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
- `Bridge` - a linked list node. A `Bridge` node represents a key in common between one or several lists, and the lists are maintained as a map from dictionary id to link.

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.

```
module StringMap :
   Map.Make(  sig
     type t = Key.dict_id_t
     val compare : 'a -> 'a -> int
  end )
```

Internal module to maintain owner set.

```
module StringSet :
   Set.Make(  sig
     type t = Key.dict_id_t
     val compare : 'a -> 'a -> int
  end )
```

Internal module to maintain link map.

```
type t =
  | DictEnd
```

Indicates the end of the list.

```
  | Bridge of bridge_t
```

Indicates a bridge node between one or several 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

```
}
```

`link_t` represents a link to keys before and after the current item. It is used in the `Bridge` node inside a map.

```
type bridge_t = {
  bvalue : val_t ;
```

The value of the `Bridge` node.

```
  owners : StringSet.t ;
```

The set of owners of the value in this node.

```
  mutable links : link_t StringMap.t ;
```

The set of dictionaries which this node is linked to.

```
}
```

`bridge_t` is the type of a `Bridge` node. A `Bridge` node uses a `StringMap` to keep track of which dictionaries this node is included in.

```
val empty_link : unit -> link_t
```

Creates an empty link.

```
val create : val_t -> StringMap.key -> t
```

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

```
val belongs_to : StringSet.elt -> t -> bool
```

Tests whether the input node belongs to `dict`.

```
val add_link : t -> StringMap.key -> unit
```

Adds a link to `did` in node `key`.

```
exception No_link
```

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

```
val get_link : StringMap.key -> t -> link_t
```

Returns the link for the node for dictionary `dict`, raising `No_link` if the node is a `DictEnd`. If the node doesn't contain a link for `dict`, then a link is added to the node and the new link is returned.

`val insert_after : t -> t -> StringMap.key -> t`

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

`val insert_before : t -> t -> StringMap.key -> t`

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

`val to_list : t -> StringMap.key -> 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 -> StringMap.key -> t`

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

`val next_key : t -> StringMap.key -> 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.