

# Assignment 3

Ruofei HUANG(z5141448)      Anqi ZHU(z5141541)

May 29, 2018

## 1 Task 1

### 1.1 Some useful symbol definitions about manipulating words

A **word**, as mentioned in the assignment specification, is defined as a finite sequence of letters over  $L = 'a', \dots, 'z'$ . The specification already defines the relationship of ' $\leq$ ' between a **word**  $v$  and a **word**  $w$  (which means  $v$  is a prefix of  $w$  or  $w$  itself if  $v \leq w$ ).

So we would like to further this definition and define a relationship of ' $<$ ' when  $v$  is a proper prefix of  $w$  which cannot be  $w$  itself if  $v < w$ . That means:

$$v < w \Leftrightarrow v \leq w \wedge v \neq w$$

We also would like to define a symbol  $|w|$  that represents the length of a *word*  $w$ . We formally define this by:

$$|w| = \begin{cases} 0 & \text{if } w = \epsilon \\ 1 + |w'| & \text{else} \end{cases}$$

$$\text{where } \exists l \in L \{w = w' l\}$$

### 1.2 Syntactic(Abstract) Data Type *Dict*

Inspired by the program sketch and the assignment statement, we could describe the syntactic data type *Dict* as below (the encapsulated state would be a dictionary word set  $W$ ).

$$Dict = (W = \phi, \left( \begin{array}{l} \mathbf{proc} \text{ addword}^{Dict}(\mathbf{word} \ w) \cdot b, W : [\mathbf{TRUE}, b = b_0 \wedge W = W_0 \cup \{w\}] \\ \mathbf{func} \text{ checkword}^{Dict}(\mathbf{word} \ w) : \\ \quad \mathbb{B} \cdot \mathbf{var} \ b \cdot b, W : [\mathbf{TRUE}, b = (w \in W) \wedge W = W_0]; \mathbf{return} \ b \\ \mathbf{proc} \text{ delword}^{Dict}(\mathbf{word} \ w) \cdot b, W : [w \in W, b = b_0 \wedge W = W_0 \setminus \{w\}] \end{array} \right) )$$

## 2 Task 2

### 2.1 Data Type Refinement

Now we would like to refine *Dict* to a second data type *DictA* where we replace *W* with a trie *t*. We would also like to define the domain of a *t* as **dom**(*t*). We shall use this definition later in our refinement.

#### 2.1.1 Inductive Relation Predicate

The correspondence between the two state space *W* and *t* is captured by the inductively defined predicate.

$$r = (W = \{w \in \mathbf{dom}(t) | t(w) = 1\})$$

which we can translate into a relation function that transfers a concrete state space *t* to an abstract state space *W*:

$$f(t) = \{w \in \mathbf{dom}(t) | t(w) = 1\}$$

With that in mind, we can propose the initialisation predicate and corresponding operations of *DictA*.

#### 2.1.2 Initialisation Predicate

We would like to define the initialisation predicate of *DictA* as follows:

$$\mathit{init}^{DictA} = (t := \{\epsilon \mapsto 0\})$$

#### 2.1.3 Operations

We would like to define the operations of *DictA* as follows:

```
proc addwordDictA(word w) · b, t :  
  [ TRUE, b = b0 ∧  
     $t = \left( \begin{array}{l} w \notin \mathbf{dom}(t) \wedge t = t_0 \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \mathbf{dom}(t) | w' \mapsto 0\} \\ w \in \mathbf{dom}(t) \wedge t = (t_0 : w \mapsto 1) \end{array} \right)$  ]  
func checkwordDictA(word w) :  $\mathbb{B}$  · var b · b, t :  
  [TRUE, t = t0 ∧ b = (w ∈ dom(t) ∧ t(w) = 1)]; return b  
proc delwordDictA(word w) · b, t :  
  [TRUE, b = b0 ∧ (w ∉ dom(t) ∨ t := (t : w ↦ 0))]
```

## 2.2 Proof of Refinement

Now we would like to start proving the refinements of the initialization and each operation from  $t$  to  $W$ .

We start from proving the refinement between  $init^{DictA}$  and  $init^{Dict}$ .

$$\begin{aligned}
& init^{DictA} \Rightarrow init^{Dict}[f(t)/W] \\
\Leftrightarrow & \quad \langle \text{Definition of } init^{DictA} \text{ and } init^{Dict} \rangle \\
& \forall w \in \mathbf{dom}(t) (t(w) = 0) \Rightarrow W = \phi
\end{aligned}$$

Since all our precondition of concrete is trivial which all of them are TRUE, we don't need to proof the condition (3<sub>f</sub>). But condition (4<sub>f</sub>) must be checked for all three operations. For the *addword* we proof:

$$\begin{aligned}
& pre_{addword}^{Dict}[f(t_0)/W] \wedge post_{addword}^{DictA} \\
\Leftrightarrow & \quad \langle \text{Definition of } addword^{Dict} \text{ and } addword^{DictA} \rangle \\
& TRUE[f(t_0)/W] \wedge b = b_0 \wedge t = t_0 \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \mathbf{dom}(t) | w' \mapsto 0\} \\
\Rightarrow & \quad \langle \text{Definition of } f \rangle \\
& f(t) = f(t_0 \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \mathbf{dom}(t) | w' \mapsto 0\}) \\
\Leftrightarrow & \quad \langle \text{Logic} \rangle \\
& f(t) = f(t_0) \cup \{w\} \\
\Leftrightarrow & \quad \langle \text{Definition of } addword^{Dict} \text{ and } addword^{DictA} \rangle \\
& post_{addword}^{Dict}[f(t_0), f(t)/W_0, W]
\end{aligned}$$

For the *checkword* we proof:

$$\begin{aligned}
& pre_{checkword}^{Dict}[f(t_0)/W] \wedge post_{checkword}^{DictA} \\
\Leftrightarrow & \quad \langle \text{Definition of } checkword^{DictA} \text{ and } checkword^{Dict} \rangle \\
& TRUE[f(t_0)/W] \wedge b = (w \in \mathbf{dom}(t) \wedge t = t_0) \\
\Rightarrow & \quad \langle \text{Definition of } f \rangle \\
& b = (w \in f(t) \wedge t(w) = 1) \wedge t = t_0 \\
\Leftrightarrow & \quad \langle \text{Definition of } checkword^{DictA} \text{ and } checkword^{Dict} \rangle \\
& post_{checkword}^{Dict}[f(t_0), f(t)/W_0, W]
\end{aligned}$$

For the *delword* we proof:

$$\begin{aligned}
& pre_{delword}^{Dict} [f(t_0)/W] \wedge post_{delword}^{DictA} \\
\Leftrightarrow & \langle \text{Definition of } delword^{DictA} \text{ and } delword^{Dict} \rangle \\
& w \in f(t_0) \wedge b = b_0 \wedge (w \notin \mathbf{dom}(t) \vee t := t : w \mapsto 0) \\
\Rightarrow & \langle \text{Definition of } f \rangle \\
& f(t) = f(t_0) \setminus \{w\} \\
\Leftrightarrow & \langle \text{Definition of } delword^{DictA} \text{ and } delword^{Dict} \rangle \\
& post_{delword}^{Dict} [f(t_0), f(t)/W_0, W]
\end{aligned}$$

### 3 Task 3

We derive our code in to five parts by *init*, data type's operations and a *popWord* for recursive calls.

#### 3.1 init

From the spec we have:

$$\begin{aligned}
& \mathbf{dom}(t) = \{\epsilon\} \wedge f(t) = \phi \\
\sqsubseteq & \langle \text{ass} \rangle \\
& t := \{\epsilon \mapsto 0\}
\end{aligned}$$

#### 3.2 popWord

A popWord is for us to have a easily use recursive calls, it is also a bridge between C and our Toy language<sup>1</sup>. The purpose of this function is to pop the first given letters of a given word.

##### 3.2.1 Math Function

$$\begin{aligned}
& POPWORD(w, i) = w' \\
& \text{where } w, w' \in \mathbf{word} \wedge i \in \mathbb{N} \wedge w' \leq w \wedge |w'| = i
\end{aligned}$$

##### 3.2.2 Toy Language Function

```

func popWord(value w, value i).
  └ var w' · w' : [i ≤ |w|, w' ≤ w ∧ |w'| = i]; return w' ┘(P1)

```

---

<sup>1</sup>This fuction would not appear in our C code, the pointer to the node would be solve this problem. And pointer is a difficult part for Toy Language to express and proof.

### 3.3 addword

From the spec<sup>2</sup> we have:

```

proc addwordDictA(value w).
   $\sqsubseteq b, t : [\text{TRUE}, b = b_0 \wedge t = t_0 \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \text{dom}(t) | w' \mapsto 0\}] \sqsubseteq_{(A1)}$ 

(A1)  $\sqsubseteq$   $\langle \text{c-frame} \rangle$ 
       $t : [\text{TRUE}, t = t_0 \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \text{dom}(t) | w' \mapsto 0\}]$ 

```

### 3.4 checkword

From the spec we have:

```

func checkwordDictA(value w)
   $\sqsubseteq \mathbb{B} \cdot \text{var } b \cdot b, t : [\text{TRUE}, b = (w \in \text{dom}(t) \wedge t = t_0)]; \text{ return } b \sqsubseteq_{(C1)}$ 

(C1)  $\sqsubseteq$   $\langle \text{c-frame} \rangle$ 
       $\mathbb{B} \cdot \text{var } b \cdot b : [\text{TRUE}, b = (w \in \text{dom}(t))]; \text{ return } b$ 
       $\sqsubseteq$   $\langle \text{proc}, 0 \leq |w| \rangle$ 
      return doCheckword(w, 0);

```

Where we define a recursive procedure call to do the dirty work also align the pre and post condition:

```

func doCheckword(value w, value index :  $\mathbb{N}$ )
   $\sqsubseteq \mathbb{B} \cdot \text{var } b \cdot b, \text{var } index[index \leq w, b = (w \in \text{dom}(t) \wedge t(w) = 1)]; \text{ return } b \sqsubseteq_{(C2)}$ 

(C2)  $\sqsubseteq$   $\langle \text{if} \rangle$ 
      if index < |w|
      then  $\sqsubseteq b, index \sqsubseteq [index < |w| \wedge \text{pre}(C2), \text{post}(C2)] \sqsubseteq_{(C3-1)}$ 
      else  $\sqsubseteq b, index \sqsubseteq [index \geq |w| \wedge \text{pre}(C2), \text{post}(C2)] \sqsubseteq_{(C3-2)}$ 
      fi

(C3-1)  $\sqsubseteq$   $\langle \text{if} \rangle$ 
      if (POPWORD(w, index)  $\in \text{dom}(t)$ )
      then
      else
      fi

(C3-1)  $\sqsubseteq$   $\langle \text{func} \rangle$ 
      return doCheckword(w, index + 1);

(C3-2)  $\sqsubseteq$   $\langle index \geq |w| \wedge index \leq |w| \Rightarrow index = |w| \text{ and definition of } \text{post}(C2) \rangle$ 
      return (w  $\in \text{dom}(t) \wedge t(w) = 1$ )

```

---

<sup>2</sup>Definition of this is in the Assignment 3 requirements of cs2111.

### 3.5 delword

From the spec we have:

```
proc delwordDictA(value w)  
   $\perp \cdot b, t : [\text{TRUE}, b = b_0 \wedge (w \notin \mathbf{dom}(t) \vee t := t : w \mapsto 0)] \perp_{(D1)}$ 
```