This module contains operations for working with radix trees. A radix tree is a data structure for efficient storage and lookup of values that often share prefixes, typically used with strings.

A common question when I show this to people is: how do I add to the tree? delete? update? For these, grab the *Range* of the tree, use set logic to add/remove any elements, and construct a new tree with *RadixTree*.

For educational purposes, I've heavily commented all the operations. I recommend using the constant expression evaluator to try the building blocks to learn how things work if you're confused. That's how I learned.

MODULE *RadixTrees*

LOCAL INSTANCE *FiniteSets*
LOCAL INSTANCE *Sequences*
LOCAL INSTANCE *Integers*

Helpers that aren't Radix-tree specific.

shortestSeq returns the shortest sequence in a set.

LOCAL $shortestSeq(set) \triangleq$
  CHOOSE $seq \in set$:
    $\forall\, other \in set$:
      $Len(seq) \leq Len(other)$

Filter the sequence to only return the values that start with c.

LOCAL $filterPrefix(set, c) \triangleq \{seq \in set : seq[1] = c\}$

Strip the prefix from each element in set. This assumes that each element in set already starts with prefix. Empty values will not be returned.

LOCAL $stripPrefix(set, prefix) \triangleq$
  $\{SubSeq(seq, Len(prefix) + 1, Len(seq)) : seq \in set \setminus \{prefix\}\}$

Returns the set of first characters of a set of char sequences.

LOCAL $firstChars(set) \triangleq \{seq[1] : seq \in set\}$

Find the longest shared prefix of a set of sequences. Sequences can be different lengths, but must have comparable types.
i.e. $longestPrefix(\{1, 2\}, \{1, 2, 3\}, \{1, 2, 5\}) \triangleq \{1, 2\}$

LOCAL $longestPrefix(set) \triangleq$
  LET
    $seq \triangleq shortestSeq(set)$
      shortest sequence is the longest possible prefix
    $lengths \triangleq \{0\} \cup \{$
      $i \in 1 \,..\, Len(seq)$:
        $\forall\, s1, s2 \in set : SubSeq(s1, 1, i) = SubSeq(s2, 1, i)\}$
      all the lengths we match all others in the set
    $maxLength \triangleq$

```
    CHOOSE x ∈ lengths :
      ∀ y ∈ lengths :
        x ≥ y
      choose the largest length that matched
  IN  SubSeq(seq, 1, maxLength)
```

```
Radix tree helpers

RECURSIVE range(_, _)
LOCAL range(T, prefix) ≜
  LET
    current ≜ IF Len(T.Value) > 0 THEN {T.Value} ELSE {}
      current value of node (if exists)

    children ≜ UNION {
      range(T.Edges[edge], prefix ∘ T.Prefix) :
      edge ∈ DOMAIN T.Edges
    }
      child values for each edge. this creates a set of sets
      so we call union to flatten it.
  IN  current ∪ children

Returns the constructed radix tree for the set of keys Keys.
RECURSIVE radixTree(_, _)
LOCAL radixTree(Keys, Base) ≜
  IF Keys = {} THEN {} base case, no keys empty tree
   ELSE LET
    prefix ≜ longestPrefix(Keys)
      longest shared prefix
    base ≜ Base ∘ prefix
      our new base
    keys ≜ stripPrefix(Keys, prefix)
      keys for children, prefix stripped
    edgeLabels ≜ firstChars(stripPrefix(Keys, prefix))
      labels for each edge (single characters)
  IN  [
    Prefix ↦ prefix,
    Value ↦ IF prefix ∈ Keys THEN base ELSE ⟨⟩,
    Edges ↦ [L ∈ edgeLabels ↦ radixTree(filterPrefix(keys, L), base)]
  ]
```

```
Returns the minimal radix tree for the set of keys Keys.
RadixTree(Keys) ≜
```

LET
  $edgeLabels \triangleq firstChars(Keys)$
IN [
    The root of a radix tree is always a non-value that only has outward edges to the first sets of values.
  $Prefix \mapsto \langle\rangle,$
  $Value \mapsto \langle\rangle,$
  $Edges \mapsto [L \in edgeLabels \mapsto radixTree(filterPrefix(Keys, L), \langle\rangle)]$
]

Range returns all of the values that are in the radix tree T.
$Range(T) \triangleq range(T, \langle\rangle)$

Nodes returns all the nodes of the tree T.
RECURSIVE $Nodes(\_)$
$Nodes(T) \triangleq \{T\} \cup \text{UNION } \{Nodes(T.Edges[e]) : e \in \text{DOMAIN } T.Edges\}$

TRUE iff the radix tree T is minimal. A tree T is minimal if there are the minimum number of nodes present to represent the range of the tree.
$Minimal(T) \triangleq$
  $\neg \exists\, n \in (Nodes(T) \setminus \{T\}) :$ have to remove root T cause its always empty
    $\land\, Cardinality(\text{DOMAIN } n.Edges) = 1$
    $\land\, n.Value = \langle\rangle$

---

* Modification History
* Last modified *Fri Jul* 02 16:00:40 *PDT* 2021 by *mitchellh*
* Created *Mon Jun* 28 08:08:13 *PDT* 2021 by *mitchellh*