Contents

1 Module `Key` : Implementation of a triply-linked list, an extension of a doubly linked list.

A node in a triply linked list serves one of three purposes:

- `DictEnd` - the end of the list
- `Simple` - a normal linked list node with a next and prev pointer
- `Bridge` - a special linked list node. A `Bridge` node represents a key in common between two lists, and the two lists are maintained as two fields within the node.

For now, the types of `dict_id_t` and `val_t` have been hardcoded into the system, but later they could be modularized using a functor.

```
type dict_id_t = string
```

The type of the identifier for the dictionary.

```
type val_t = string
```

The type of the value of nodes in the list.

```
type t =
  | DictEnd
```

Indicates the end of the list.

```
  | Simple of simple_t
```

Indicates a simple node in the list.

```
  | Bridge of bridge_t
```

Indicates a bridge node between two lists.

The basic key type.

```
type link_t = {
  mutable next : t ;
```

The next key in the list

```
  mutable prev : t ;
```

The previous key in the list

```
  dict : dict_id_t ;
```

The dictionary id for this list

```
}
```

`link_t` represents a link to keys before and after the current item. It is used in one copy in the `Simple` node and in two copies in the `Bridge` node.

```
type simple_t = {
  svalue : val_t ;
```

The value of the simple node.

```
  link : link_t ;
```

The links to nodes before and after this node.

```
}
```

`simple_t` is the type of a `Simple` node. It has a value of `val_t` and a link of `link_t`.

```
type bridge_t = {
  bvalue : val_t ;
  d1 : link_t ;
  d2 : link_t ;
}
```

`bridge_t` is the type of a `Bridge` node. it is identical to a `Simple` node except for the fact that it has two links rather than just one.

`val create_simple : val_t -> dict_id_t -> simple_t`

Creates a simple key containing `value` for dictionary `did`.

`val belongs_to : dict_id_t -> t -> bool`

Tests whether the input node belongs to `dict`.

`exception No_link`

An exception that is thrown when trying to get the link of a `DictEnd` node.

`val get_link : dict_id_t -> t -> link_t`

Returns the link for the node for dictionary `dict`, raising `No_link` if the node is a `DictEnd`.

`val insert_after : t -> t -> dict_id_t -> t`

Inserts `ins` immediately after `key` in dictionary `dict`, returning the new `ins` node.

`val insert_before : t -> t -> dict_id_t -> t`

Inserts `ins` immediately before `key` in dictionary `dict`, returning the new `ins` node.

`val create_bridge : simple_t -> simple_t -> bridge_t`

Takes two keys `k1` and `k2` and forms a bridge between them.

`val to_list : t -> dict_id_t -> val_t list`

Converts a triply-linked list into a regular list starting with node `key` and using the `dict` link in any `Bridge` nodes.

`val prev_key : t -> dict_id_t -> t`

Returns the key before `key` in `did`.

`val next_key : t -> dict_id_t -> t`

Returns the key after `key` in `did`.

`val is_bridge : t -> bool`

Returns true if the node is a `Bridge`.

`val value : t -> val_t`

Returns the value of a node, failing on DictEnd.