	Credit	UPS	Over10WorkingDays	false
Available	Paypal	Mail	2-5WorkingDays	true
Available	GiftVoucher	Fedex	OneDay	true
Available	GiftVoucher	Mail	6-10WorkingDays	false
Available	Paypal	UPS	OneDay	false
Available	GiftVoucher	UPS	2-5WorkingDays	false
Available	Credit	Mail	6-10WorkingDays	true
Available	Credit	Fedex	OneDay	false
Available	Paypal	UPS	6-10WorkingDays	true
Available	GiftVoucher	Mail	Over10WorkingDays	true
Available	Credit	Mail	2-5WorkingDays	true
Available	Paypal	UPS	Over10WorkingDays	false
Available	Paypal	Fedex	OneDay	true

Bad path test plan contains exactly one negative value in each test



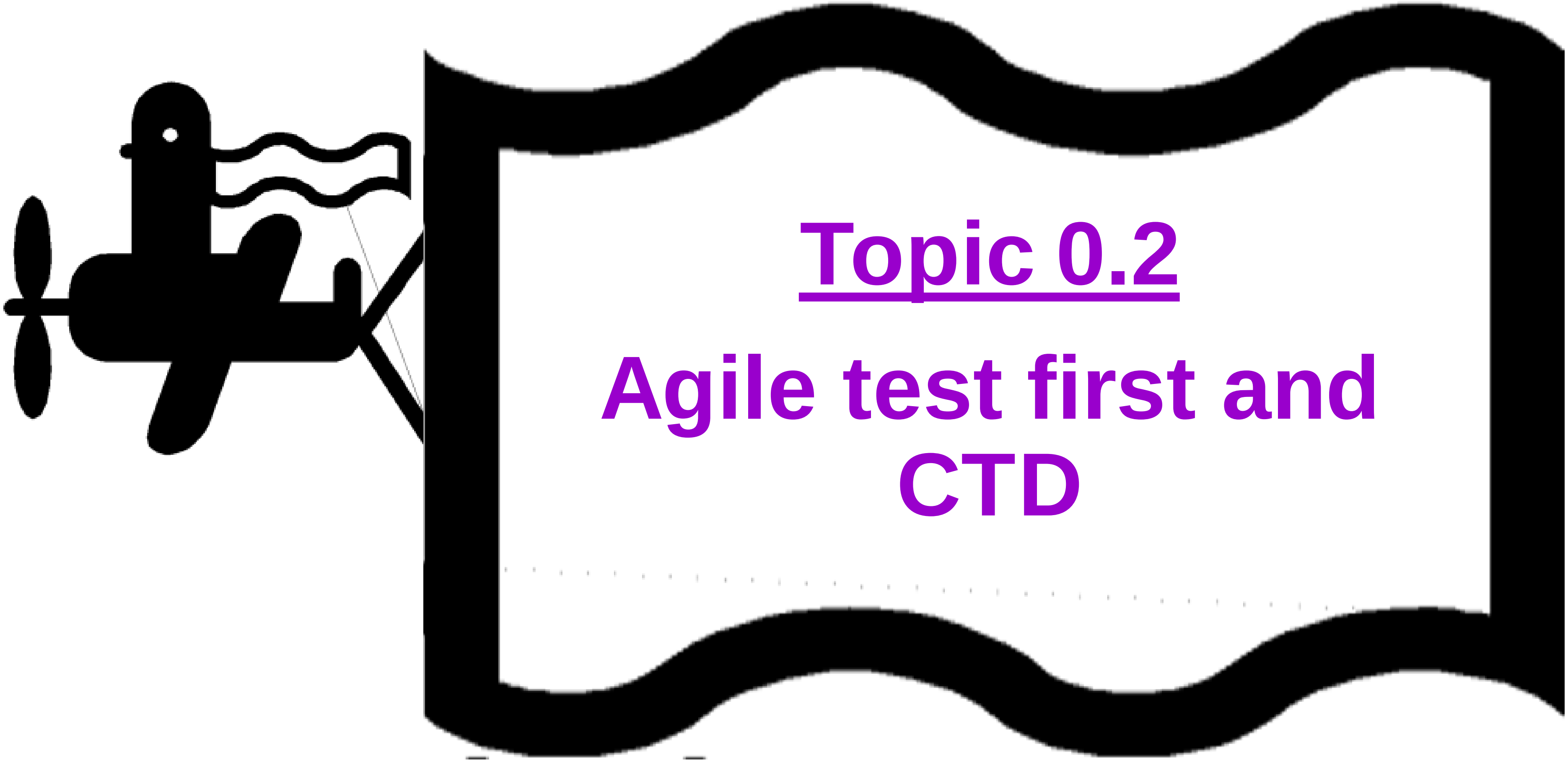
Displaying CTD solution: 13 tasks in good path, 3 tasks in bad path
Actions
Good Path
Bad Path
Export
Modify Test Plan

Availability	Payment	Carrier	DeliverySchedule	ExportControl
NoSuchProduct	Paypal	Mail	2-5WorkingDays	true
NotInStock	Paypal	Fedex	OneDay	true
Discontinued	Credit	Mail	6-10WorkingDays	true

Discussion

- Consider availability=OutOfStock – is this a negative value?

- Depends on how the specific system under test works
- Requires understanding the precise details of the system through interviews and documents



Topic 0.2

Agile test first and CTD

Agile and test driven design (TDD)

- Agile is a dominating software development trend
- Its manifesto emphasizes:
  - “Working software over comprehensive documentation”
  - See Manifesto for Agile Software Development (agilemanifesto.org) for details on the Agile manifesto

- Test first, or test driven design (TDD), is an agile parctice, essentially it constitues of
  - Writing a test
  - Running it and making sure that it fails
  - Implementing the minimal code that makes the test pass
  - Make sure acumlated tests still pass and nothing was broken
  - Refactor the code to keep it clean and run all acumlated tests to make usre they still pass nad nothing was broken

- The software is essentially specified using its test suite

TDD example

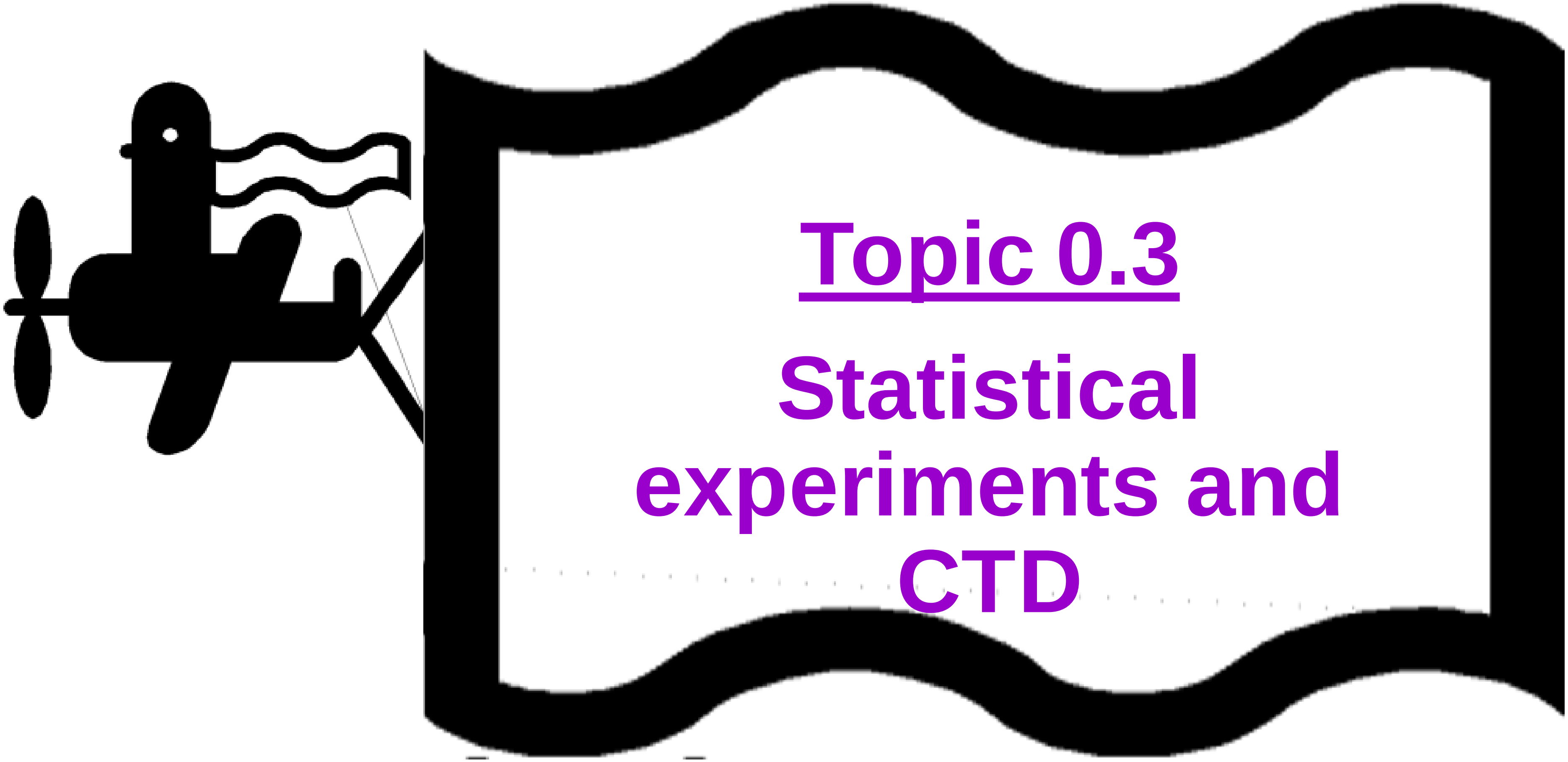
- Requirements:
  - i is initialized to 0
  - N threads execute add(){i++ } atomically and concurrently on the global variable i
  - Result of the execution is N

- Interfaces to be implemented to meet the requirements lock() and unlock()

- Test first –
  - “implement” the empty interfaces, e.g., lock(){}
  - Run the tests that spawn N threads that execute add(){lock(); i++; unlock()}
  - Test execution should fail producing values between 0 and N
    - That indicates that the test is strong enough
  - Next implement the synchronization interfaces lock() and unlock()
  - Rerun the test to see that only the value N is obtained

TDD and CTD

- TDD results in a test suite, but it is not clear what it covers
  - It is a non ambiguous specification, at least for the scenarios it specifies
  - It is not clear what part of the specification or implementation is covered
  - CTD can be used to bridge that gap using the tests created by TDD as a starting point



Topic 0.3

Statistical experiments and CTD