MODULE *SplitOrder*

This module implements a *hashmap* using *Shalev* et al.'s split-ordered list structure

EXTENDS *Integers*

CONSTANTS *NULL*, *PossibleKeys*, *PossibleValues*, *LoadFactor*, *MaxSize*

VARIABLES *keys*, *list*, *buckets*, *size*, *count*

ASSUME
$\land\ PossibleKeys \subseteq 0 \,..\, 15$
$\land\ NULL \notin PossibleKeys$
$\land\ NULL \notin PossibleValues$

Lookup table for bit-reversed keys with *MSB* set

$$\begin{aligned} SORegularKey(k) \triangleq\ & \text{CASE } k = 0 \to 1 \\ & \square k = 1 \to 9 \\ & \square k = 2 \to 5 \\ & \square k = 3 \to 13 \\ & \square k = 4 \to 3 \\ & \square k = 5 \to 11 \\ & \square k = 6 \to 7 \\ & \square k = 7 \to 15 \\ & \square k = 8 \to 1 \\ & \square k = 9 \to 9 \\ & \square k = 10 \to 5 \\ & \square k = 11 \to 13 \\ & \square k = 12 \to 3 \\ & \square k = 13 \to 11 \\ & \square k = 14 \to 7 \\ & \square k = 15 \to 15 \end{aligned}$$

Lookup table for bit-reversed keys

$$\begin{aligned} SODummyKey(k) \triangleq\ & \text{CASE } k = 0 \to 0 \\ & \square k = 1 \to 8 \\ & \square k = 2 \to 4 \\ & \square k = 3 \to 12 \\ & \square k = 4 \to 2 \\ & \square k = 5 \to 10 \\ & \square k = 6 \to 6 \\ & \square k = 7 \to 14 \\ & \square k = 8 \to 1 \\ & \square k = 9 \to 9 \\ & \square k = 10 \to 5 \\ & \square k = 11 \to 13 \\ & \square k = 12 \to 3 \end{aligned}$$

$\Box k = 13 \to 11$
$\Box k = 14 \to 7$
$\Box k = 15 \to 15$

Lookup table for parent buckets

$Parent(b) \triangleq$ CASE $b = 0 \to 0$
$\Box b = 1 \to 0$
$\Box b = 2 \to 0$
$\Box b = 3 \to 1$
$\Box b = 4 \to 0$
$\Box b = 5 \to 1$
$\Box b = 6 \to 2$
$\Box b = 7 \to 3$
$\Box b = 8 \to 0$
$\Box b = 9 \to 1$
$\Box b = 10 \to 2$
$\Box b = 11 \to 3$
$\Box b = 12 \to 8$
$\Box b = 13 \to 5$
$\Box b = 14 \to 6$
$\Box b = 15 \to 7$

The *Init* for split-order keys is initially empty the map maps every possible key to *NULL* The list initially contains only the 0 dummy node

$SOInit \triangleq \land keys = \{\}$
$\land list = [n \in 0 \,..\, 255 \mapsto$ IF $n = 0$ THEN $SODummyKey(0)$ ELSE $NULL]$
$\land buckets = [m \in PossibleKeys \mapsto$ IF $m = 0$ THEN $SODummyKey(0)$ ELSE $NULL]$
$\land size = 1$
$\land count = 0$

Inserting into the "linked list"

$ListInsert(k, v) \triangleq$ IF $list[k] = NULL$
THEN $list' = [list$ EXCEPT $![k] = v] \land count' = count + 1$
ELSE UNCHANGED $\langle list, count \rangle$

Removing from the "linked list"

$ListRemove(k) \triangleq$ IF $list[k] = NULL$
THEN UNCHANGED $\langle list, count \rangle$
ELSE $list' = [list$ EXCEPT $![k] = NULL] \land count' = count - 1$

Recursively initializes buckets

RECURSIVE $BucketInit(\_)$
$BucketInit(b) \triangleq$ IF $buckets[Parent(b)] = NULL \land Parent(b) \neq 0$
THEN $BucketInit(Parent(b))$
ELSE $buckets' = [buckets$ EXCEPT $![b] = SODummyKey(b)]$

Find the value of the key k in the bucket b Results in the value if b is initialized and k is in b

$ListFind(b,\, k) \triangleq \text{IF } k > b \land list[b] \neq NULL \text{ THEN } list[k] \text{ ELSE } NULL$

SOFind finds a key in the map

$$\begin{aligned} SOFind(k) \triangleq\ & \text{IF } buckets[k\%size] = NULL \\ & \text{THEN } NULL \quad \text{should initialize bucket, but also needs a "return value"} \\ & \text{ELSE } ListFind(buckets[k\%size],\, k) \end{aligned}$$

$$\begin{aligned} Min(a,\, b) \triangleq\ & \text{IF } a > b \text{ THEN } b \text{ ELSE } a \\ BucketGrow \triangleq\ & \text{IF } count' \neq 0 \land size \div count' > LoadFactor \\ & \text{THEN } size' = Min(size * 2,\, MaxSize) \\ & \text{ELSE TRUE} \end{aligned}$$

Inserting into the buckets

$$\begin{aligned} BucketInsert(k,\, v) \triangleq\ & \text{Either a bucket needs to be initialized} \\ \lor\ & \land buckets[k\%size] = NULL \\ & \land BucketInit(k\%size) \\ & \land ListInsert(SORegularKey(k),\, v) \\ & \land keys' = keys \cup \{k\} \\ & \text{Or the bucket is already initialized} \\ \lor\ & \land buckets[k\%size] \neq NULL \\ & \land ListInsert(SORegularKey(k),\, v) \\ & \land keys' = keys \cup \{k\} \\ & \land \text{UNCHANGED } \langle buckets \rangle \end{aligned}$$

Removing from the buckets

$$\begin{aligned} BucketRemove(k) \triangleq\ & \land ListRemove(SORegularKey(k)) \\ & \land keys' = keys \setminus \{k\} \\ & \land \text{UNCHANGED } \langle buckets \rangle \end{aligned}$$

$$\begin{aligned} SOInsert \triangleq\ & \land\ \exists\, k \in PossibleKeys : \\ & \qquad \exists\, v \in PossibleValues : \\ & \qquad\quad BucketInsert(k,\, v) \\ & \land \text{UNCHANGED } size \end{aligned}$$

$$\begin{aligned} SORemove \triangleq\ & \land\ \exists\, k \in PossibleKeys : \\ & \qquad BucketRemove(k) \\ & \land \text{UNCHANGED } size \end{aligned}$$

The Next for split order

$$\begin{aligned} SONext \triangleq\ & \land \lor SOInsert \\ & \quad \lor SORemove \\ & \land BucketGrow \end{aligned}$$

Split-order spec

$SOSpec \triangleq SOInit \land \Box[SONext]_{\langle keys, list, buckets, size, count\rangle}$

A refinement mapping of the *hashmap* spec with the map defined by the *SOFind* action

INSTANCE $hashmap$ WITH $map \leftarrow [k \in PossibleKeys \mapsto SOFind(k)]$

Split-order implements *hashmap*

THEOREM $SOSpec \Rightarrow HashmapSpec$

Type correctness of keys and values

$$SOTypeOK \triangleq \forall\, k \in keys :\\ \land\, k \in PossibleKeys\\ \land\, list[SORegularKey(k)] \in PossibleValues$$

Each bucket is either uninitialized or points to a dummy node

$$SOBucketOK \triangleq \forall\, n \in 0\,..\,size :\\ \lor\, buckets[n] = NULL\\ \lor\, buckets[n] = SODummyKey(0)$$