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 $L = 'a', .., 'z'$. 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:

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

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

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).

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

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 $\mathbf{dom}(t)$. 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:

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

2.1.3 Operations

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

$$
\begin{aligned}
&\mathbf{proc}\ addword^{DictA}(\mathbf{word}\ w) \cdot b, t : \\
&\quad [\ \text{TRUE}, b = b_0 \wedge \\
&\qquad t = \begin{pmatrix} w \notin \mathbf{dom}(t) \wedge t = t_0 \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \mathbf{dom}(t) | w' \mapsto 0\} \\ w \in \mathbf{dom}(t) \wedge t = (t_0 : w \mapsto 1) \end{pmatrix}] \\
&\mathbf{func}\ checkword^{DictA}(\mathbf{word}\ w) : \mathbb{B} \cdot \mathbf{var}\ b \cdot b, t : \\
&\quad [\text{TRUE}, t = t_0 \wedge b = (w \in \mathbf{dom}(t) \wedge t(w) = 1)];\ \mathbf{return}\ b \\
&\mathbf{proc}\ delword^{DictA}(\mathbf{word}\ w) \cdot b, t : \\
&\quad [\text{TRUE}, b = b_0 \wedge (w \notin \mathbf{dom}(t) \vee t := (t : w \mapsto 0))]
\end{aligned}
$$

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_f).But condition (4_f) must be checked for all three operations. For the *addword* we proof:

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

For the *checkword* we proof:

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

For the *delword* we proof:

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

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:

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

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[1]. The purpose of this function is to pop the first given letters of a given word.

3.2.1 Math Function

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

3.2.2 Toy Language Function

$$
\begin{aligned}
& \mathbf{func}\ popWord(\mathbf{value}\ w, \mathbf{value}\ i) \cdot \\
& \quad \llcorner \mathbf{var}\ w' \cdot w' : [i \leq |w|, w' \leq w \wedge |w'| = i]; \mathbf{return}\ w' \lrcorner_{(P1)}
\end{aligned}
$$

[1]This fucntion 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[2] we have:

$$\textbf{proc } addword^{DictA}(\textbf{value } w)\cdot$$
$$\llcorner b, t : [\textsc{true}, b = b_0 \wedge t = t_0 \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \textbf{dom}(t) | w' \mapsto 0\}]\lrcorner_{(A1)}$$

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

3.4 checkword

From the spec we have:

$$\textbf{func } checkword^{DictA}(\textbf{value } w)$$
$$\llcorner \mathbb{B} \cdot \textbf{var } b \cdot b, t : [\textsc{true}, b = (w \in \textbf{dom}(t) \wedge t = t_0)];\ \textbf{return } b\lrcorner_{(C1)}$$

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

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

$$\textbf{func } doCheckword(\textbf{value } w, \textbf{value } index : \mathbb{N})$$
$$\llcorner \mathbb{B} \cdot \textbf{var } b \cdot b, \textbf{var } index[prefix \leq w, b = (w \in \textbf{dom}(t))];\ \textbf{return } b\lrcorner_{(C2)}$$

$$(C2) \sqsubseteq \quad \langle\text{if}\rangle$$
$$\textbf{if } index < |w|$$
$$\textbf{then} \llcorner b, indexp[index < |w| \wedge pre(C2), post(C2)]\lrcorner_{(C3-1)}$$
$$\textbf{else} \llcorner b, index[index < |w| \wedge pre(C2), post(C2)]\lrcorner_{(C3-2)}$$
$$\textbf{fi}$$
$$(C3-1) \sqsubseteq \quad \langle\text{func}\rangle$$
$$\textbf{return } doCheckword(w, index + 1);$$

3.5 delword

From the spec we have:

$$\textbf{proc } delword^{DictA}(\textbf{value } w)$$
$$\llcorner \cdot b, t : [\textsc{true}, b = b_0 \wedge (w \notin \textbf{dom}(t) \vee t := t : w \mapsto 0)]\lrcorner_{(D1)}$$

[2] Definition of this is in the Assignment 3 requirements of cs2111.