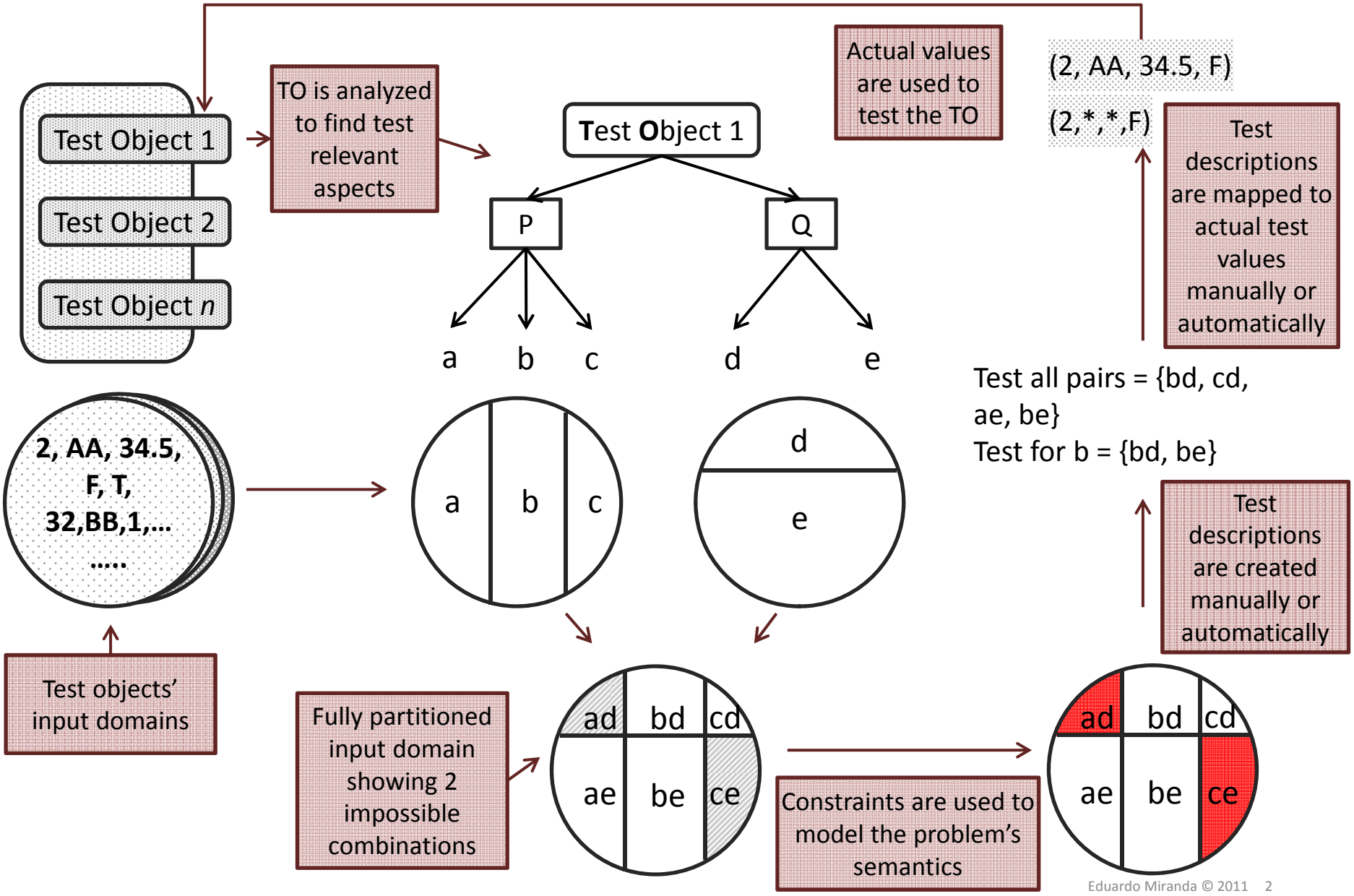
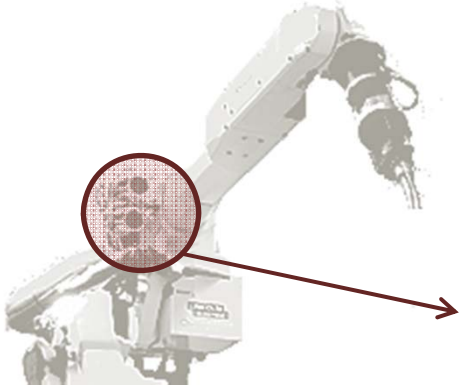
An Illustrated Introduction to the Classification Tree Method

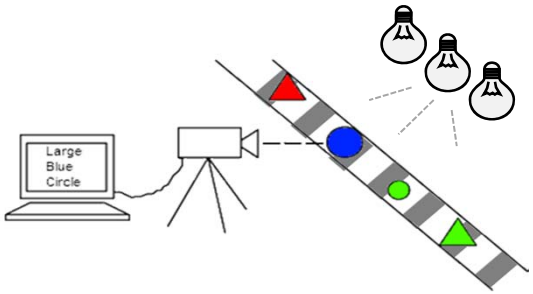
Eduardo Miranda, PhD.
CMU/MSE
MSE/NIST Seminar
June 7th, 2011
Pittsburgh, PA

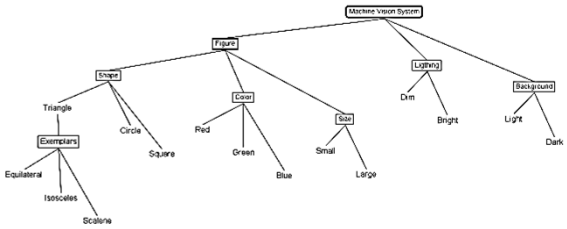
The Classification Tree Method illustrated



The Classification Tree Method in practice



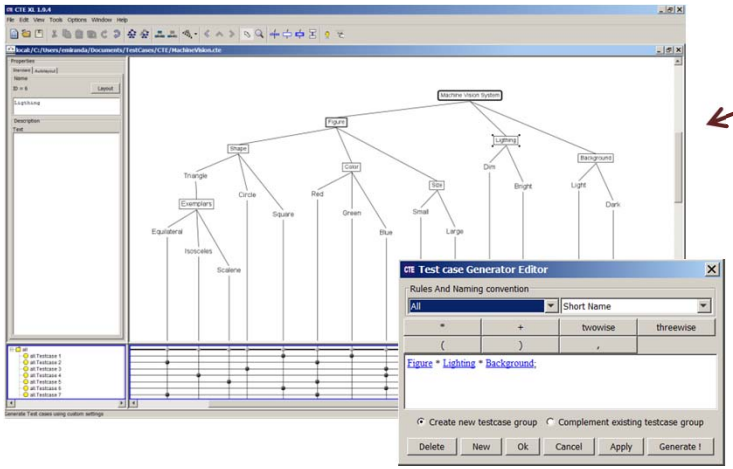




1) Select test object

2) Analyze

3) Model



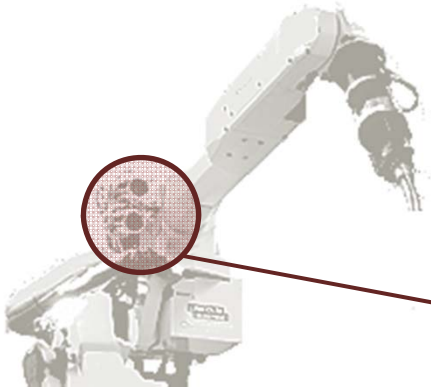
4) Constrain

5) Generate

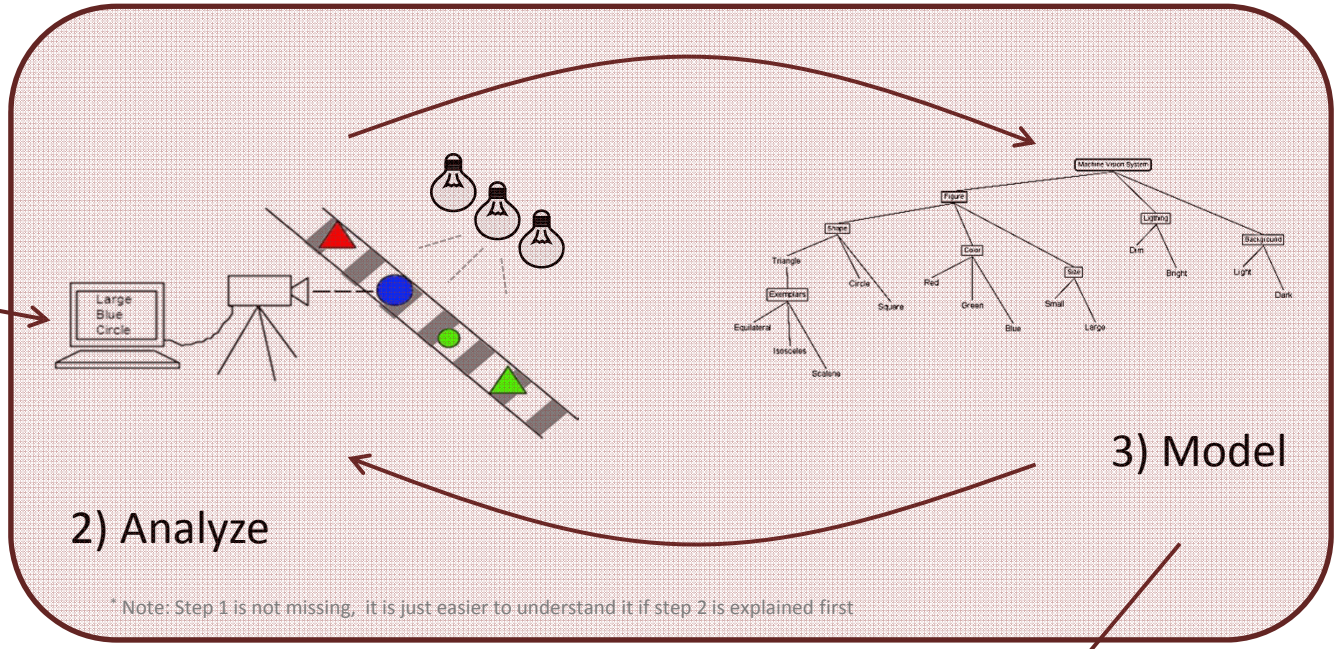
Subject	Description	TestName	StepName	Description (Design Steps)
TrEquivalence		TrEquivalence.Testcase 1	1.1	- Shape: Triangle - Is-a: Equilateral - Color: Red - Background: Light
TrEquivalence		TrEquivalence.Testcase 2	1.2	- Shape: Triangle - Is-a: Isosceles - Color: Red - Background: Light
TrEquivalence		TrEquivalence.Testcase 3	1.3	- Shape: Triangle - Is-a: Scalene - Color: Red - Background: Light
All		All.Testcase 1	2.1	- Shape: Triangle - Is-a: Scalene - Color: Green - Background: Light
All		All.Testcase 2	2.2	- Shape: Circle - Color: Green - Background: Dark
All		All.Testcase 3	2.3	- Shape: Square - Color: Red - Background: Dark
All		All.Testcase 4	2.4	- Shape: Triangle - Is-a: Isosceles - Color: Blue - Background: Light
All		All.Testcase 5	2.5	- Shape: Triangle - Is-a: Scalene - Color: Blue - Background: Dark
All		All.Testcase 6	2.6	- Shape: Triangle - Is-a: Isosceles - Color: Red - Background: Dark
All		All.Testcase 7	2.7	- Shape: Triangle - Is-a: Isosceles - Color: Blue - Background: Light

6) Export

The Classification Tree Method in practice



1) Select test object

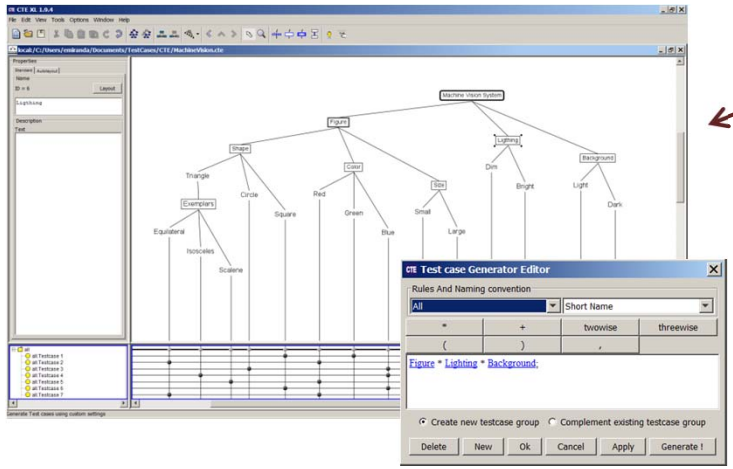


2) Analyze

3) Model

* Note: Step 1 is not missing, it is just easier to understand it if step 2 is explained first

4) Constrain

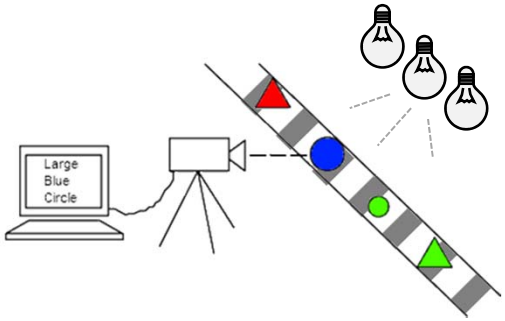


5) Generate

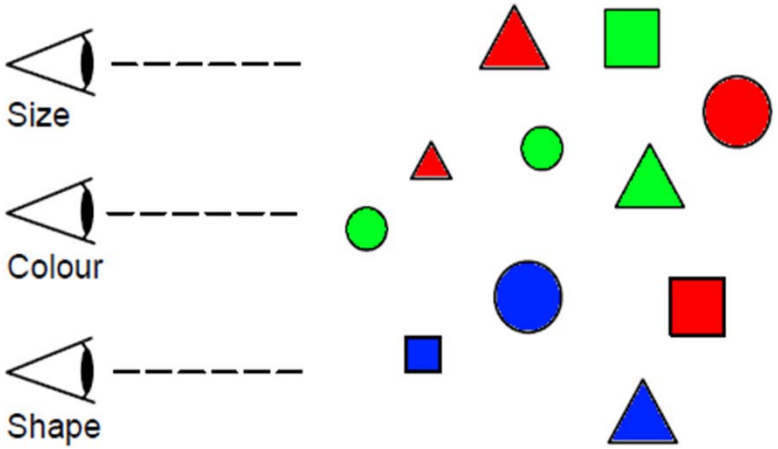
Subject	Description	TestName	StepName	Description (Design Steps)
TrEquivalence		TrEquivalence.Testcase 1	1.1	- Shape: Triangle - Is-a: Equilateral - Color: Red - Background: Light
TrEquivalence		TrEquivalence.Testcase 2	1.2	- Shape: Triangle - Is-a: Isosceles - Color: Red - Background: Light
TrEquivalence		TrEquivalence.Testcase 3	1.3	- Shape: Triangle - Is-a: Scalene - Color: Red - Background: Light
All		All.Testcase 1	2.1	- Shape: Triangle - Is-a: Scalene - Color: Green - Background: Light
All		All.Testcase 2	2.2	- Shape: Circle - Color: Green - Background: Dark
All		All.Testcase 3	2.3	- Shape: Square - Color: Red - Background: Dark
All		All.Testcase 4	2.4	- Shape: Triangle - Is-a: Isosceles - Color: Blue - Background: Light
All		All.Testcase 5	2.5	- Shape: Triangle - Is-a: Scalene - Color: Blue - Background: Dark
All		All.Testcase 6	2.6	- Shape: Triangle - Is-a: Isosceles - Color: Red - Background: Dark
All		All.Testcase 7	2.7	- Shape: Triangle - Is-a: Isosceles - Color: Blue - Background: Light

6) Export

Machine vision subsystem: Identifying test-relevant aspects

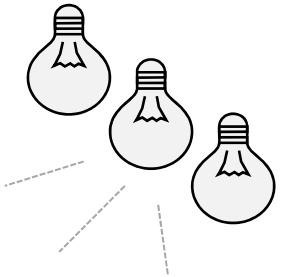


The vision subsystem shall classify figures of different:

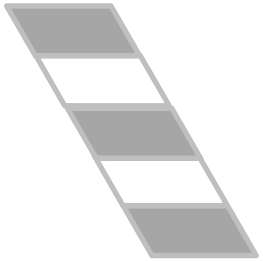


Adapted from M. Grochtmann, Test Case Design Using Classification Trees, 1994

Is the capability of the system affected by:



the lighting conditions?



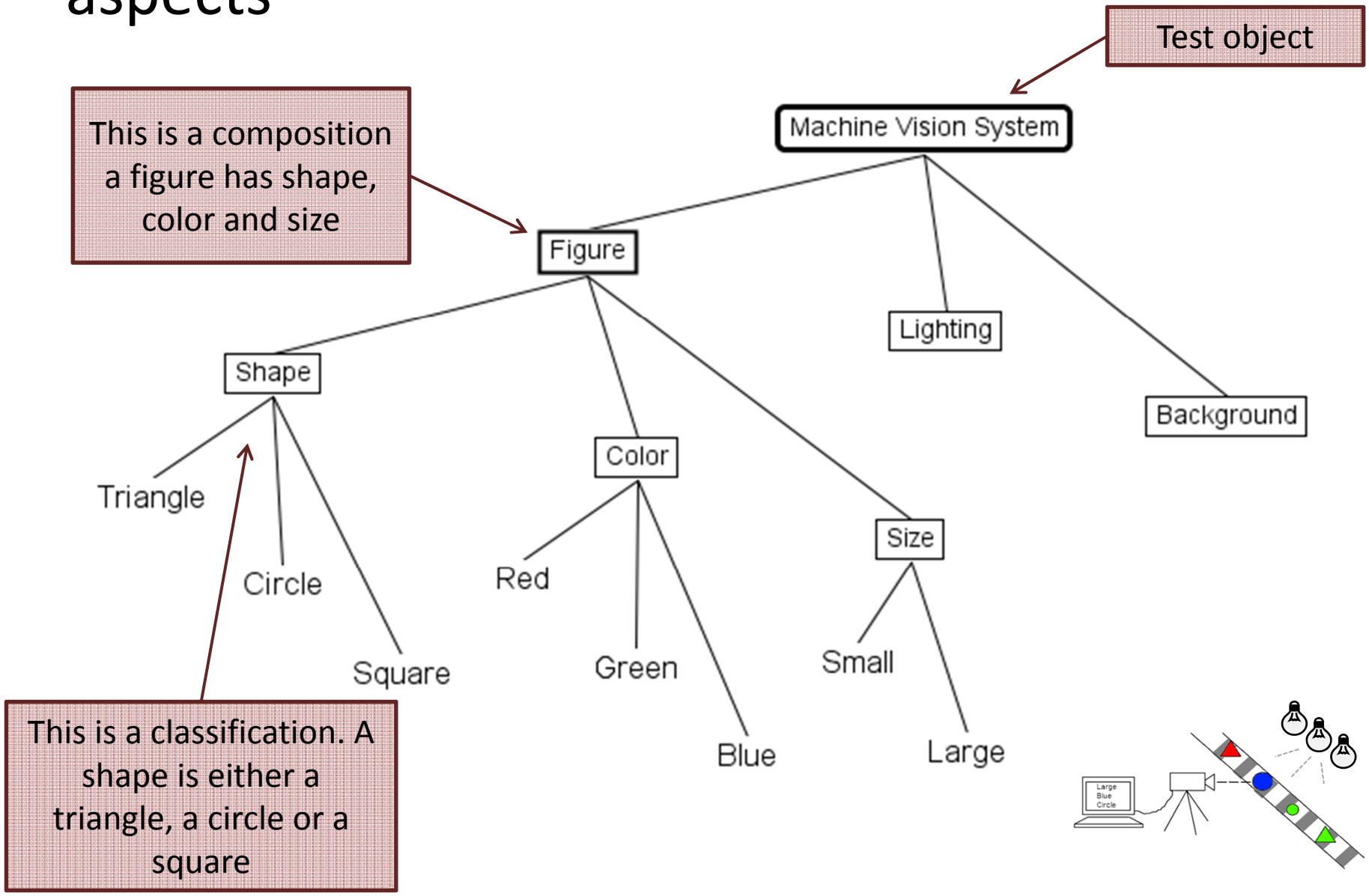
the conveyor background?

the speed at which the figures pass in front of the camera?

Classification trees' syntax & semantics: Root, compositions, classifiers and classes

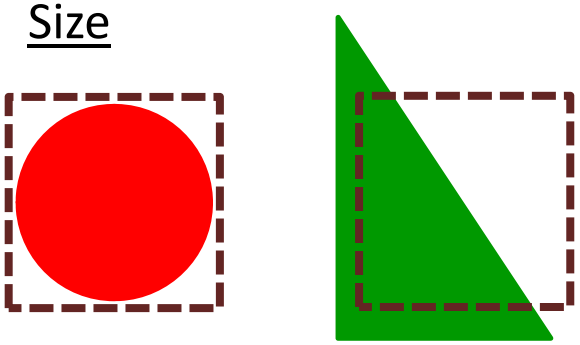
Modeling entities	Uses	Allowed Parents	Allowed descendants	Test Selectable
Root	•Denotes the test object (TO)		•Compositions •Classifications	No
Compositions	•Consist-of relationship •To model an aspect that is broken down into other aspects •To refine a value consisting of several parts	•Root •Composition •Class	•Composition •Classification	No
Classifications	•Is-a relationship •To model an aspect that is partitioned into a collectively exhaustive and mutually exclusive sets of values (equivalence classes)	•Root •Composition •Class	•Class	No
Classes	To model equivalence classes (set of values that elicits the same behavior from the TO) and actual test values	•Classification	•Composition •Classification	Yes

Machine vision system: Modeling test-relevant aspects



Equivalence classes

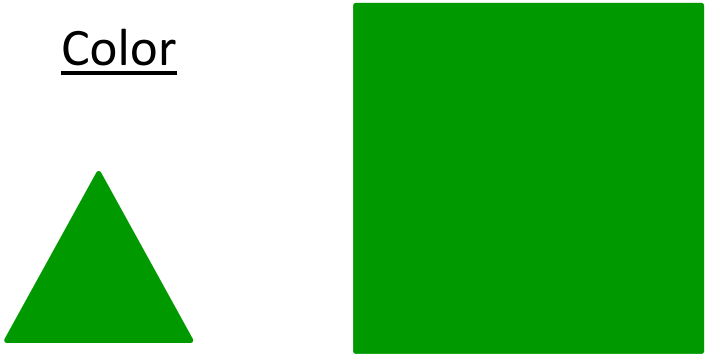
Size



Small Large

A figure is small if it is inscribed in a square area of 300 x 300 pixels otherwise is large

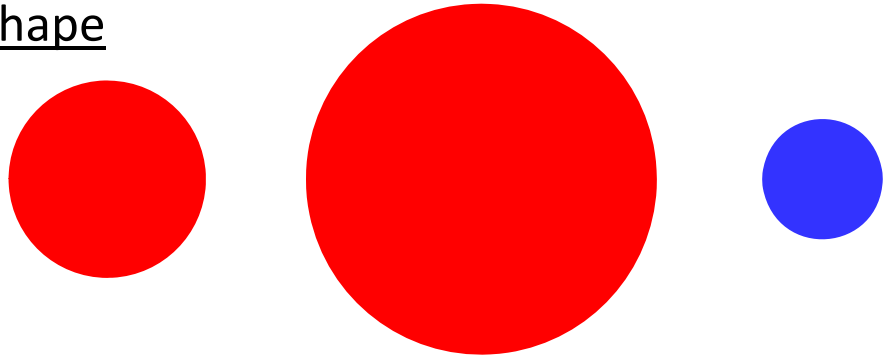
Color



Green

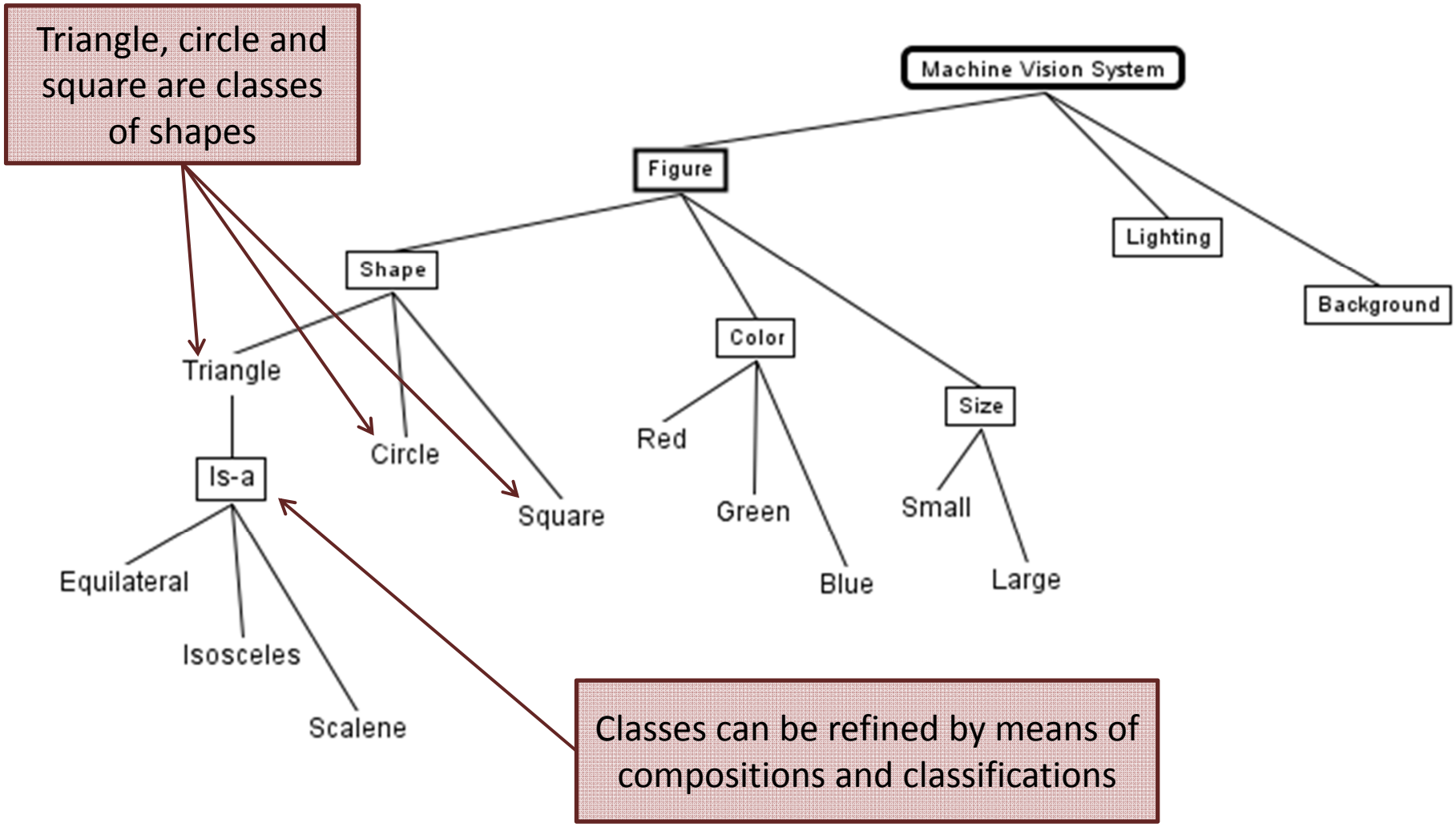
Each color should be treated the same independently of the size and shape of the figure

Shape

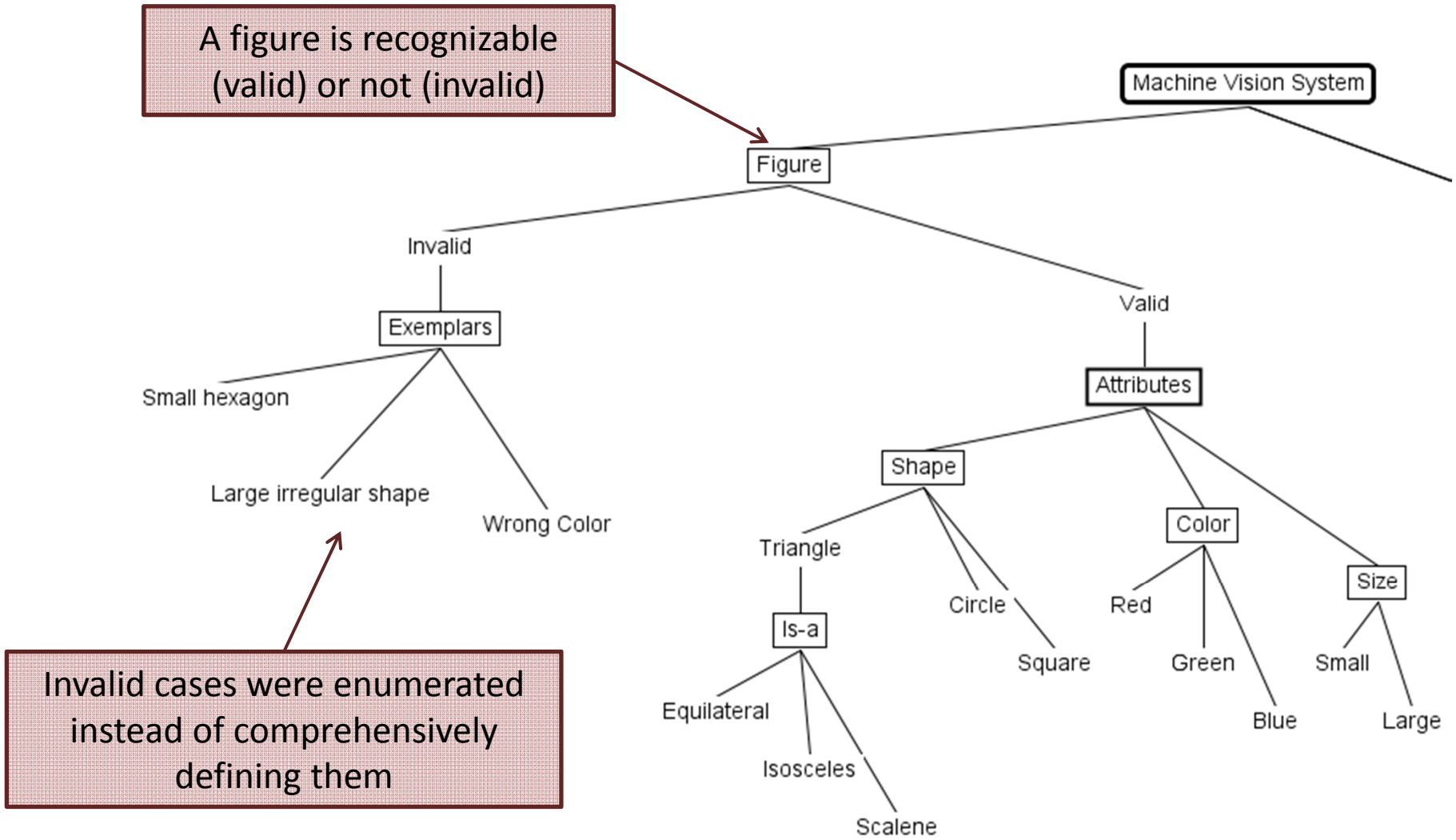


All these are circles

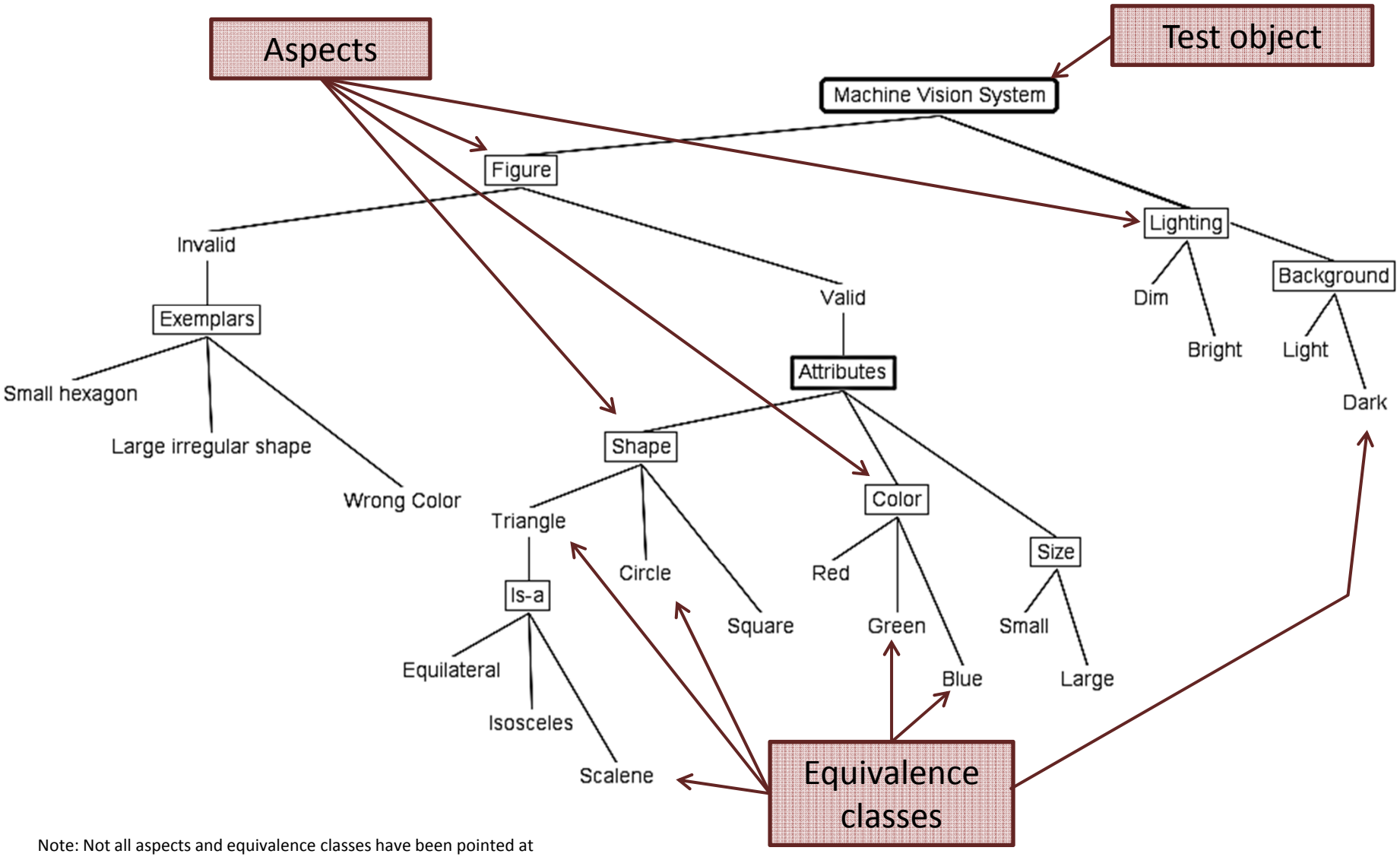
Machine vision system: Partitioning test-relevant aspects



What happens if the figure is not of the right shape or the right color?



Machine vision system: The complete picture

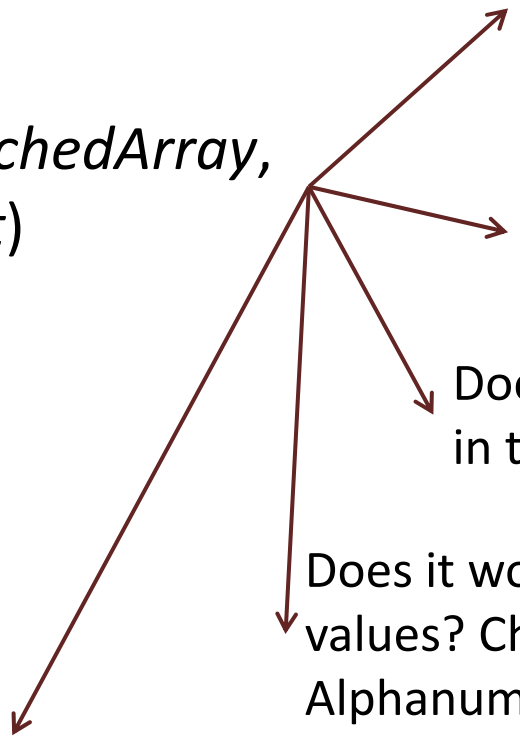


Note: Not all aspects and equivalence classes have been pointed at

Count function

The count function shall provide a tally of the number of occurrences of a given value in an array

count (IN array: *searchedArray*, IN string: *countWhat*)



Do the number of elements in the array have any bearing in the processing?

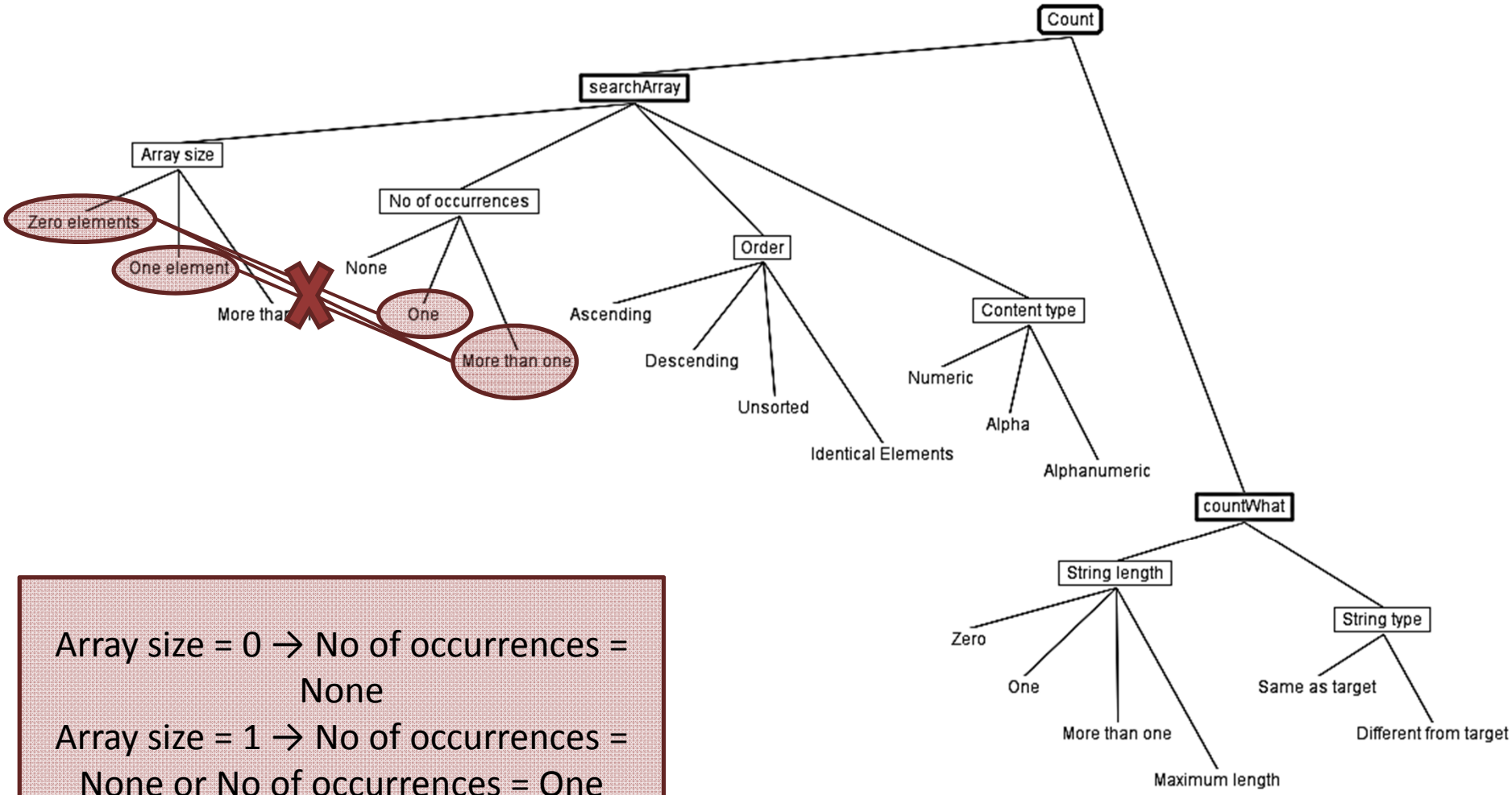
Does the function correctly tally the number of occurrences of the searched element in the array?

Does the order of the elements in the array affect the results?

Does it work with numerical values? Characters? Alphanumeric?

What does it happen if the *countWhat* string is null?

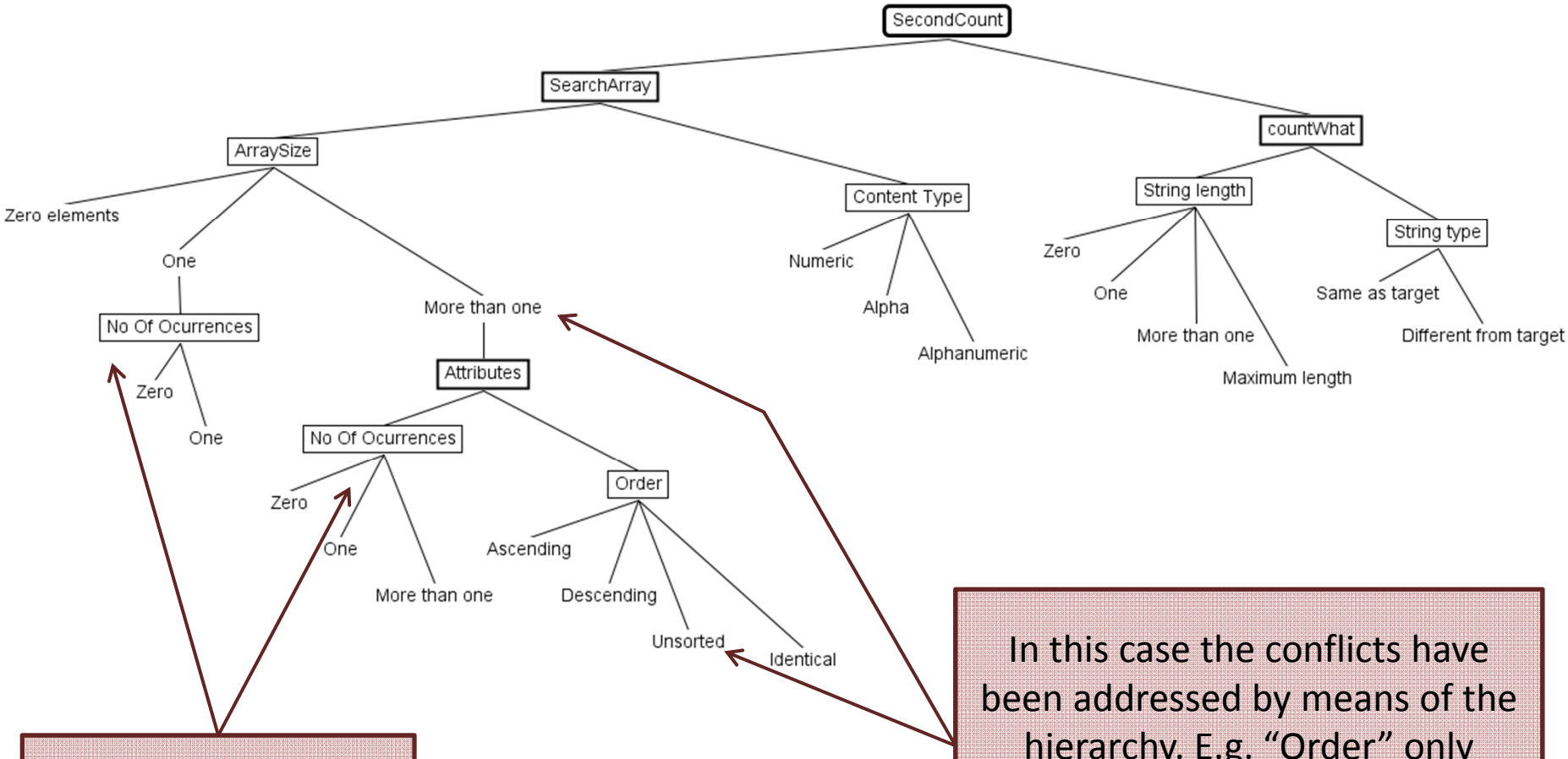
Count function: Classification tree showing potential conflicts among equivalence classes



Array size = 0 → No of occurrences = None

Array size = 1 → No of occurrences = None or No of occurrences = One

A different partition for the *Count* classification tree

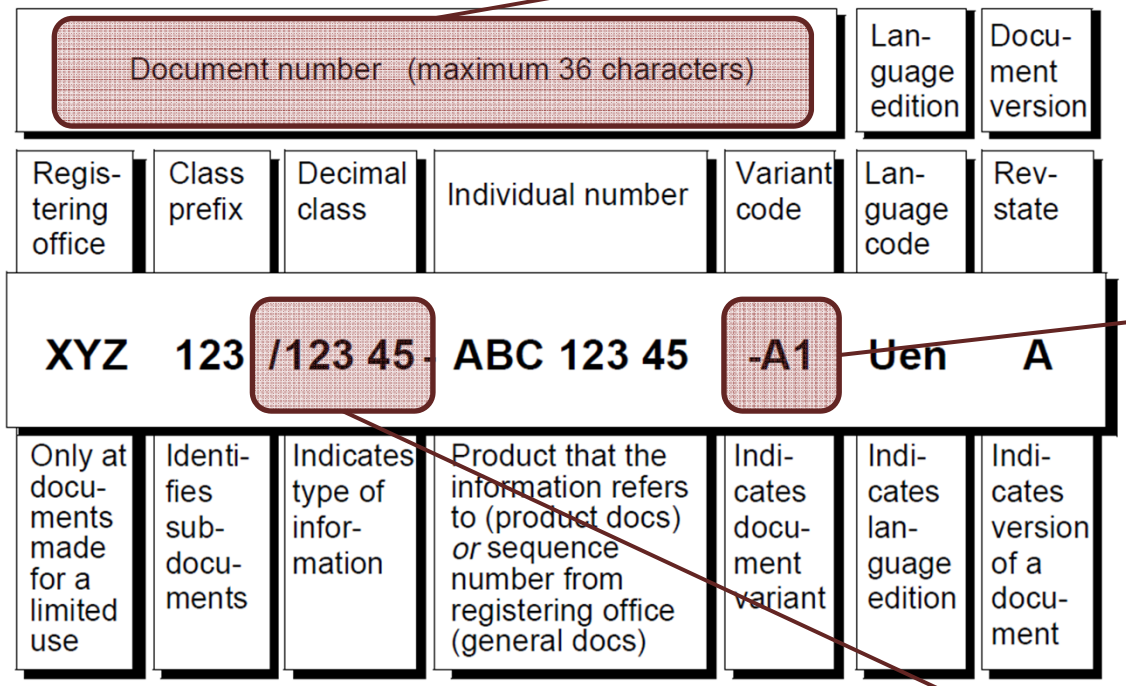


Some aspects are duplicated

In this case the conflicts have been addressed by means of the hierarchy. E.g. “Order” only makes sense if the array has two or more elements

Document filing system

What happens if the length of the document number is more than 36 characters? What is the minimum?



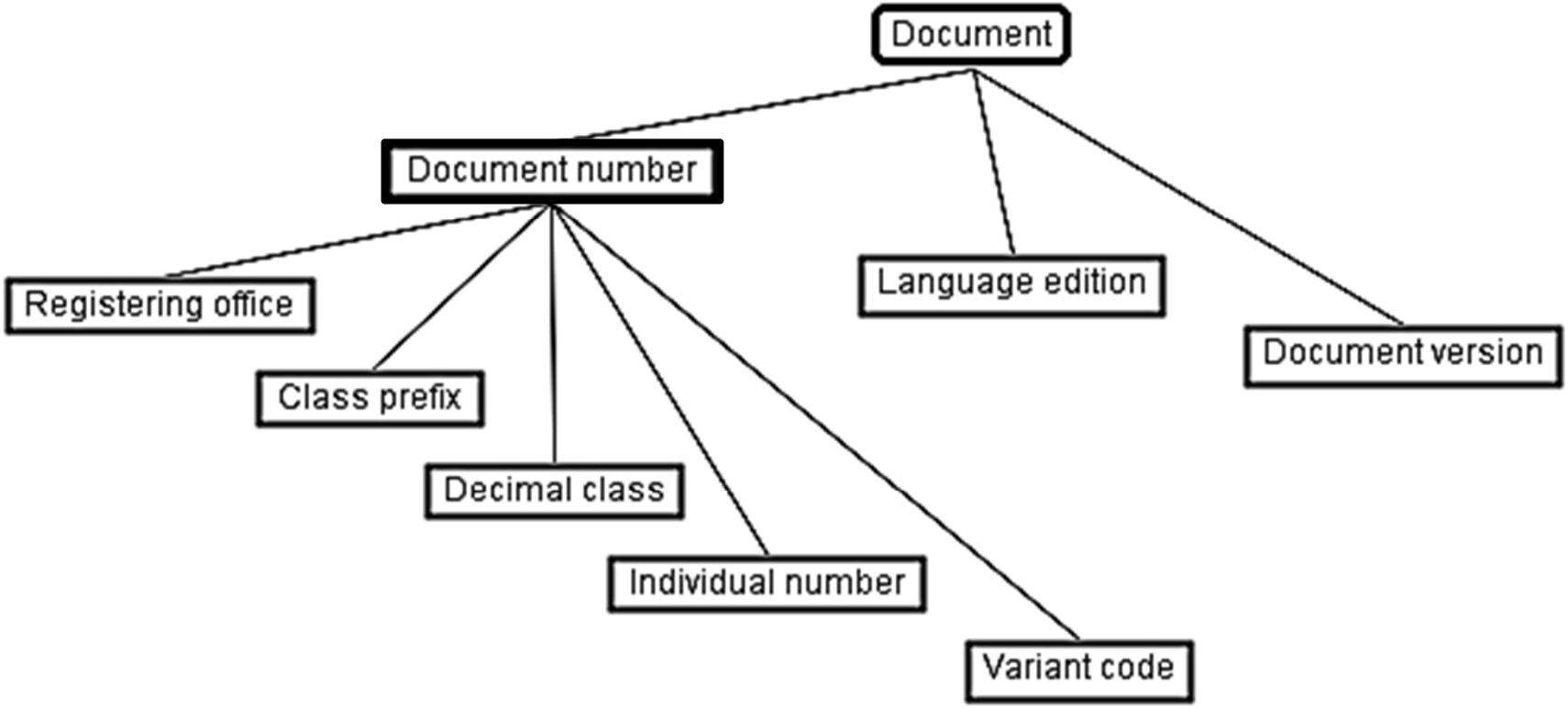
Does the system do anything different with these variants?

Examples of document numbers could be

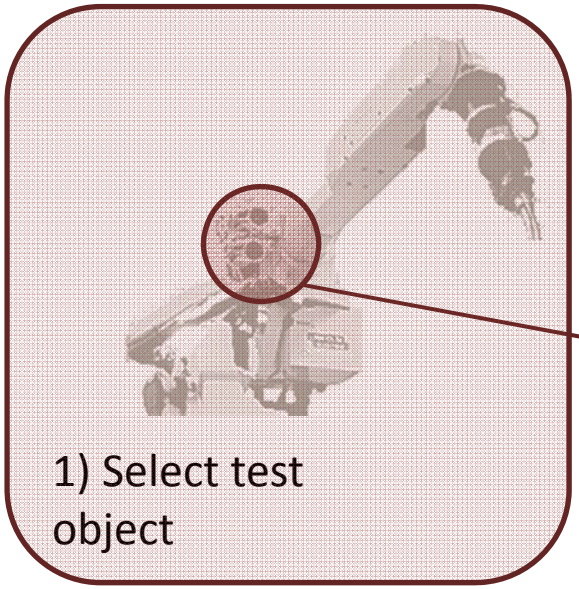
- 109 21- ASB 501 04, Product Revision Information;
- 131 32 -ROF 137 5054/2, Manufacturing Specification;
- 102 62-CAA 111 1305, Design Specification;
- 1050-EN/LZB 103 04, Description.

Does the system do anything different based on the decimal class of the document?

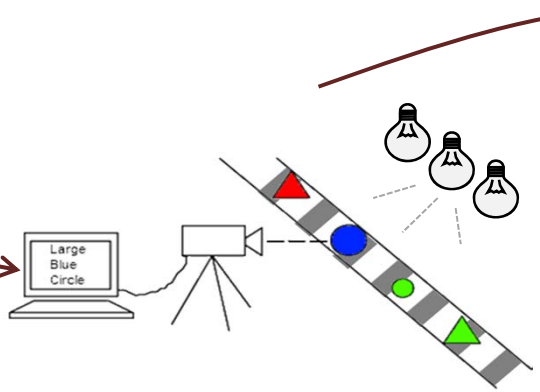
Document filing system: Classification tree with relevant test aspects



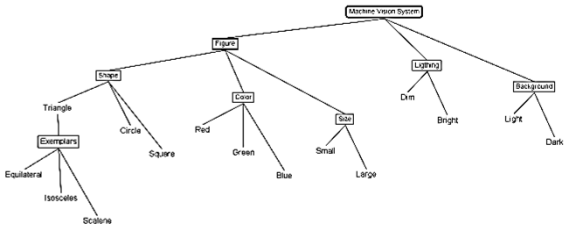
The Classification Tree Method in practice



1) Select test object



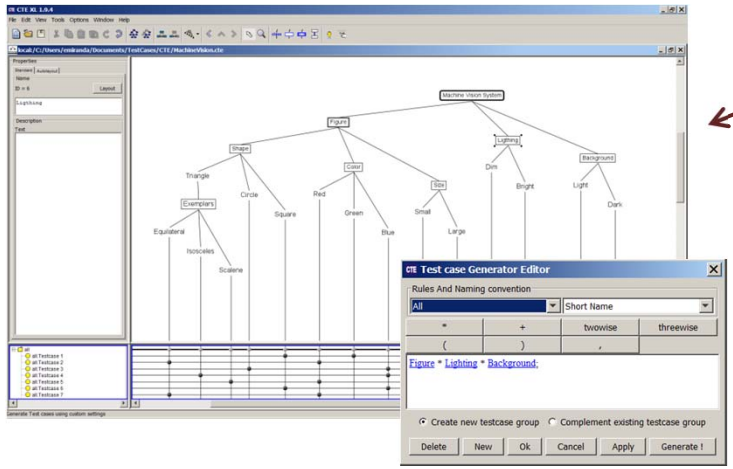
2) Analyze



3) Model

Subject	Description	TestName	StepName	Description (Design Steps)
TrEquivalence		TrEquivalence.Testcase 1	1.1	- Shape: Triangle - Is-a: Equilateral - Color: Red - Background: Light
TrEquivalence		TrEquivalence.Testcase 2	1.2	- Shape: Triangle - Is-a: Isosceles - Color: Red - Background: Light
TrEquivalence		TrEquivalence.Testcase 3	1.3	- Shape: Triangle - Is-a: Scalene - Color: Red - Background: Light
All		All.Testcase 1	2.1	- Shape: Triangle - Is-a: Scalene - Color: Green - Background: Light
All		All.Testcase 2	2.2	- Shape: Circle - Color: Green - Background: Dark
All		All.Testcase 3	2.3	- Shape: Square - Color: Red - Background: Dark
All		All.Testcase 4	2.4	- Shape: Triangle - Is-a: Isosceles - Color: Blue - Background: Light
All		All.Testcase 5	2.5	- Shape: Triangle - Is-a: Scalene - Color: Blue - Background: Dark
All		All.Testcase 6	2.6	- Shape: Triangle - Is-a: Isosceles - Color: Red - Background: Dark
All		All.Testcase 7	2.7	- Shape: Triangle - Is-a: Isosceles - Color: Blue - Background: Light

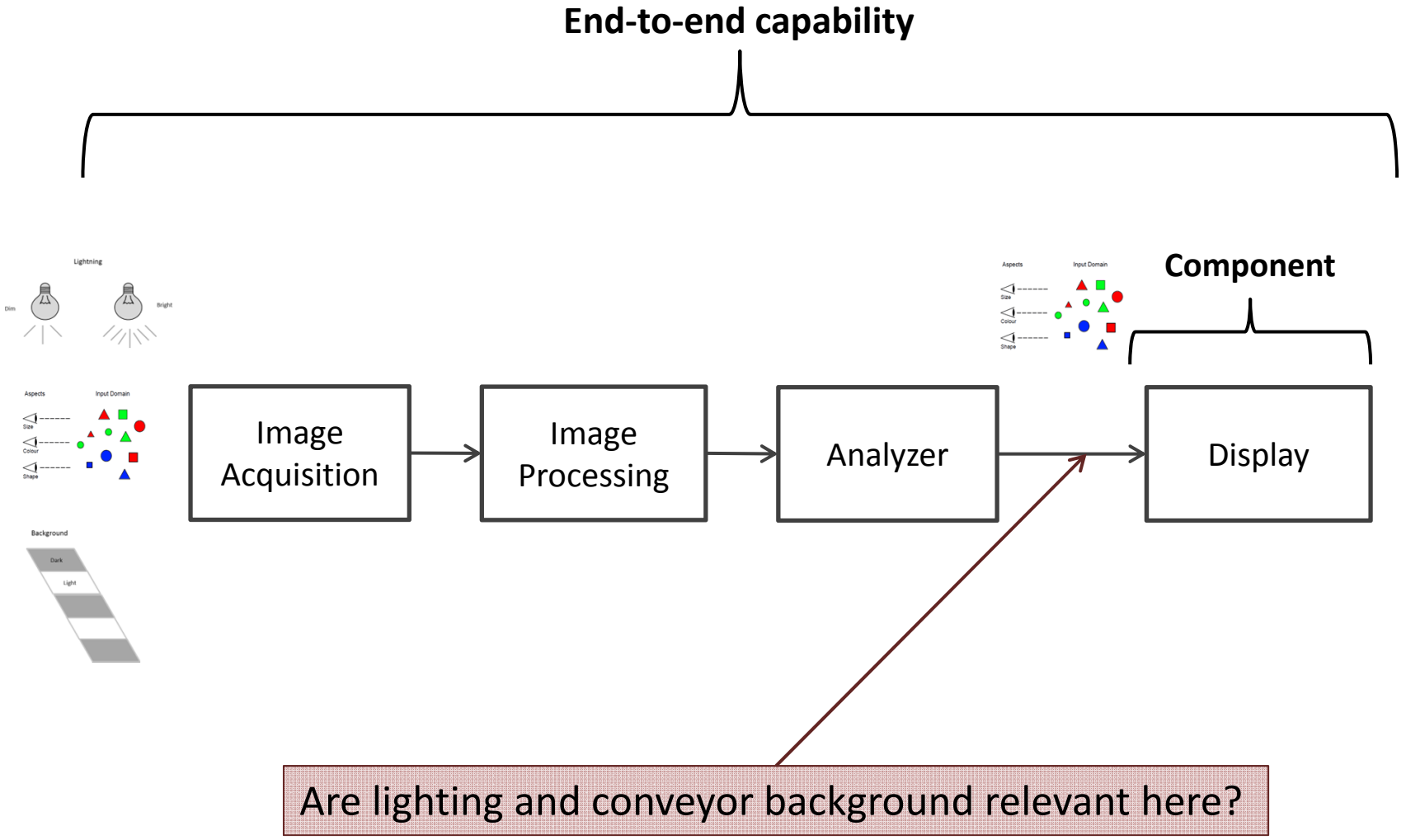
6) Export



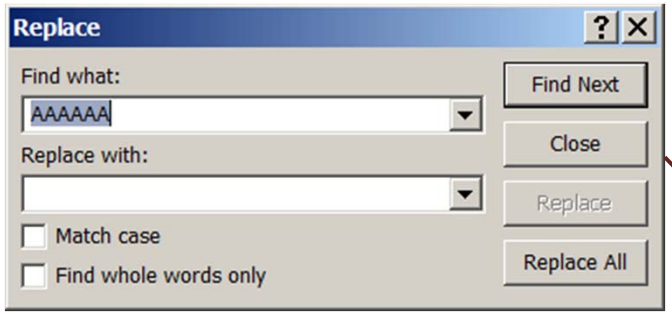
5) Generate

4) Constrain

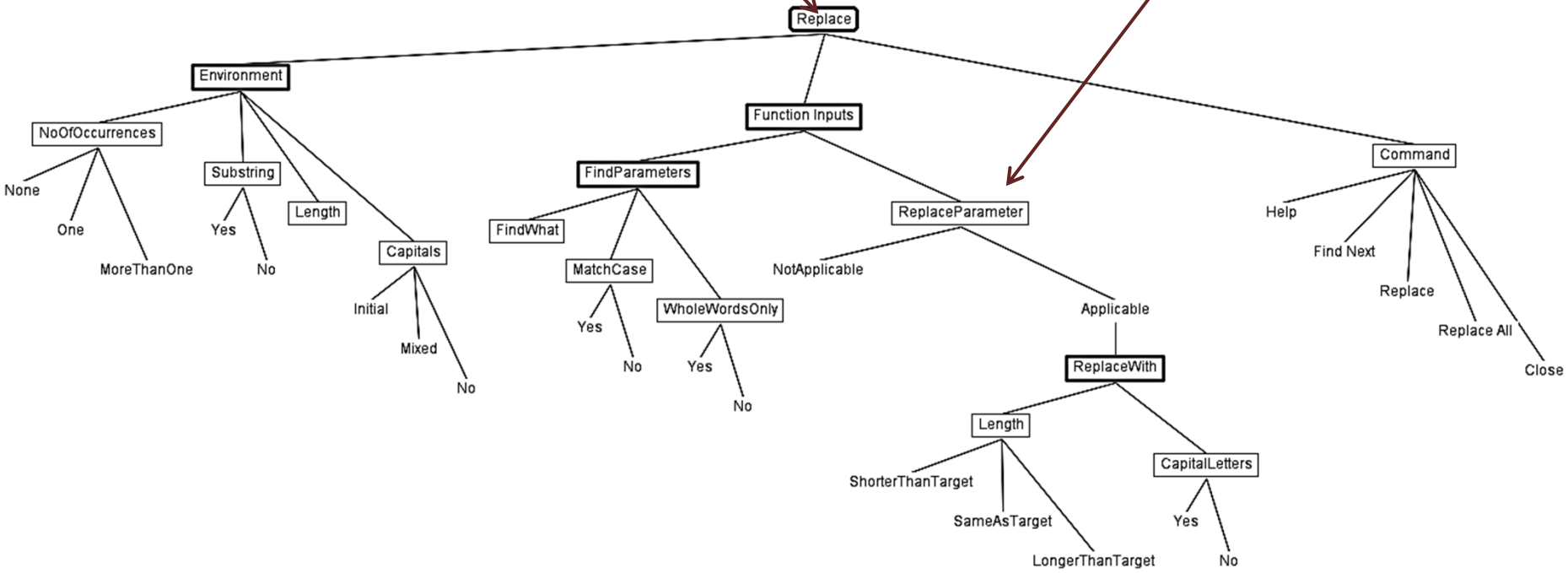
Implications of test object selection on aspect relevance



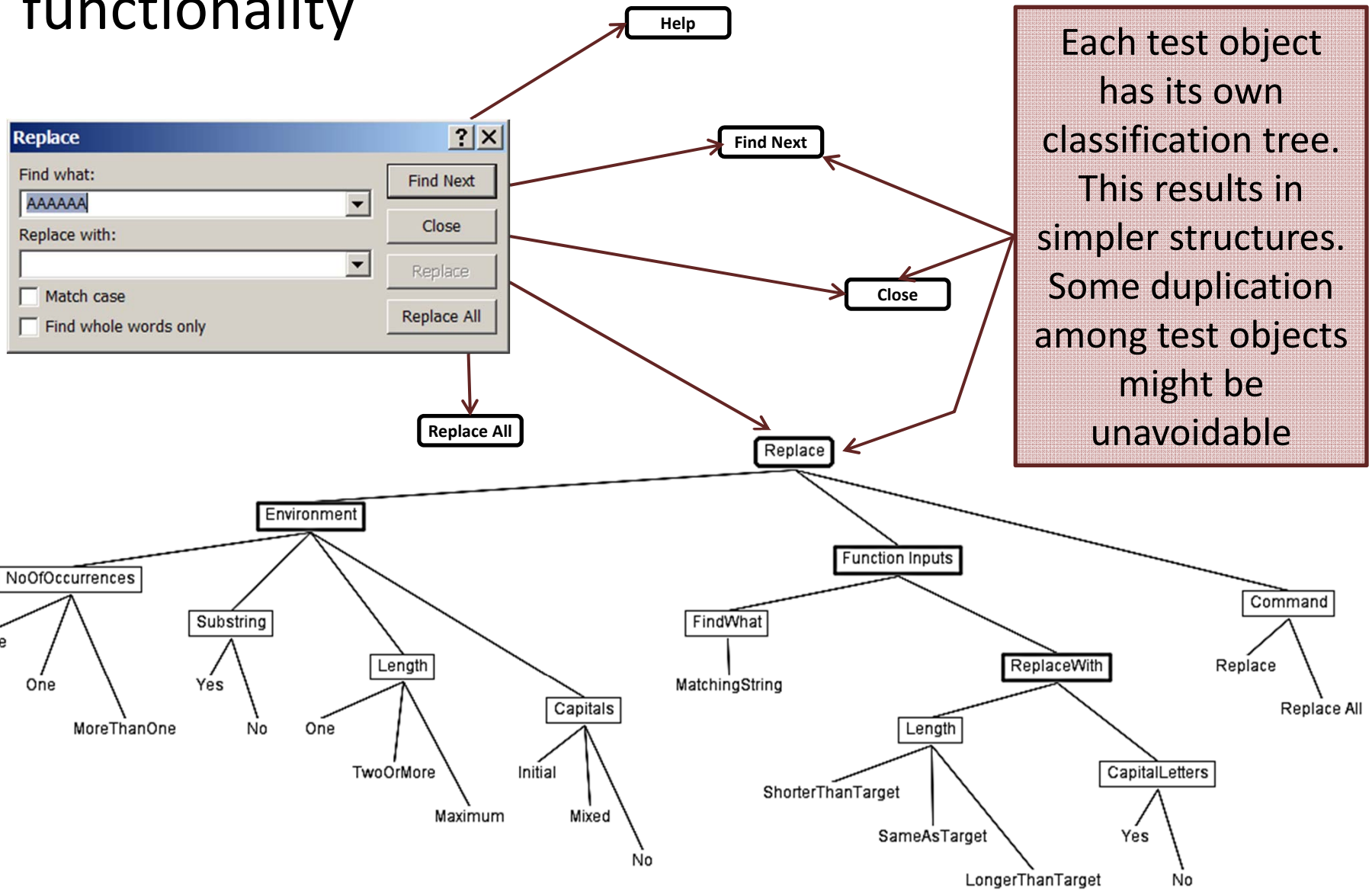
Is “Replace” a test object?



Complicated tree structure. Will require many constraints to avoid generation of “not applicable” and impossible combinations in the test specifications

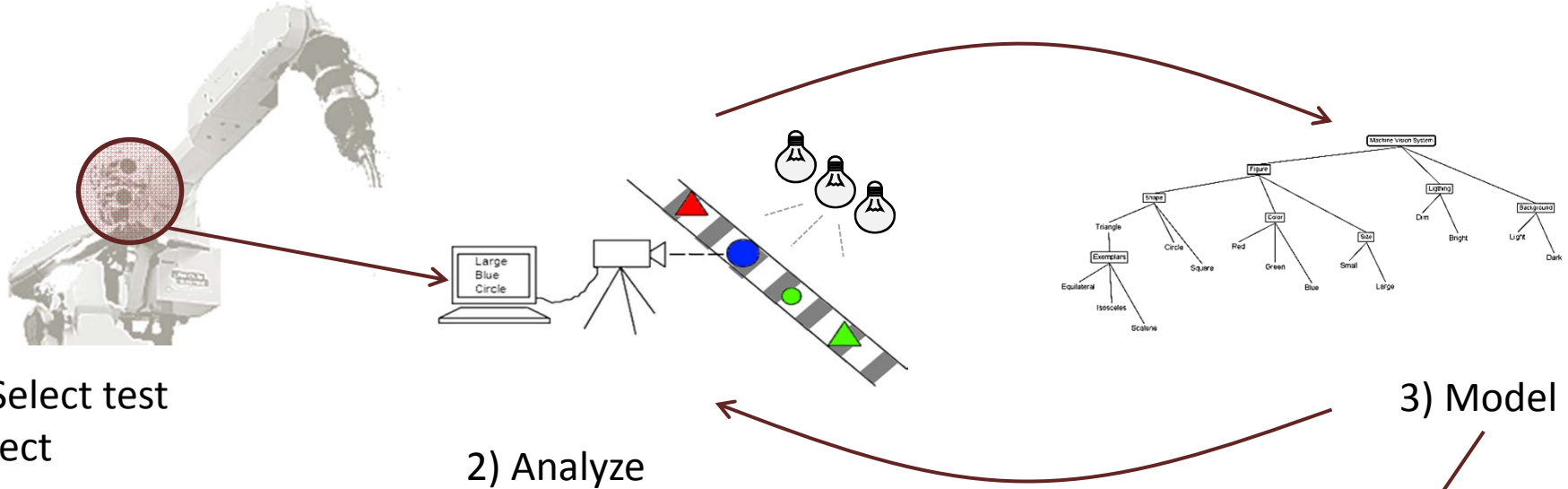


The “Replace” form encompasses multiple functionality



Each test object has its own classification tree. This results in simpler structures. Some duplication among test objects might be unavoidable

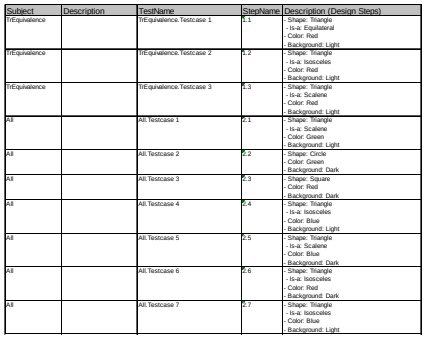
The Classification Tree Method in practice



1) Select test object

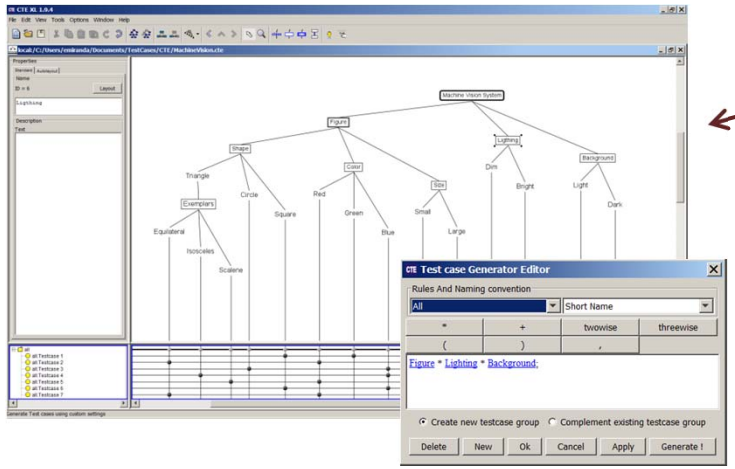
2) Analyze

3) Model

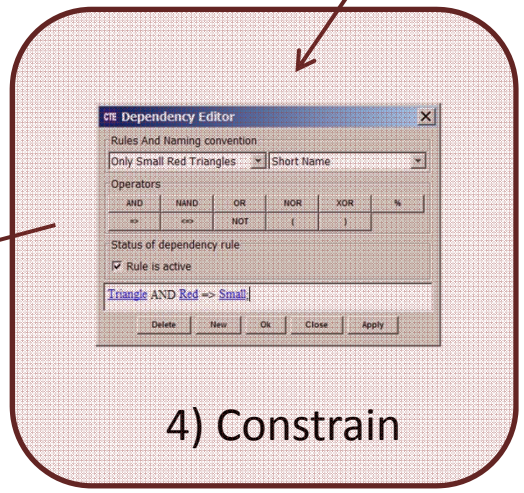


Subject	Description	TestName	StepName	Description (Design Steps)
TrEquivalence		TrEquivalence.Testcase 1	5.1	- Shape: Triangle - Is-a: Equilateral - Color: Red - Background: Light
TrEquivalence		TrEquivalence.Testcase 2	5.2	- Shape: Triangle - Is-a: Isosceles - Color: Red - Background: Light
TrEquivalence		TrEquivalence.Testcase 3	5.3	- Shape: Triangle - Is-a: Scalene - Color: Red - Background: Light
All		All.Testcase 1	5.1	- Shape: Triangle - Is-a: Scalene - Color: Green - Background: Light
All		All.Testcase 2	5.2	- Shape: Circle - Color: Green - Background: Dark
All		All.Testcase 3	5.3	- Shape: Square - Color: Red - Background: Dark
All		All.Testcase 4	5.4	- Shape: Triangle - Is-a: Isosceles - Color: Blue - Background: Light
All		All.Testcase 5	5.5	- Shape: Triangle - Is-a: Scalene - Color: Blue - Background: Dark
All		All.Testcase 6	5.6	- Shape: Triangle - Is-a: Isosceles - Color: Red - Background: Dark
All		All.Testcase 7	5.7	- Shape: Triangle - Is-a: Isosceles - Color: Blue - Background: Light

6) Export

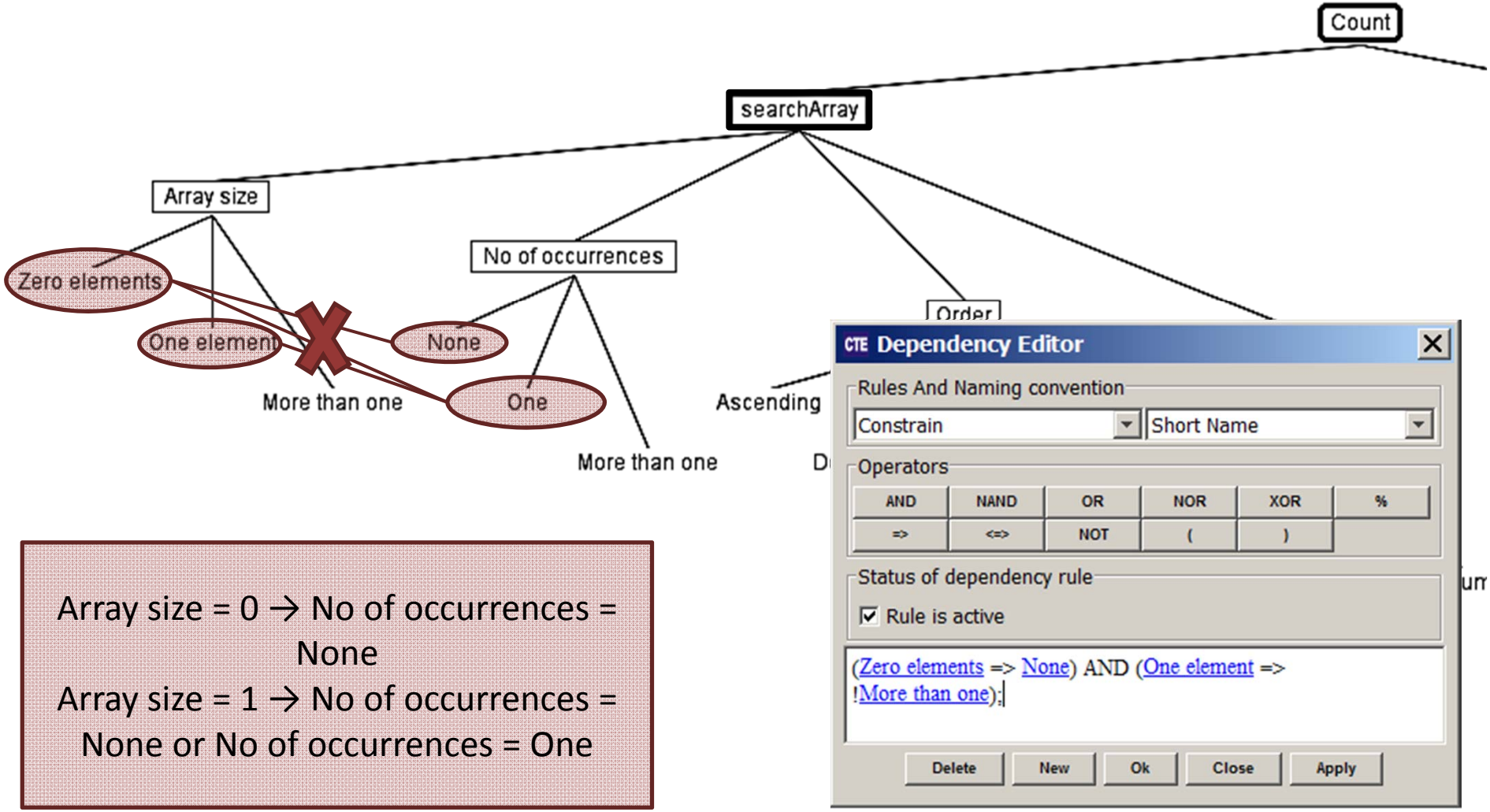


5) Generate



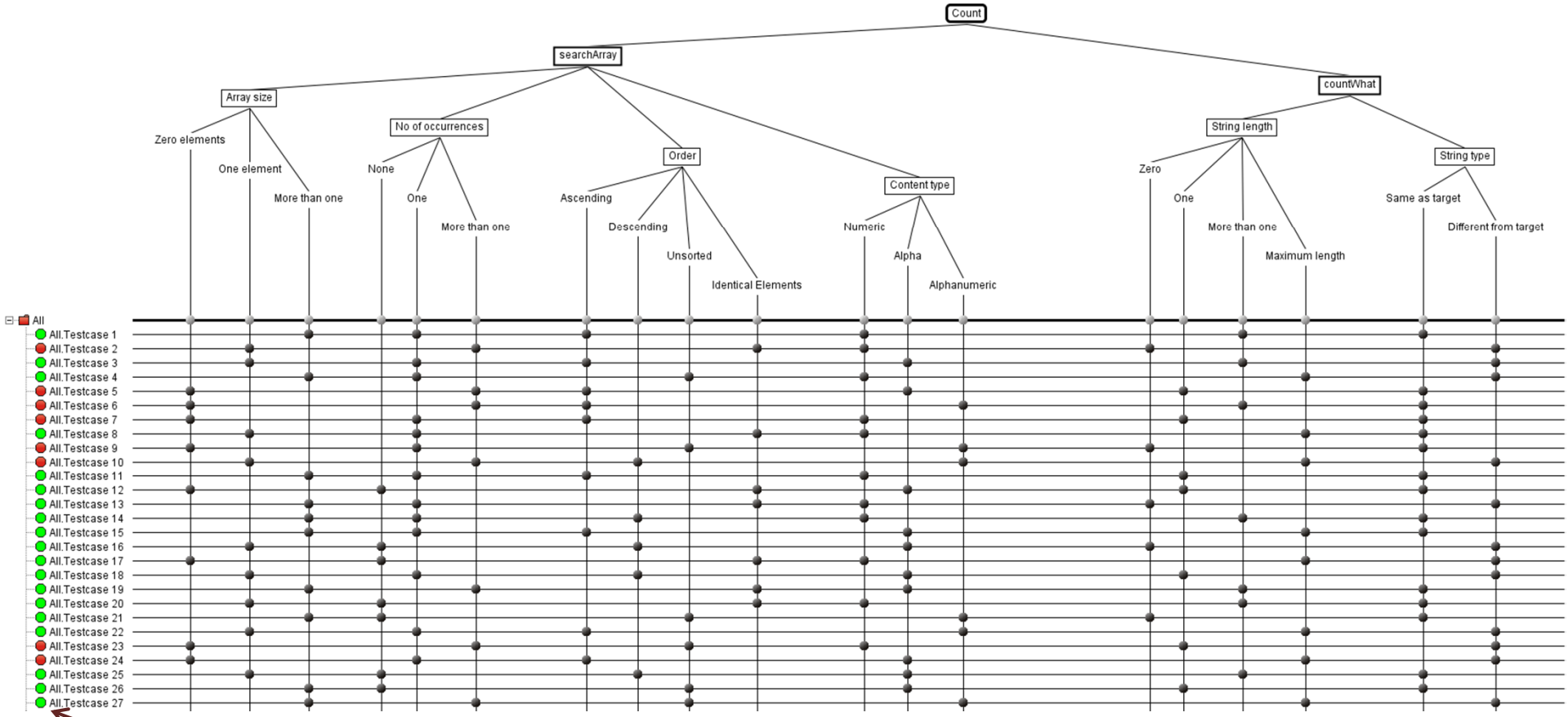
4) Constrain

Count function: Classification tree showing potential conflicts among equivalence classes



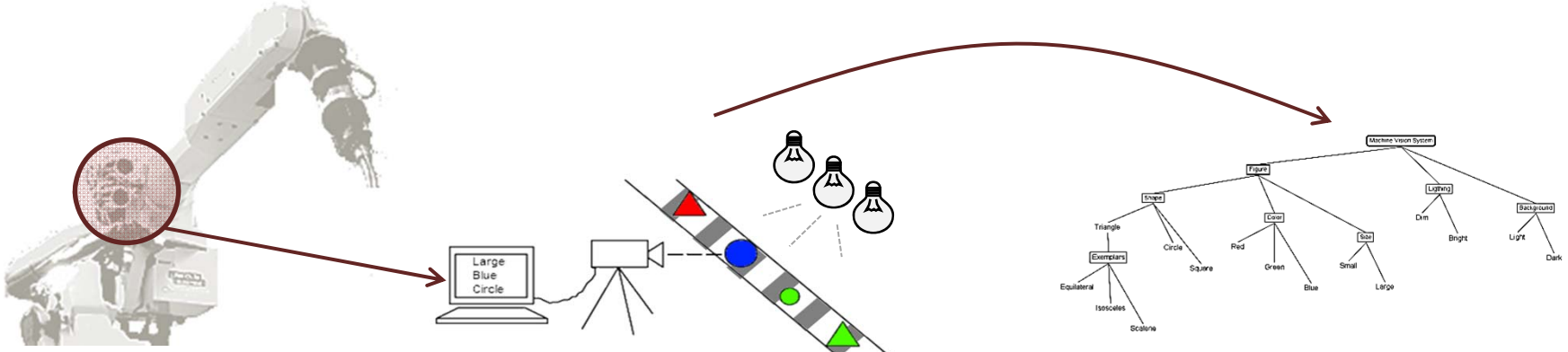
Array size = 0 → No of occurrences = None
Array size = 1 → No of occurrences = None or No of occurrences = One

Conflicts are flagged



Lines that yield a true value for all the constrains specified are flagged with green

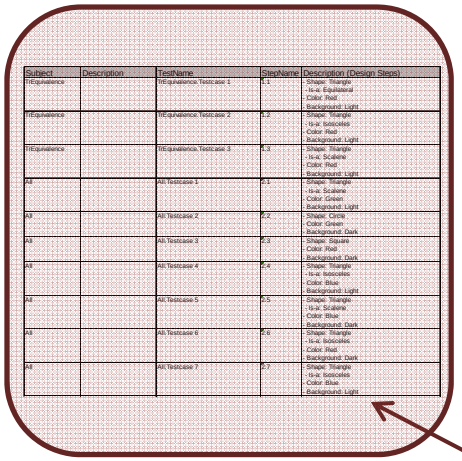
The Classification Tree Method in practice



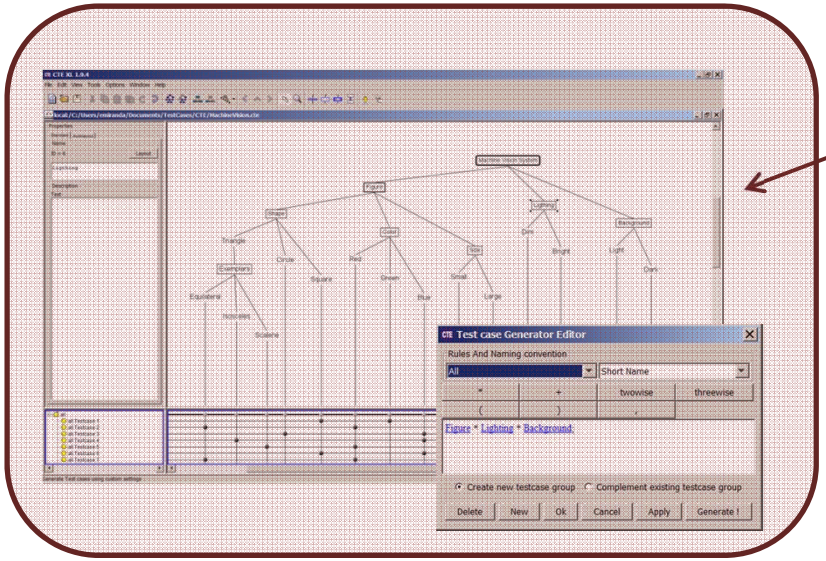
1) Select test object

2) Analyze

3) Model



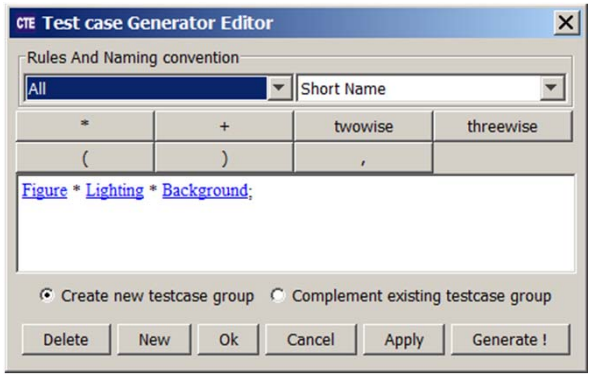
6) Export

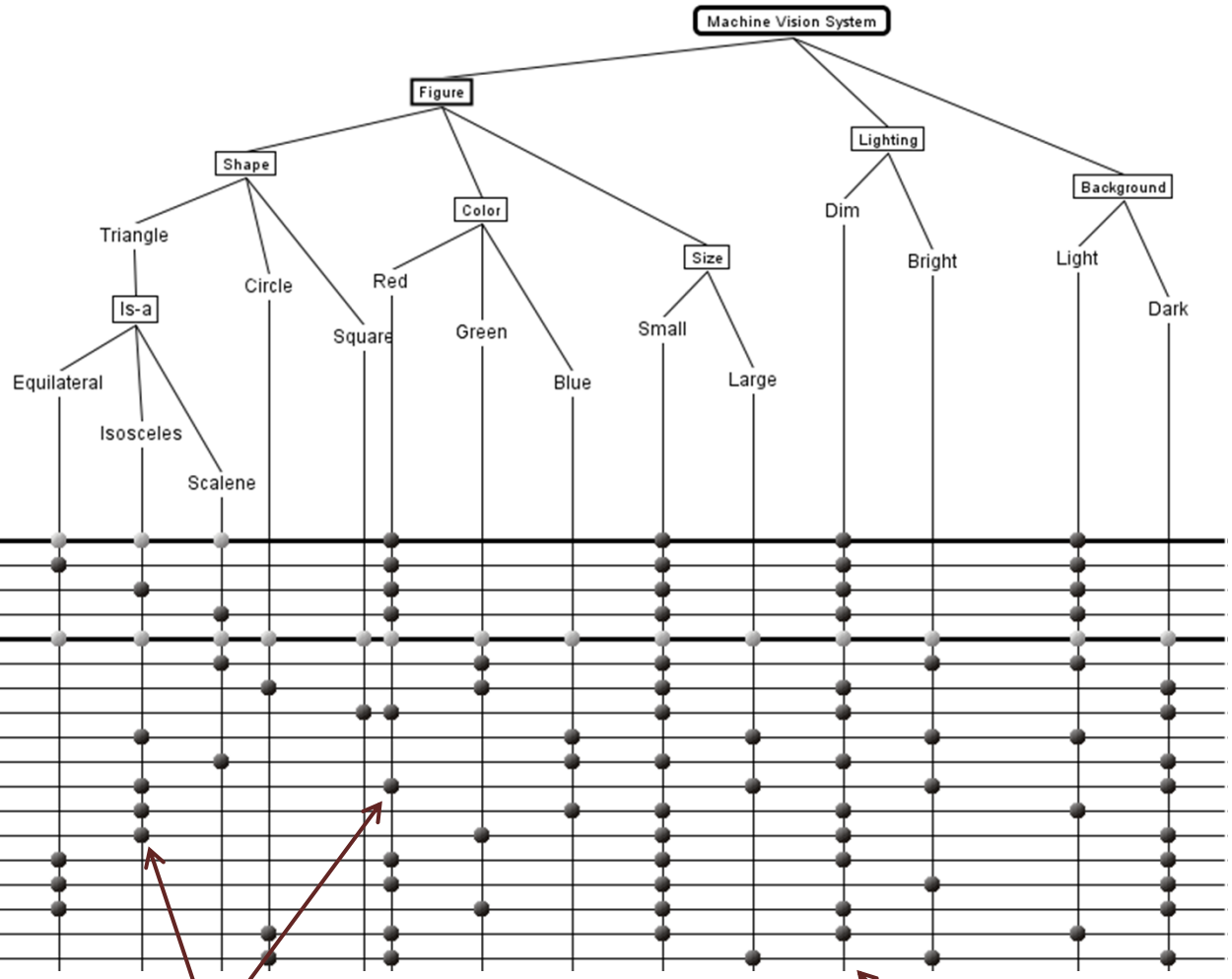


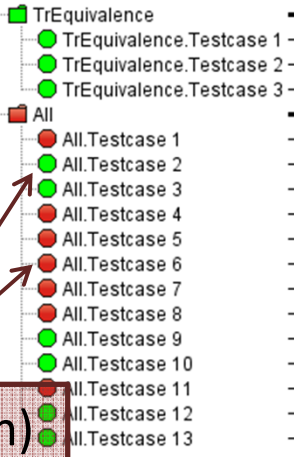
5) Generate

4) Constrain

Generating test specifications / test cases







Test groups

Valid (green) and invalid (red) specifications

Selection marks. Allow the manual selection of values

Combination table

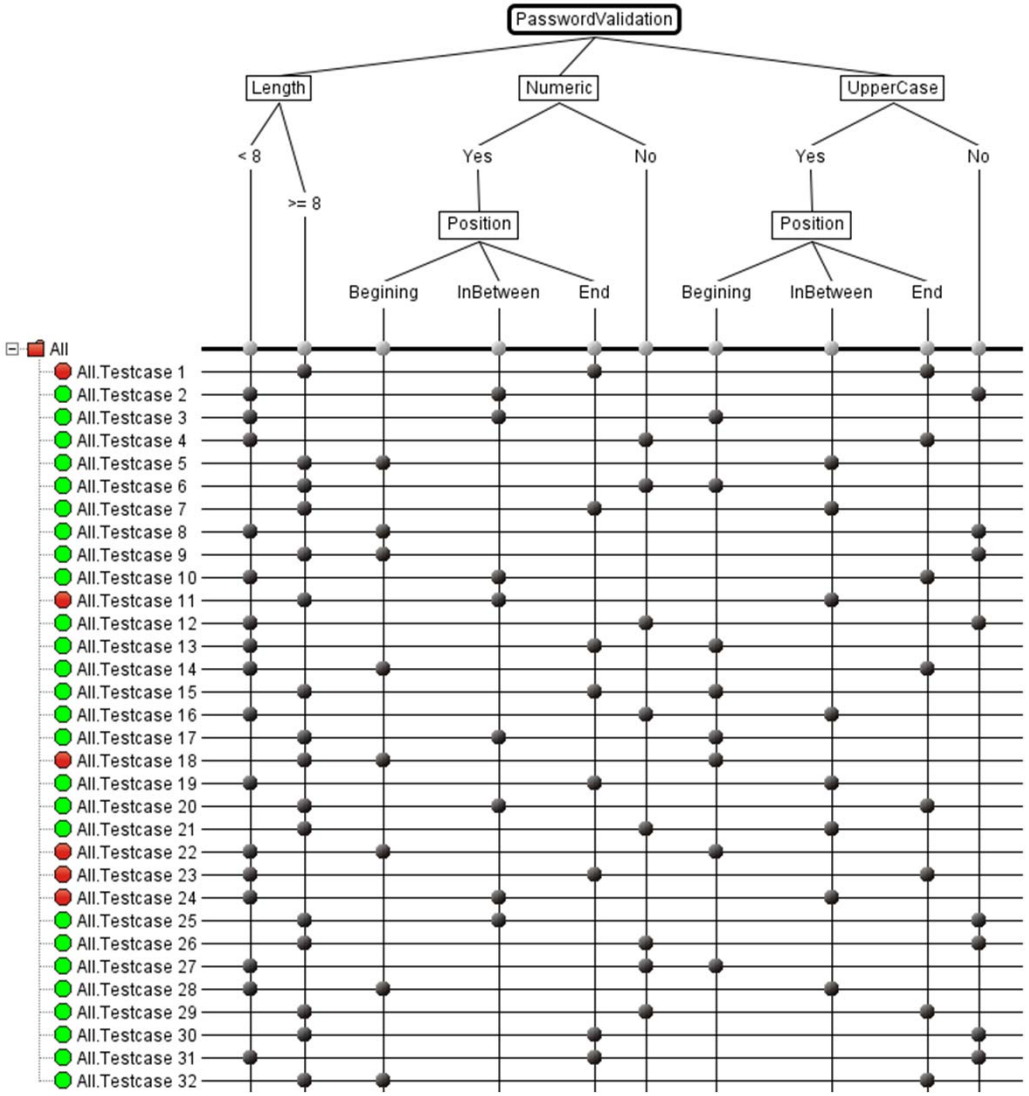
Test specifications vs. test cases

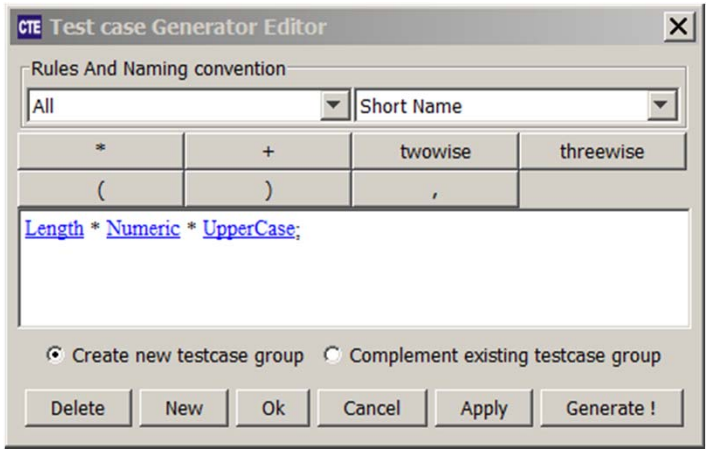
- Test specification for the *count* function:
  - Array
    - Size greater than one
    - Number of occurrences of counted string greater than one
    - Unsorted
    - Content alphanumeric
  - Count what
    - String of more than one character
    - Same data type as array
- 2 test cases satisfying the same specification
  - (“world”, “mother”, “string”, “string”, “country”, “Sunday”); “string”; count = 2
  - (“1st” , “bike”, “3rd”, “bike”, “5th”, “6th”, “7th”, “bike”, “9th”); “bike”; count = 3

Two situations

- Case 1. When an input's or environment condition's only test relevant aspect are its values, its leaf classes could be made to correspond to actual test values and directly used to generate test cases
- Case 2. When the actual test value is the result of the intersection of two or more aspects the classification tree method will produce specifications that the tester will use in developing actual test cases

Case 2: A valid password is the result of 3 intersecting separate aspects: Length, Numeric & Upper case character





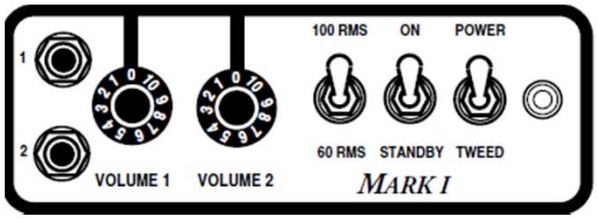
Test generator expression describing the Cartesian product of the 3 aspects: Length, Numeric and Upper Case. Invalid test specification are the result of the ruless applied through the Dependency Editor (not shown)

The output from the tool is a specification the tester must transform into actual test values before they can be used to test the application

Subject	Description	TestName	Step Name	Specification	Expected Result	Test Case
All		All.Testcase 2	1.2	- Length: < 8 - Numeric: Yes - Position: InBetween - UpperCase: No	Invalid	ric3k
All		All.Testcase 3	1.3	- Length: < 8 - Numeric: Yes - Position: InBetween - UpperCase: Yes - Position: Begining	Invalid	Rag2u
All		All.Testcase 4	1.4	- Length: < 8 - Numeric: No - UpperCase: Yes - Position: End	Invalid	marY
All		All.Testcase 5	1.5	- Length: >= 8 - Numeric: Yes - Position: Begining - UpperCase: Yes - Position: InBetween	Valid	2Eduardo
All		All.Testcase 6	1.6	- Length: >= 8 - Numeric: No - UpperCase: Yes - Position: Begining	Invalid	Jonathan
All		All.Testcase 7	1.7	- Length: >= 8		

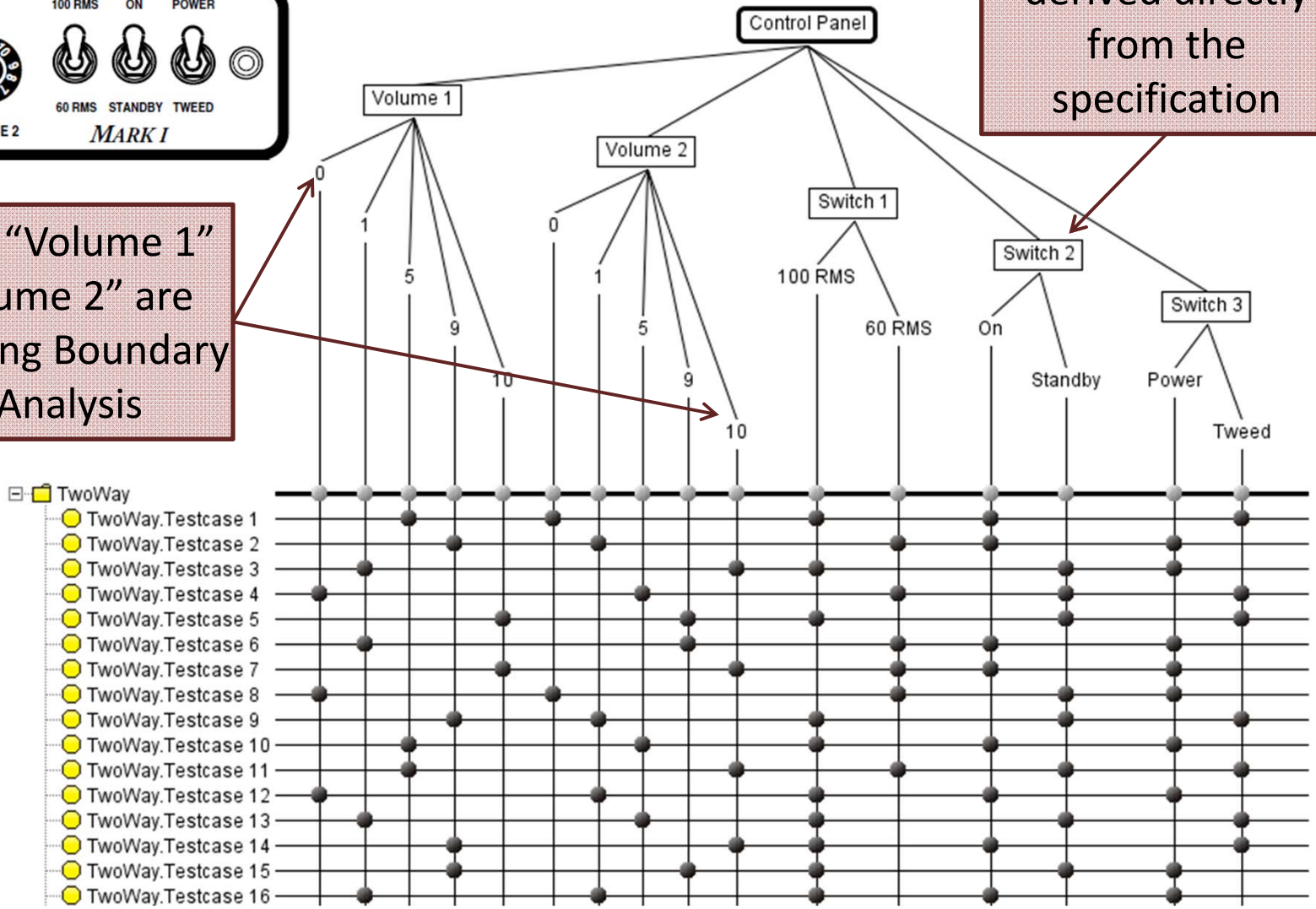
Based on the specification, the tester must produce the actual value to be used in the test case. Automating the encoding process is not trivial in most cases. The expected result must also be specified by the tester

Case 2: The classes in the classification tree correspond to actual test values



Values for the switches are derived directly from the specification

Values for "Volume 1" and "Volume 2" are derived using Boundary Value Analysis



In this case the tool's output are actual test cases that could be fed directly to the application or implemented through Junit or any other test harness

	Switch 1	Switch 2	Switch 3	Volume 1	Volume 2
TwoWay					
Testcase 1	100 RMS	On	Tweed	5	0
Testcase 2	60 RMS	On	Power	9	1
Testcase 3	100 RMS	Standby	Power	1	10
Testcase 4	60 RMS	Standby	Tweed	0	5
Testcase 5	100 RMS	Standby	Tweed	10	9
Testcase 6	60 RMS	On	Power	1	9
Testcase 7	60 RMS	On	Power	10	10
Testcase 8	60 RMS	Standby	Power	0	0
Testcase 9	100 RMS	Standby	Tweed	9	1
Testcase 10	100 RMS	On	Power	5	5
Testcase 11	60 RMS	Standby	Tweed	5	10
Testcase 12	100 RMS	On	Power	0	1
Testcase 13	100 RMS	Standby	Tweed	1	5
Testcase 14	100 RMS	On	Tweed	9	10
Testcase 15	100 RMS	Standby	Power	9	9
Testcase 16	100 RMS	On	Power	1	1

Pairwise test cases generated using CTE XL

Metrics

- Minimum number of test specifications (rows in the combination table) not test cases
  - This criterion requires every single class in the tree to be selected in at least one test case
  - The number of test specifications is equal to the largest number of leaf classes with a common classification at the highest level
  - The number does not take in consideration potential impossible combinations that would require extra combinations to keep the required coverage
- More realistic estimate. The largest of:
  - The product of the number of leaf classes up to the highest composition level
  - The sum of all leaf classes
- Maximum number of test cases (All combinations as per tree construction)
  - The maximum criterion requires every possible class combination to be selected in at least one test case or test specification. It corresponds to the Cartesian product of all the leaf classes
  - Its exponential growth makes it impracticable for most situations
- Combinatorial criterion
  - This criterion corresponds to the *t-wise* combination of the number of classes
  - An approximation to the total number of classes is given by the formula $\left\lceil \left(\prod_{Largest(t,k_1,k_2,..,k_l)} k_i \right)^t \ln l \right\rceil$ where $l \geq t$ is the number of lowest level aspects k_1, k_2, k_l, the number of classes belonging to them and t the strength of the interactions (pair-wise, three wise, n-wise)

Other capabilities of the method not covered here

- Test sequences
- Values transitions
- Timing

Questions?

Classification tree method

- Systematic technique for generating adequate test specifications from explicit and tacit knowledge about a test object
  - Identifying and documenting all test object's test-relevant aspects
  - Partitioning the values that characterize each test relevant aspect into disjoint subsets called classes which will serve as a basis for generating test specifications or test cases
  - Generating test specifications or test cases for the test object
- Retrospective
  - The Category-Partition Method for Specifying and Generating Functional Tests, T. Ostrand & M. Balcer, 1988
  - Classification Trees for Partition Testing, M. Grochtmann & K. Grimm, 1993,
  - Test Case Design Using Classification Trees, M. Grochtmann, 1994
- Tools
  - CTE XL
  - CTE XL (Professional)
  - Tessy
- Current usage
  - Mainly automotive industry, where it originated at Daimler-Benz
  - Telecommunications. Interfaces with TTCN-3

Inputs, environment and test relevant aspects

- Inputs, aka parameters. Explicit inputs to a test object, supplied either by the user or by another program
- Environment. Relevant conditions present at execution time
- Aspects
  - An aspect is any characteristic of an input or environment condition that the test designer knows or suspects might influence the behavior of the test object. Aspects include:
    - Values – this is how most information is conveyed. However beware of single values encoding different type of information corresponding to different test-relevant aspect
    - Multiplicity – a given value might be absent, exist once, exist more than once, exist a minimum number of times, exist a maximum number of times
    - Sizes, lengths and sequences – the first, the second and the last elements; the minimum and maximum number of occurrences, the minimum and maximum lengths

Guidelines for selecting test objects

- The choice of test object largely determines what aspects are relevant and which ones are not
- The function being tested can be invoked by methods consistent with the level of testing
- Its output can be observed by methods appropriate to the level of testing
- After producing the output the function “rests” until its next invocation

Guidelines for selecting test-relevant aspects

- The aspect's purpose is to document and guide the analysis of the input domain by decomposing it into independent factors. Aspects are represented by compositions and classifications
- There is no unique criteria for choosing them, it is not a mechanical task. Aspects sources include
  - Specifications, if they exist, identifying how the functional unit ought to behave with respect to some characteristic of the parameter or environment condition
  - Common knowledge about what things may or ought to be treated differently by an implementation
  - Test catalogues

Guidelines for identifying equivalence classes

- The range of values a test-relevant aspect can take is partitioned into equivalence classes
  - If necessary these classes can be further subdivided
  - If necessary introduce a class corresponding to invalid values (robust testing)
- Each class corresponds to a subset of values of the aspect that should elicit an equivalent behavior from the test object
- Under certain circumstances equivalence classes can be refined into actual test values to support the automatic generation of test cases

Introducing invalid values for robust testing

- Robust or negative testing is used to test error or exception handling mechanisms
- Invalid values need to be considered when dealing with data from external sources, when creating reusable code and, depending on the quality requirements of the application, even when dealing with data from internal sources (defensive programming)