Assignment 3 - Trie Harder

Rahil Agrawal z5165505
Aditya Karia z5163287

COMP2111 18s1

1 Syntactic Data Type - Dict

We define a syntactic data type **Dict** that encapsulates a dictionary set W as follows :

$$\textbf{Dict} = (init^{Dict}, (W, x : [pre_j^{Dict}, post_j^{Dict}]_{j \in J}))$$

which consists of an initialisation predicate $init^{Dict} = (W = \{\})$ and the following operations:
proc addword^{Dict}(value w) . $W : [True, W = W_0 \cup w]$
proc checkword^{Dict}(value W, value w, result b) . $b : [True, b = (W = \{\})]$
proc delword^{Dict}(value w) . $W : [W \neq \{\}, W = W_0 \setminus w]$

2 Refinement to DictA

Were refining this to a data type DictA where we replace W with a Trie t. From Ass3 2018 S1 Specification, "A *trie domain* is a prefixclosed finite subset of L^*. A *trie* is a function from a trie domain to Booleans. Given a trie t we write **dom**t for its trie domain. Let T be the set of all tries."
The correspondance between the two data types is captured by the function $f : T \mapsto P(L^*)$ given by :

$$f(t) = \{w \in L^* \mid w \in \textbf{dom}t \wedge t(w) = 1\}$$

Also, from Ass3 2018 S1 Specification, "Formally, we write $v \leq w$ if word $v \in L^*$ is a prefix of $w \in L^*$, i.e., $\exists v' \in L^* \, (vv' = w)$. We write B for 0, 1 where 0 represents false and 1 true."
We define our With the aforementioned facts in mind, we define a concrete syntactic data type **DictA** that encapsulates a trie t as follows :

$$\textbf{DictA} = (init^{DictA}, (t, x : [pre_j^{DictA}, post_j^{DictA}]_{j \in J}))$$

which consists of an initialisation predicate $init^{DictA} = (t = \{\})$ and the following operations:
$proc\ addword^{DictA}$(value w) . $t : [True, post(addword^{DictA})]$ where

$$post(addword^{DictA}) = \mathbf{dom}t = \mathbf{dom}t_0 \cup \{v \in L^* \mid v \leq w\} \wedge t = t_0 \wedge \\ \forall v \in \mathbf{dom}t\,(t(v) = 1 \iff (w = y \vee (y \in \mathbf{dom}t_0 \wedge t_0(y) = 1)))$$

$proc\ checkword^{DictA}$(value t, value w, result b) . $b : [True, b = (t(w) = 1)]$
$proc\ delword^{DictA}$(value w) . $t : [t \neq \{\}, t(w) = 0]$

That this indeed is a refinement requires checking the relevant proof obligations:

$$init^{DictA} \Rightarrow init^{Dict}[^{f(t)}/_{W}] \tag{1}$$
$$pre_j^{Dict}[^{f(t)}/_{W}] \Rightarrow pre_j^{DictA}, for\ j \in J \tag{2}$$
$$pre_j^{Dict}[^{f(t_0),x_0}/_{W,x}] \wedge post_j^{DictA} \Rightarrow post_j^{Dict}[^{f(t_0),f(t)}/_{W_0,W}], for\ j \in J \tag{3}$$

We begin with (1).

$$\begin{aligned} & init^{DictA}[^{f(t_0),x_0}/_{W,x}] = f(t) \neq \{\} \\ \Rightarrow\ & \langle \text{def. of f} \rangle \\ & f(t) = \{\} \Rightarrow \\ \Rightarrow\ & \langle \text{def. of Dict} \rangle \\ & init^{Dict}[^{f(t)}/_{W}] \end{aligned}$$

Condition (2) is only required to be proven when the concrete pre-condition is non-trivial (not True). This is only the case in delword.

$$\begin{aligned} & pre_{delword}^{Dict}[^{f(t_0),x_0}/_{W,x}] = f(t) \neq \{\} \\ \Rightarrow\ & \langle \text{def. of f} \rangle \\ & w \in \mathbf{dom}t \wedge t(w) = 1 \\ \Rightarrow\ & \langle \text{def. of trie} \rangle \\ & t \neq \{\} \\ \Rightarrow\ & \langle \text{def. of } pre_{delword}^{DictA} \rangle \\ & pre_{delword}^{DictA} \end{aligned}$$

Finally, condition (4) needs to be checked for all operations.

For addword, we prove

$$pre^{Dict}_{addword}[^{f(t)}/_{W}] \land post^{DictA}_{addword} = True \land \mathbf{dom}t = \mathbf{dom}t_0 \cup \{v \in L^* \mid v \leq w\} \land$$
$$\forall v \in \mathbf{dom}t\,(t(v) = 1 \iff (w = y \lor (y \in \mathbf{dom}t_0 \land t_0(y) = 1)))$$
$\Rightarrow$ ⟨Trie is the same as old trie but we added the prefixes of w⟩
⟨without changing their old mappings⟩
⟨New prefixes are mapped to 0 and w is mapped to 1.⟩
⟨This means we added a word w if it did not already exist, def. of trie and f⟩
$$f(t) = f(t_0) \cup w$$
$\Rightarrow$ ⟨Definition of $addword^{Dict}$⟩
$$post_{addword^{Dict}}[^{f(t_0),f(t)}/_{W_0,W}]$$

For checkword, we prove

$$pre^{Dict}_{checkword}[^{f(t)}/_{W}] \land post^{DictA}_{checkword} = True \land b = (t(w) = 1)$$
$\Rightarrow$ ⟨b is 1 if w is in $\mathbf{dom}t$ and trie maps w to 1, 0 otherwise. def. of trie.⟩
$\Rightarrow$ ⟨b = 1 means w is in our set of words. def of f⟩
$$b = (w \in f(t))$$
$\Rightarrow$ ⟨def. of $checkword^{Dict}$⟩
$$post_{checkword^{Dict}}[^{f(t_0),f(t)}/_{W_0,W}]$$

For delword, we prove

$$pre^{Dict}_{delword}[^{f(t)}/_{W}] \land post^{DictA}_{delword} = W \neq \{\} \land t(w) = 0$$
$\Rightarrow$ ⟨def of f⟩
$$w \notin f(t)$$
$\Rightarrow$ ⟨Clearly⟩
$$f(t) = f(t_0) \setminus w$$
$\Rightarrow$ ⟨def. of $delword^{Dict}$⟩
$$post_{delword^{Dict}}[^{f(t_0),f(t)}/_{W_0,W}]$$

3 Derivation

Before we refine the code for our functions for **DictA** , we define certain predicates to help us during derivation.

We associate a natural number i with each element x of a set X and define $y_X^{(i)}$ to be the i^{th} element.
The total number of elements in a set X is given by $size(X)$.

```
    $\mathbf{proc}\ delword(\mathbf{value}\ w) \cdot t : [\ t \neq \{\}, t(w) = 0\ ]$
$\sqsubseteq$    $\langle$i-loc, seq$\rangle$
    $t, i : [\ t \neq \{\}, i = 0\ ];$
$\sqsubseteq$    $\langle$ass$\rangle$
    $\mathbf{var\ i := 0;}$
    $t, i : [\ i = 0, t(w) = 0\ ]$
$\sqsubseteq$    $\langle\mathbf{proc,}\ i = 0 \Rightarrow i \leq size(t)\rangle$
    $\mathbf{delR(w, i);}$

    $\mathbf{proc}\ checkword(\mathbf{value}\ b, \mathbf{value}\ w) \cdot b, t : [\ TRUE, b = (t(w) = 1)\ ]$
$\sqsubseteq$    $\langle$i-loc, seq$\rangle$
    $b, t, i : [\ TRUE, i = 0\ ];$
$\sqsubseteq$    $\langle$ass$\rangle$
    $\mathbf{var\ i := 0;}$
    $t, i : [\ i = 0, b = (t(w) = 1)\ ]$
$\sqsubseteq$    $\langle\mathbf{proc,}\ i = 0 \Rightarrow i \leq size(t)\rangle$
    $\mathbf{checkR(w, b, i);}$


    $\mathbf{proc}\ addword(\mathbf{value}\ w) \cdot b, t : [\ TRUE, post_{addword}^{DictA}\ ]$
$\sqsubseteq$    $\langle$i-loc, seq$\rangle$
    $b, t, i : [\ TRUE, i = 0\ ];$
$\sqsubseteq$    $\langle$ass$\rangle$
    $\mathbf{var\ i := 0;}$
    $t, i : [\ i = 0, post_{addword}^{DictA}\ ]$
$\sqsubseteq$    $\langle\mathbf{proc,}\ i = 0 \Rightarrow i \leq size(t)\rangle$
    $\mathbf{addR(w, b, i);}$
```

```
        proc $delR$(value $w$, value $i$) $\cdot\, t : [\ i \le size(t), t(w) = 0\ ]$
    $\sqsubseteq$    $\langle$if$\rangle$
        if i $\ne$ size(t)
        then $\llcorner t, i : [i < size(t), t(w) = 0]\lrcorner_{(1)}$
        else $t, i : [i = size(t), t(w) = 0]$
    $\sqsubseteq$    $\langle$skip - Proof(1)$\rangle$
            skip;
        fi;
(1) $\sqsubseteq$    $\langle$if$\rangle$
        if $y_t^i = w \mapsto t(w)$
        then $t, i : [i < size(t) \wedge y_t^i = w \mapsto t(w), t(w) = 0]$
    $\sqsubseteq$    $\langle$ass, $0 = 0\rangle$
            $t(w) := 0$;
        else $\llcorner t, i : [i < size(t) \wedge y_t^i \ne w \mapsto t(w), t(w) = 0]\lrcorner_{(2)}$
        fi;
(2) $\sqsubseteq$    $\langle$seq, con c$\rangle$
        $t, i : [i < size(t) \wedge y_t^i \ne w \mapsto t(w) \wedge i = c, i = c + 1]$;
    $\sqsubseteq$    $\langle$ass, $c = i_0 \wedge i = c + 1 \Rightarrow i = i_0 + 1\rangle$
            $i := i + 1$;
        $t, i : [i = c + 1, t(w) = 0]$
    $\sqsubseteq$    $\langle$proc, $c = i < size(t) \Rightarrow i + 1 \le size(t)\rangle$
            delR(w, i);
```

```
    proc $checkR$(value $w$, result $b$, value $i$) $\cdot\, t : [\ i \le size(t), b = (t(w) = 1)\ ]$
$\sqsubseteq$    $\langle$if$\rangle$
    if i $\ne$ size(t)
    then $\llcorner t, i : [i < size(t), b = (t(w) = 1)]\lrcorner_{(1)}$
    else $t, i : [i = size(t), b = (t(w) = 1)]$
$\sqsubseteq$    $\langle$skip - Proof(2)$\rangle$
        skip;
    fi;
```

```
(1) $\sqsubseteq$    $\langle$if$\rangle$
    if $\mathbf{y_t^i = w \mapsto 1}$
    then $t, i : [i < size(t) \wedge y_t^i = w \mapsto 1, b = (t(w) = 1)]$
    $\sqsubseteq$    $\langle$ass, $w \mapsto 1 \Rightarrow t(w) = 1 \Rightarrow b = TRUE\rangle$
         $\mathbf{b := TRUE}$;
    else $\llcorner t, i : [i = size(t) \wedge y_t^i \neq w \mapsto 1, b = (t(w) = 1)]\lrcorner_{(2)}$
    fi;
(2) $\sqsubseteq$    $\langle$seq, con c$\rangle$
    $t, i : [i < size(t) \wedge y_t^i \neq w \mapsto 1 \wedge i = c, i = c + 1]$;
    $\sqsubseteq$    $\langle$ass, $c = i_0 \wedge i = c + 1 \Rightarrow i = i_0 + 1\rangle$
         $\mathbf{i := i + 1}$;
    $t, i : [i = c + 1, b = (t(w) = 1)]$
    $\sqsubseteq$    $\langle$proc, $c = i < size(t) \Rightarrow i + 1 \leq size(t)\rangle$
         $\mathbf{checkR(w, b, i)}$;
```

Before we derive code for addR, we define S to be the set of all prefixes of a word w.

```
    proc $addR$(value $w$, value $i$) $\cdot\, t : \left[\ i \leq size(t), post_{addword}^{DictA}\ \right]$
    $\sqsubseteq$    $\langle$if$\rangle$
    if $\mathbf{i \neq size(S)}$
    then $\llcorner t, i : [i < size(S), post_{addword}^{DictA}]\lrcorner_{(1)}$
    else $t, i : [i < size(S), post_{addword}^{DictA}]$
    $\sqsubseteq$    $\langle$skip - Proof(3)$\rangle$
         skip;
    fi;
(1) $\sqsubseteq$    $\langle$seq, con G$\rangle$
    $t, i : [i < size(S) \wedge G = \mathbf{dom}t, i < size(S) \wedge \mathbf{dom}t = G \cup y_S^i]$;
    $\sqsubseteq$    $\langle$ass, $G = \mathbf{dom}t_0 \wedge \mathbf{dom}t = G \cup y_S^i\rangle$
         $\mathbf{domt := domt_0 \cup y_S^i}$;
    $\llcorner t, i : [i < size(S) \wedge \mathbf{dom}t = G \cup y_S^i, post_{addword}^{DictA}]\lrcorner_{(2)}$
```

```
(2) $\sqsubseteq$    $\langle$if$\rangle$
    if $\mathbf{t_0(y^i_S) \neq 0, 1}$
    then $\llcorner t, i : [i < size(S) \wedge \mathbf{dom}t = G \cup y^i_S \wedge t(y^i_S) \neq 0, 1, post^{DictA}_{addword}] \lrcorner_{(3)}$
    else $\llcorner t, i : [i < size(S) \wedge \mathbf{dom}t = G \cup y^i_S \wedge t(y^i_S) = 0, 1, post^{DictA}_{addword}] \lrcorner_{(4)}$
    fi;
(3) $\sqsubseteq$    $\langle$if$\rangle$
    if $y^i_S = w$
    then $t, i : [i < size(S) \wedge \mathbf{dom}t = G \cup y^i_S \wedge t(y^i_S) \neq 0, 1 \wedge y^i_S = w, post^{DictA}_{addword}]$
    $\sqsubseteq$    $\langle$ass, $t_0(y^i_S) \neq 0, 1 \wedge y^i_S = w \Rightarrow$ Add to trie and map to 1 ($\because$ w is added)$\rangle$
        $\mathbf{t = t_0 : y^i_S \mapsto 1}$
    else $t, i : [i < size(S) \wedge \mathbf{dom}t = G \cup y^i_S \wedge t(y^i_S) \neq 0, 1 \wedge y^i_S \neq w, post^{DictA}_{addword}]$
    $\sqsubseteq$    $\langle$ass,$t_0(y^i_S) \neq 0, 1 \wedge y^i_S \neq w \Rightarrow$Add to trie and map to 0($\because$ prefix of w is added)$\rangle$
        $\mathbf{t = t_0 : y^i_S \mapsto 0}$
    fi;
(4) $\sqsubseteq$    $\langle$seq, con c$\rangle$
    $t, i : [i < size(S) \wedge \mathbf{dom}t = G \cup y^i_S \wedge t(y^i_S) = 0, 1 \wedge i = c, i = c + 1];$
    $\sqsubseteq$    $\langle$ass, $c = i_0 \wedge i = c + 1 \Rightarrow i = i_0 + 1$$\rangle$
        $\mathbf{i := i + 1;}$
    $t, i : [i = c + 1, post^{DictA}_{addword}]$
    $\sqsubseteq$    $\langle$proc, $c = i < size(t) \Rightarrow i + 1 \leq size(t)$$\rangle$
        $\mathbf{addR(w, i);}$
```

4 C Code

```c
#include "dict.h"
#include <stdio.h>
#include <stdlib.h>

void newdict(Dict *dp) {
    *dp = malloc(sizeof(TNode));
    if (*dp == NULL) {
        return;
    }
    for (int i = 0; i < VECSIZE; i++) {
        ((*dp)->cvec)[i] = NULL;
    }
    (*dp)->eow = FALSE;
}
```

```

void addR (Dict r, const word w, int i) {
    if (w[i] == '\0') {
        r->eow = TRUE;
        return;
    } else {
        if ((r->cvec)[w[i] - 'a'] == NULL) newdict(&((r->cvec)[w[i] - 'a']));
        r = (r->cvec)[w[i] - 'a'];
        i = i + 1;
        addR(r, w, i);
    }
}

bool checkR(Dict r, const word w, int i) {
    if (r == NULL) return FALSE;
    if (w[i] == '\0') {
        if (r->eow == TRUE) {
            return TRUE;
        }
        return FALSE;
    } else {
        r = (r->cvec)[w[i] - 'a'];
        i = i + 1;
        return checkR(r, w, i);
    }
}

void delR(Dict r, const word w, int i) {
    if (r == NULL) return;
    if (w[i] == '\0') {
        r->eow = FALSE;
        return;
    } else {
        r = (r->cvec)[w[i] - 'a'];
        i = i + 1;
        delR(r, w, i);
    }
}

void printDictR(const Dict r, char str[], int level)
{
    if (r->eow == TRUE)
    {
        str[level] = '\0';
```

```
            printf("%s\n", str);
        }

        int i;
        for (i = 0; i < VECSIZE; i++)
        {
            if (r->cvec[i] != NULL)
            {
                str[level] = i + 'a';
                printDictR(r->cvec[i], str, level + 1);
            }
        }
}

void barf(char *s) {
        fprintf(stderr, "%s\n", s);
}

void addword(const Dict r, const word w) {
        int i = 0;
        addR(r, w, i);
}

bool checkword (const Dict r, const word w) {
        int i = 0;
        return checkR(r, w, i);
}

void delword (const Dict r, const word w) {
        int i = 0;
        delR(r, w, i);
}

void printDict(const Dict r) {
        char str[100];
        int level = 0;
        printDictR(r, str, level);
}
```