Haskell Combinatory Library Manual

João Amorim

1. Modules

The library consists of two basic modules: Types and Combinatory. In the first we find the type definitions that are used: inputs (Categories, Partitions, Classes), results (Suites, Combinations) and rules (Operators, Logic Operators, Rule Trees). In the second one we find the definition of the important functions (minimal combination, full combination, ruled combinations, semantic), as well as all the necessary auxiliar functions.

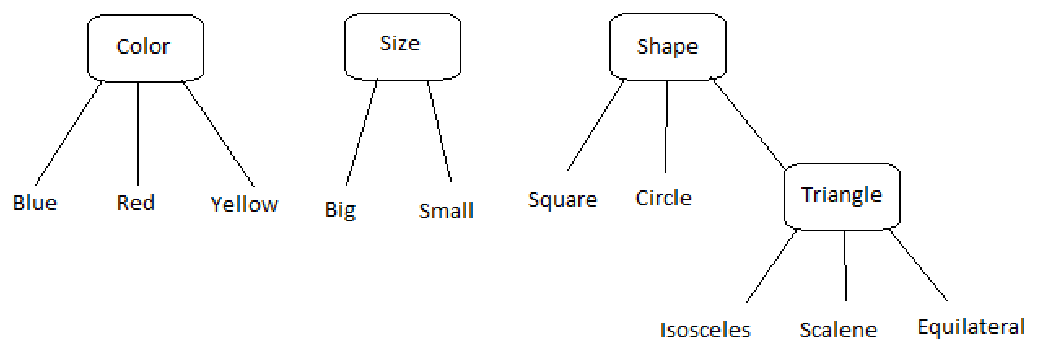
2. Type definition

2.1 Inputs

When constructing an instance of a classification tree, these are the types that are used. First, some definitions about the types:

- Categories: a category is one domain of the classification tree. In a valid combination, each category will have a value (class) chosen.
- Classes: a class is a leaf in the classification tree. It represents one of the values that its domain (category) can get.
- Partition: a partition is a subdomain of its category. It can either be a class, or a set of other partitions.

In the picture we can see an example of a classification tree, and below the code to represent it in the library.



```
color :: Category
color =  Cat "Color" [Leaf "Red", Leaf "Blue",Leaf "Yellow"]
shape :: Category
shape = Cat "Shape" [Leaf "Circle",Leaf "Square",triangle]
triangle :: Partition
triangle = Node "Triangle" [Leaf "Isosceles", Leaf "Equilateral", Leaf "Scalene"]
size :: Category
size = Cat "Size" [Leaf "Big",Leaf "Small"]
inp :: Input
inp = [color,shape,size]
```

2.2 Results

After evaluating the requested combinations, the library represents the results in the following types:

```
data Result = R String Class
type Combination = [Result]
type Suite = [Combination]
```

Note that while we could just represent a combination as a set of classes, it is very useful to know what category those classes belong to. Therefore, each combination is a set of pairs (results) of classes and the name of its category.

2.3 Rules

In this section we will just see the type definition for the possible rules. In a later section we will the valid inputs and the semantic outputs of these rules.

A rule is represented as a tree of rules, where in each node we have an Operator, and its branches are other rules. In the leafs we can have logic prepositions or "selecting" prepositions (partition selection/excluding, category selection).

A rule will be something like this:

Rule = Rule 'Operator' Rule | Logic Preposition | Select/Exclude Partition | Select Category.

The code for this definition is:

```
data Operator = Plus | Times | ST | STN | Union | Join | Intersect
data LogicOp = And | Or | Xor | Nor | Equi | Impl
data Rule = Rop Operator Rule Rule | RLog LogicOp [String] [String] | P [String] | NotP [String] |
Ctg String
```

3. Basic Combinations

The tool offers two types of basic combinations (combinations that don't take into account rules): minimal combination and full combination.

3.1. Minimal Combination

When we apply the funcition *minimalComb* to a valid input, the result will be the smallest set of combinations where each class is selected. If we apply it to the example shown in 2.1., the result is:

```
Circle, Big, Red
Square, Small, Blue
Isosceles, Big, Yellow
Equilateral, Small, Red
Scalene, Big, Blue
```

3.2. Full Combination

The other type of basic combination is the full combination, where every possible combination of classes is calculated. It can be calculated through the function *fullComb*. Beware that for big classification tree it can take a long time, or even explode. In the case of the example, as we can only have 30 different combinations, the tool can easily find them:

```
Red, Circle, Big
Red, Circle, Small
Red, Square, Big
Red, Square, Small
Red, Isosceles, Big
Red, Isosceles, Small
Red, Equilateral, Big
Red, Equilateral, Small
Red, Scalene, Big
Red, Scalene, Small
Blue, Circle, Big
Blue, Circle, Small
Blue, Square, Big
Blue, Square, Small
Blue, Isosceles, Big
Blue, Isosceles, Small
Blue, Equilateral, Big
Blue, Equilateral, Small
Blue, Scalene, Big
Blue, Scalene, Small
```

4. Ruled Combinations

4.1. Suite calculation

The process of finding the set of combinations given an input and a rule tree, is divided in two steps.

The first step, made by the function *semantic*, calculates the rule and input, and returns a set of combinations that respect that rule, but only with classes of the categories included in the rule.

The second step, that adds the remaining categories to the result, can be done in two different ways: the first one, through *minComplete*, creates the minimal number of combinations possible, such that each class of the remaining categories appears at least once; the second one, through *fullComplete*, returns all the possible combinations of the remaining categories and the semantic output.

The functions *minRuledComb* and *fullRuledComb*, when called, run these two steps, and return a valid suite. Below we have an example to show the difference between *semantic*, *minRuledComb* and *fullRuledComb*.

```
semantic (P [“Triangle”]) input

Isosceles
Equilateral
Scalene

minRuledComb (P [“Triangle”]) input

Isosceles, Red, Big
Equilateral, Blue, Small
Scalene, Yellow, Big


fullRuledComb (P [“Triangle”]) input

Isosceles, Red, Big
Isosceles, Red, Small
Isosceles, Blue, Big
Isosceles, Blue, Small
Isosceles, Yellow, Big
Isosceles, Yellow, Small
Equilateral, Red, Big
Equilateral, Red, Small
Equilateral, Blue, Big
Equilateral, Blue, Small
Equilateral, Yellow, Big
Equilateral, Yellow, Small
Scalene, Red, Big
Scalene, Red, Small
Scalene, Blue, Big
Scalene, Blue, Small
Scalene, Yellow, Big
Scalene, Yellow, Small
```

4.2. Rules input and semantic output

4.2.1. P [Partitions], NotP [Partitions]

Valid input: a list of partitions from the same category.
Examples: P [“Blue”,”Red”], NotP [“Triangle”,”Square”].

Semantic output: a list of classes from the category of the partitions.
Examples: semantic (P [“Blue”]) = [[(R “Color” “Blue”)]], semantic (NotP [“Triangle”]) = [[(R “Shape” “Square”)],[(R “Shape” “Circle”)]]

4.2.2. Ctg Category

Valid input: the name of one category.
Example: Ctg “Color”

Semantic output: the list of all classes in that category.

4.2.3. Logic Expressions (RLog LogicOp [Partitions] [Partitions])

Valid input: two sets of partitions from different categories.
Example: RLog Or [“Blue”,”Red”] [“Triangle”]

Semantic output: combinations between the two categories that satisfy the logic rule.
Example semantic (RLog Nor [“Blue”,”Red”] [“Triangle”,”Square”]) = [[(R “Color” “Yellow”),(R “Shape” “Circle”)]]

*4.2.4. Rop Plus/Times Rule Rule (Rule++Rule/Rule**Rule)*

Valid input: two rules whose semantic output have no category in common.
Examples: Rop Times (P [“Blue”]) (P[“Triangle”]) is valid; Rop Plus (P [“Big”]) (P [“Small”]) is not valid.

Semantic output: the minimal (Plus) / full (Times) combination of the semantic of the two rules.

4.2.5. Rop Union Rule Rule (Rule or Rule)

Valid input: two rules whose semantic output have the exact same categories.
Examples: Rop Union (P [“Blue”]) (P [“Red”]) is valid; Rop Union (P [“Big”]) (P [“Blue”]) is not valid.

Semantic output: a set of combinations composed by the union of the sets resulting from the semantic output of the two rules, excluding repeated combinations.

4.2.6. Rop Intersect Rule Rule (Rule and Rule)

Valid input: see Union.

Semantic output: a set of combinations composed by the combinations present in both the two semantic outputs.

4.2.7 Rop Join Rule Rule (Rule $\bowtie$ Rule)

Valid input: two rules semantic output have at least one category in common.
Example: Rop Join (RLog Or [“Blue”] [“Big”]) (RLog And [“Big”] [“Triangle”])

Semantic output: a set of combinations resulting from the composition of the combinations from both outputs that are equal on their common categories. Works like the natural join operator from relational algebra.

4.2.8. Rop ST/STN Rule Rule (Rule such that (not) Rule)

Valid input: two rules where the set of categories in the semantic output of the second one is a subset of the set of categories in the semantic output of the first one.

Semantic output: the set of combinations on the semantic output of the first rule that cover/don’t cover combinations from the set of combinations of the second semantic output.