Summary

This standard extends `TZIP-004 (A1)` by defining a right-hand balance tree structure for `or` and `pair` types. Structure of comb defined in TZIP-004 is the most obvious one, but the worst-case performance of operations on it scales linearly with number of elements in the comb which is suboptimal. Whereas tree structure allows reaching better average performance of access and update operations.

Primary purpose of this standart is to guide implementation of high-level languages over Michelson, albeit any contract can benefit, in terms of gas costs, from using it.

Definition of a Right-hand balanced Tree

A right-balanced tree can be defined recursively as follows:

- The left subtree has height equal to, or one less than the right subtree
- The right and left subtrees are balanced
- For clarity, all terminal leaves of the tree are considered balanced

For example, the following is a right-balanced

```
(or (or a b) (or c (or d e)))

       or
     /    \
    or     or
   / \    / \
  a   b  c   or
            / \
           d   e
```

If we insert a new node `f` into the above tree:

```
(or (or a b) (or (c d) (or e f)))

       or
     /    \
    or     or
   / \    /  \
  a   b  or   or
        / \  / \
       c  d  e  f
```

ADT Syntax sugar

The `Pair`s and `Or`s syntax sugar is extended to work for tuples and unions of arbitrary size:

```
(a_1, a_2, ..., a_n)    ~> mkRightBalancedTree pair '[a_1, a_2, ..., a_n]
(a_1 | a_2 | ... | a_n) ~> mkRightBalancedTree or   '[a_1, a_2, ..., a_n]
```

where `mkRightBalancedTree` is a type-level fuction from a binary type constructor and heterogenous list of types to a right balanced tree as defined above.

CASE macro

The `CASE` macro is also modified such that it selects the correct branch of a balanced entrypoint `or` tree.