# The Coq proof assistant : principles and practice

J.-F. Monin

Université Grenoble Alpes

2016

Lecture 2

# Outline

# Outline

# Definition by cases on an enumerated type



## Question

Give a `color` for each possible value in `rgb`

$$\frac{}{\texttt{rgb}}\ \texttt{Rf} \qquad \frac{}{\texttt{rgb}}\ \texttt{Gf} \qquad \frac{}{\texttt{rgb}}\ \texttt{Bf}$$

## Example

`Rf` maps to `Red`, `Gf` maps to `Green`, `Bf` maps to `Blue`

$$\frac{\dfrac{}{\texttt{rgb}}\ \downarrow r \qquad \dfrac{}{\texttt{color}}\ \texttt{Red} \qquad \dfrac{}{\texttt{color}}\ \texttt{Green} \qquad \dfrac{}{\texttt{color}}\ \texttt{Blue}}{\texttt{color}}\ \texttt{case}$$

# Definition by cases on an enumerated type



## Question

Give a `rgb` for each possible value in `rgb`

$$\frac{}{\texttt{rgb}}\ \texttt{Rf} \qquad \frac{}{\texttt{rgb}}\ \texttt{Gf} \qquad \frac{}{\texttt{rgb}}\ \texttt{Bf}$$

## Example

`Rf` maps to `Bf`, `Gf` maps to `Gf`, `Bf` maps to `Rf`

$$\frac{\dfrac{}{\texttt{rgb}}\ \downarrow r \qquad \dfrac{}{\texttt{rgb}}\ \texttt{Bf} \qquad \dfrac{}{\texttt{rgb}}\ \texttt{Gf} \qquad \dfrac{}{\texttt{rgb}}\ \texttt{Rf}}{\texttt{rgb}}\ \texttt{case}$$

# Building block

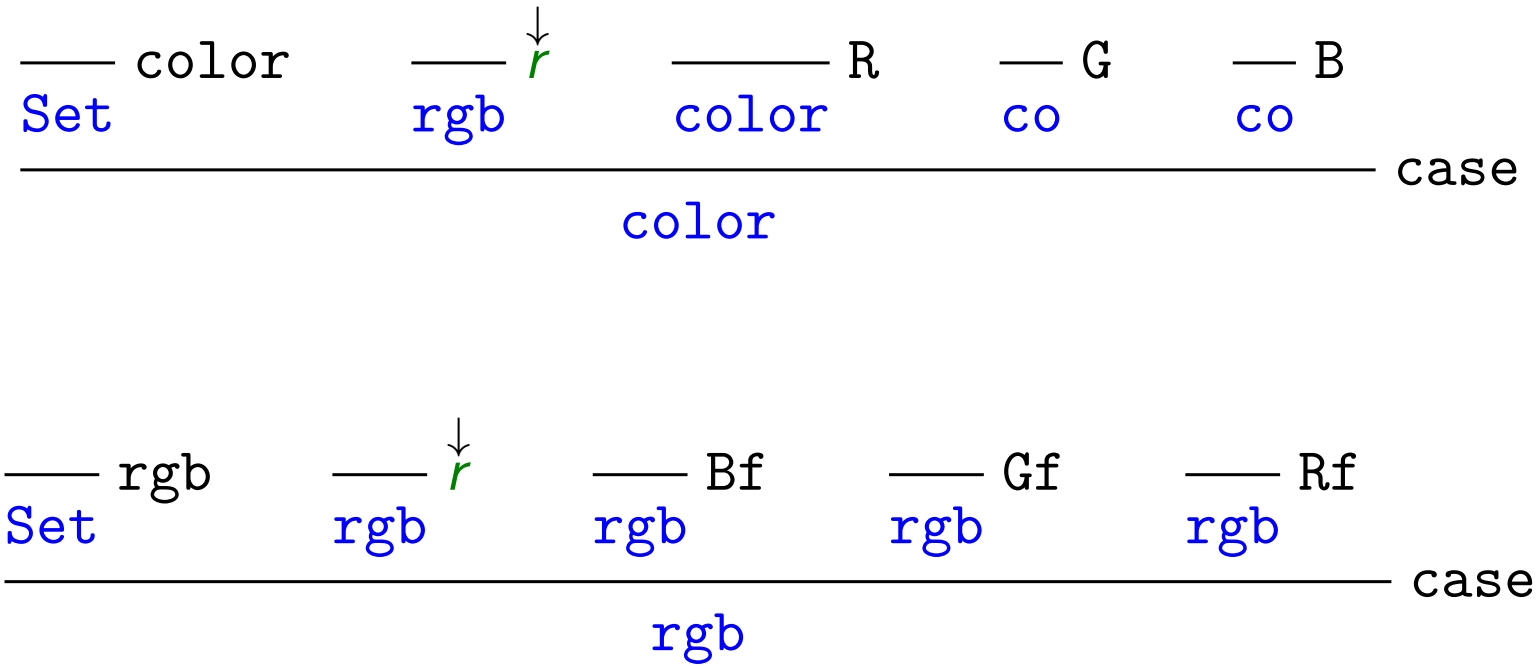


In this presentation, the order of contructors matters:
`Rf`, `Gf`, `Bf`
The `case` construct is driven by 2 parameters

- the type of the value to be analyzed
  each enumerated type (e.g. `rgb`) comes automatically with its `case` construct, which should actually be written, e.g. $\texttt{case}_{\texttt{rgb}}$
- the type of the result

$$\frac{\underset{\texttt{Set}}{\overset{\downarrow}{A}} \quad \underset{\texttt{rgb}}{\overset{\downarrow}{r}} \quad \underset{\texttt{A}}{\overset{\downarrow}{x_1}} \quad \underset{\texttt{A}}{\overset{\downarrow}{x_2}} \quad \underset{\texttt{A}}{\overset{\downarrow}{x_3}}}{\underset{\texttt{A}}{\qquad}}\ \texttt{case}$$

# Correct version of previous examples

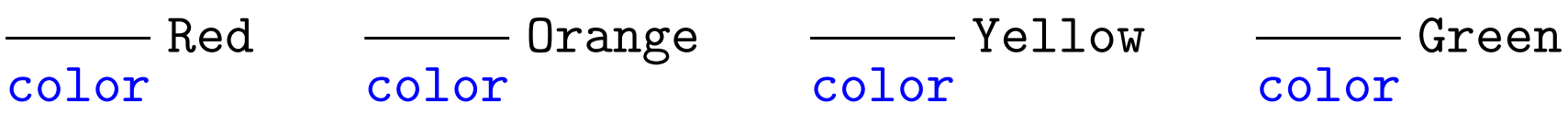


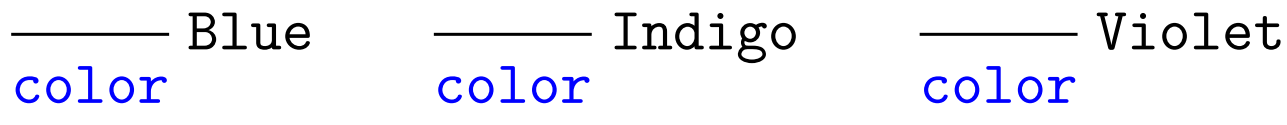
# Definition by cases on a general inductive type

To be introduced below...

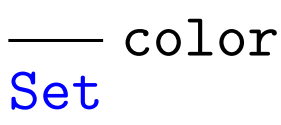
# Outline

# From graphical presentation to Coq syntax



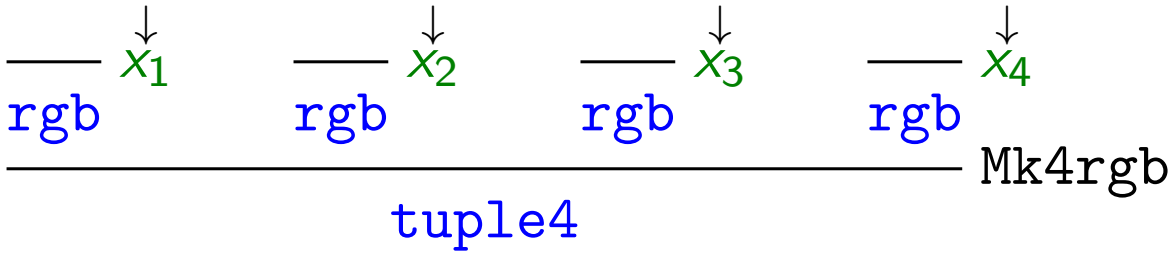

And simultaneously




```
Inductive color: Set :=
  | Red : color     | Orange : color   | Yellow : color
  | Green : color   | Blue : color     | Indigo : color
  | Violet : color
.
```

# Coq syntax of tuple4

## Making a 4-tuple of rgb

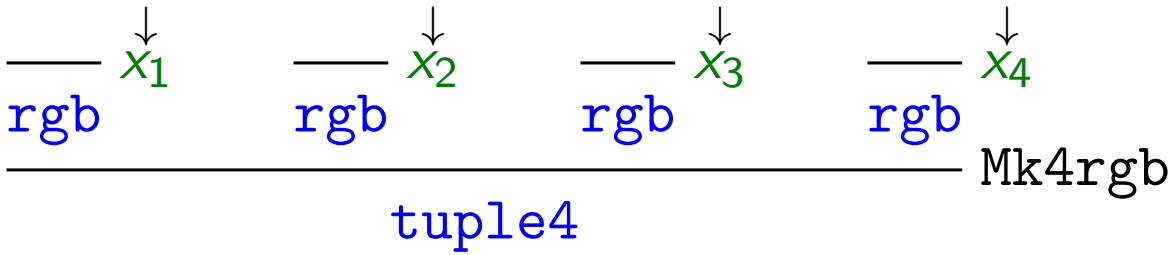



```
Inductive tuple4 : Set :=
  | Mk4rgb :
      forall x1: rgb, forall x2: rgb,
      forall x3: rgb, forall x4: rgb, tuple4
```

# Coq syntax of tuple4, shorthand

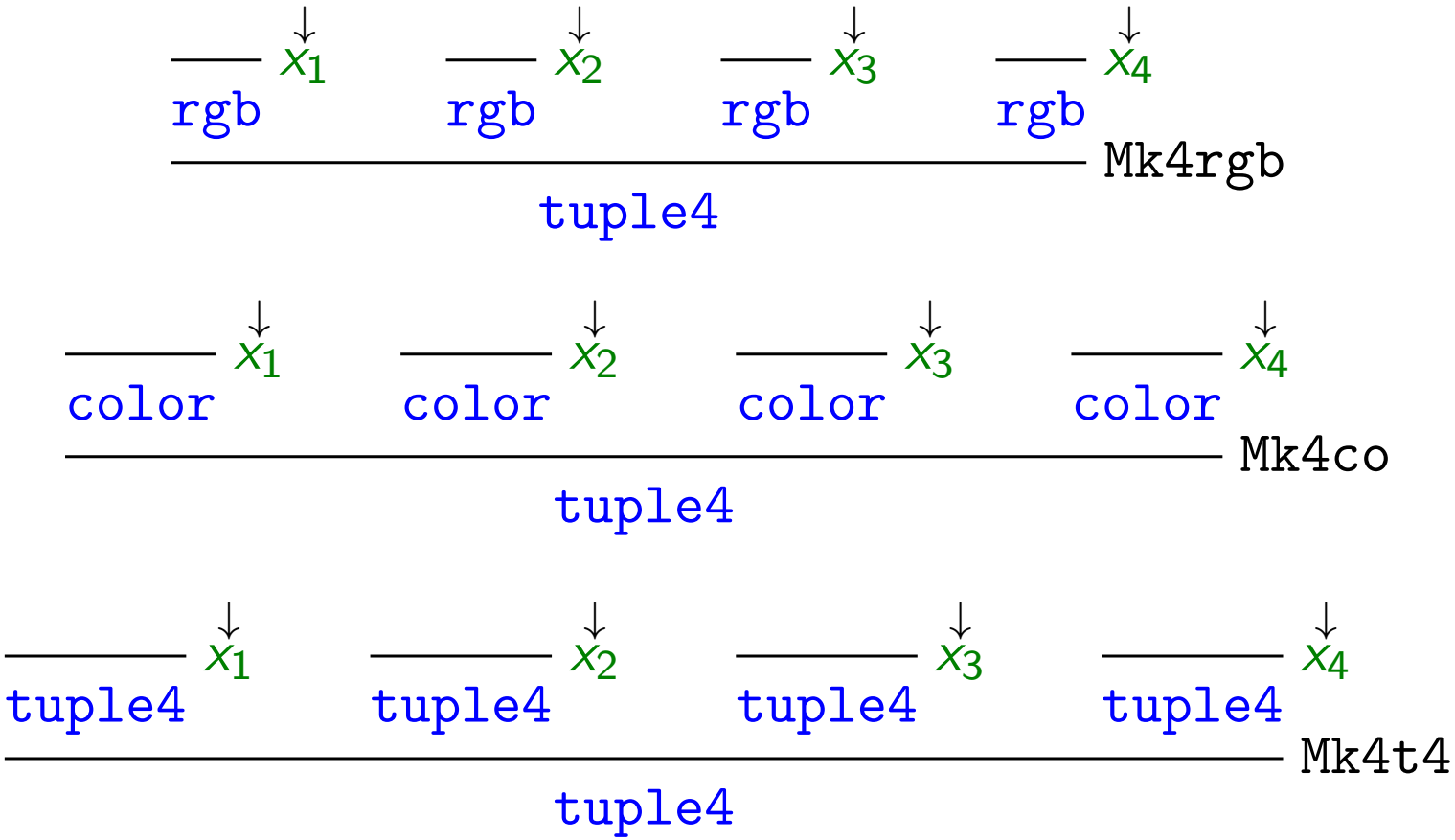


## Making a 4-tuple of rgb




```
Inductive tuple4 : Set :=
  | Mk4rgb : forall x1 x2 x3 x4: rgb, tuple4.
```

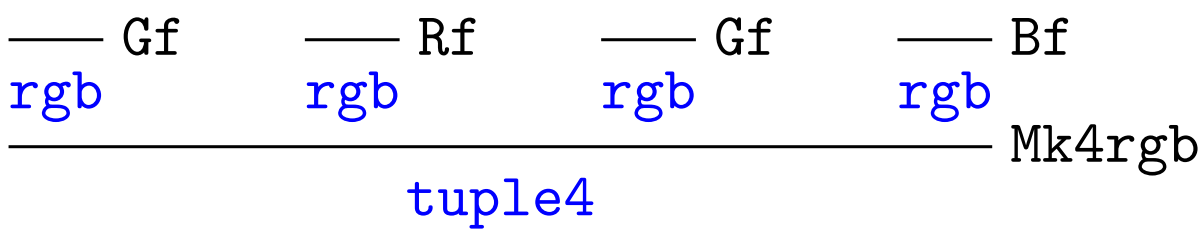
# More general 4-tuples: several constructors




```
Inductive tuple4 : Set :=
  | Mk4rgb : forall x1 x2 x3 x4: rgb, tuple4
  | Mk4color : forall x1 x2 x3 x4: color, tuple4
  | Mk4t4 : forall x1 x2 x3 x4: tuple4, tuple4
.
```

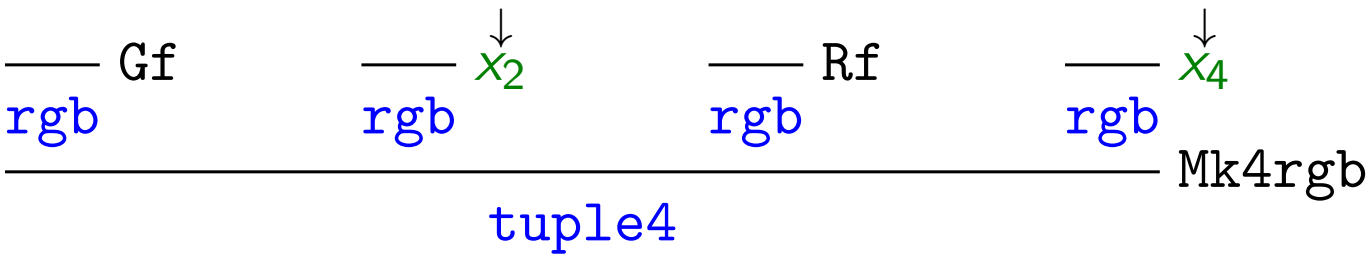
# Pluging trees: the interactive way

## A concrete 4-tuple of rgb




```
Definition t1: tuple4.
apply Mk4rgb.
  apply Gf.
  apply Rf.
  apply Gf.
  apply Bf.
Defined.
```

# Open trees




## In Coq

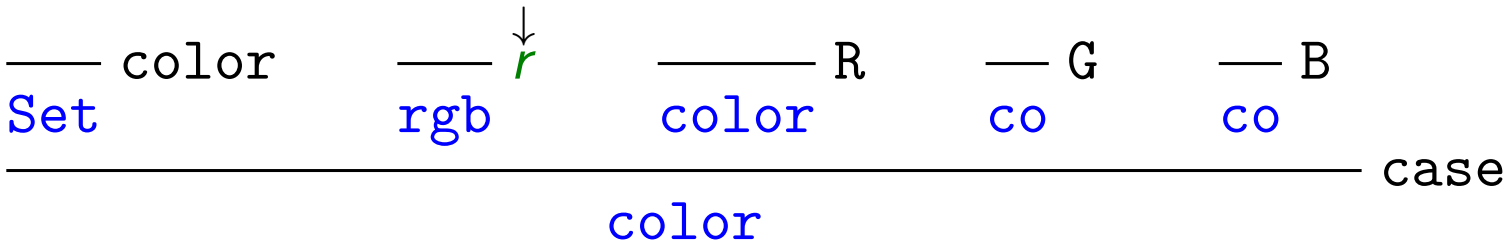
```
Section a_tuple_with_variable.

Variable x2: rgb.
Variable x4: rgb.

Definition t4 etc.

End a_tuple_with_variable.
```

# Exercises

Write trees for examples of 4-tuples of 4-tuples using `tuple4`.
Some of them, closed, some of them open
E.g. $\langle\langle \mathtt{R}, \mathtt{Y}, \mathtt{B}, \mathtt{B}\rangle, \langle \mathtt{B}, \mathtt{O}, x_4, \mathtt{R}\rangle, \langle x_7, x_7, x_7, \mathtt{V}\rangle, \langle \mathtt{V}, \mathtt{Y}, \mathtt{O}, \mathtt{R}\rangle\rangle$

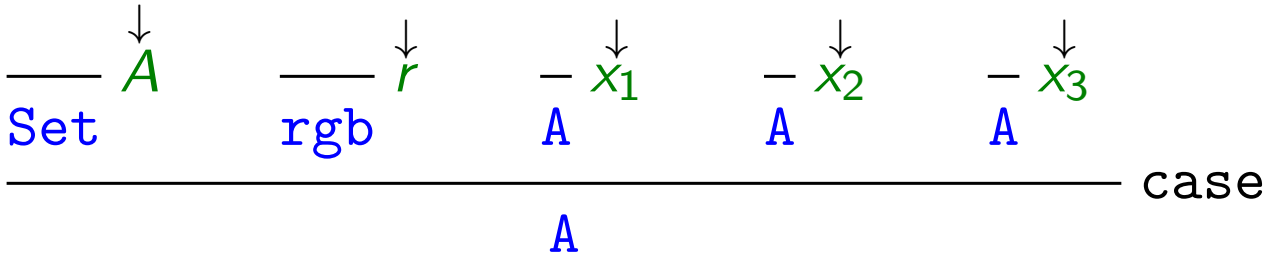
# Definition by cases on an enumerated type

Rf maps to Red, Gf maps to Green, Bf maps to Blue




```
Definition color_of_r: color.
destruct r.
  apply Red.
  apply Green.
  apply Blue.
Defined.
```

# General case




```
Definition A_of_r: A.
destruct r.
  apply x1.
  apply x2.
  apply x3.
Defined.
```

# Direct definition

```
Definition color_of_r : color :=
  match r with
  | Rf => Red
  | Gf => Green
  | Bf => Blue
  end.
```

# Case analysis on a tree

See interactive session

# Outline

# Case analysis on a constant

```
Definition color_of_Bf : color :=
  match Bf with
  | Rf => Red
  | Gf => Green
  | Bf => Blue
  end.
```

# Reduction

```
match Bf with
| Rf => Red
| Gf => Green
| Bf => Blue
end.
```

Reduces to

```
Blue
```

# Outline

# Functions

```
Definition color_of : forall (r: rgb), color :=
  fun (r: rgb) =>
  match r with
  | Rf => Red
  | Gf => Green
  | Bf => Blue
  end.
```

Application: by juxtaposition without parenthesis

```
color_of Bf
```

Parentheses can be used for grouping

# More functions

```
Definition Set_of : forall (r: rgb), Set :=
  fun (r: rgb) =>
  match r with
  | Rf => rgb
  | Gf => color
  | Bf => tuple4
  end.

Definition funny : forall (r: rgb), Set_of r :=
  fun (r: rgb) =>
  match r with
  | Rf => Bf
  | Gf => Green
  | Bf => t1
  end.
```

# Interactive definition of a function





## Use intro

```
Definition interactive_color_of :
   forall (r: rgb), color.
intro r.
destruct r.
  apply Bf.
  apply Green.
  apply t1.
Defined.
```

# Outline

# Two kinds of trees: first kind

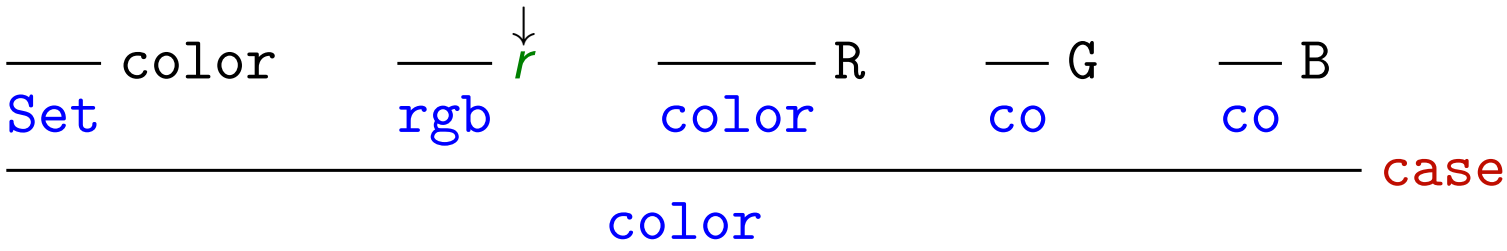
$$t1 := \left\{ \begin{array}{cccc} \frac{\text{Gf}}{\texttt{rgb}} & \frac{\text{Rf}}{\texttt{rgb}} & \frac{\downarrow r}{\texttt{rgb}} & \frac{\text{Bf}}{\texttt{rgb}} \\ \multicolumn{4}{c}{\frac{\qquad\qquad\qquad\qquad\qquad\qquad}{\texttt{tuple4}}\ \texttt{Mk4rgb}} \end{array} \right.$$

t2:= ..., t3 := ... t4 := ...
(similar to t1, using constructors Mk4co, Mk4rgb, Mk4rt4 and variables only)

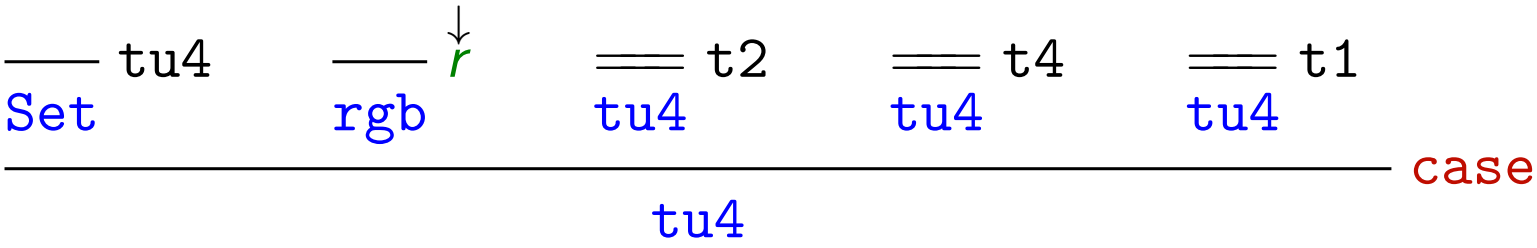
$$\left\{ \begin{array}{cccc} \frac{\downarrow r}{\texttt{rgb}} & & & \\ \frac{}{\texttt{tuple4}}\ \texttt{t1} & \frac{}{\texttt{tuple4}}\ \texttt{t2} & \frac{}{\texttt{tuple4}}\ \texttt{t3} & \frac{}{\texttt{tuple4}}\ \texttt{t4} \\ \multicolumn{4}{c}{\frac{\qquad\qquad\qquad\qquad\qquad\qquad}{\texttt{tuple4}}\ \texttt{Mk4t4}} \end{array} \right.$$

**Similar to the usual data structures in programming**

# Two kinds of trees: second kind (with case)




*With* co := color




*With* tu4 := tuple4

**Here, case looks strange: the usual intuition associates it to control, not to data**

# Explanation

## Inside Coq

The internal representation *all* trees is really what you expect. Rules labelled with `case` are implemented by a node pointing to all branches representing the subtrees on top of the corresponding line (5 of them in the previous examples).

To some extent, `case` can be seen as a primitive (and very flexible) constructor.

## Evolution

The intuitive idea of control behind `case` can be understood as the fate of the corresponding node: when a constant, e.g., `Bf` will be plugged to the key argument (`r:rgb` in our examples), then this part of the tree will be reduced to the corresponding subtree (here: the rightmost, i.e., respectively `B` and `t1` on the 2 previous examples).

# Reduction

Things will be come clear after the introduction of the notion of reduction in lecture 03.

Type Theory (the mathematical foundation of Coq) relies on 3 tightly coupled notions, which only make sense when they are together:

- constructors of an inductive type
- case analysis on an inductive type
- reduction

# Last remark

The idea of plugging a tree making a given type into an input (having the same type) of another tree is completely uniform.
Hence, a case can be embedded in a tree.

$$c_1 \stackrel{\text{def}}{=} \left\{ \begin{array}{c} \text{co} \quad \overset{\downarrow}{r} \quad \text{R} \quad \text{G} \quad \text{B} \\ \text{Set} \quad \text{rgb} \quad \text{co} \quad \text{co} \quad \text{co} \\ \hline \text{color} \end{array} \right. \text{case}$$

$$\begin{array}{c} \text{G} \quad \overset{\downarrow}{r} \quad c_1 \quad \text{B} \quad \text{R} \\ \text{color} \quad \text{rgb} \quad \text{color} \quad \text{color} \quad \text{color} \\ \hline \text{tuple4} \end{array} \text{Mk4co}$$