

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 = t0 \ {w ↦ 0} ∪ {w ↦ 1} ∪ {w' < w ∧ w' ∉ dom(t) | w' ↦ 0}]  
func checkwordDictA(word w) :  $\mathbb{B}$  · var b · b, t :  
  [TRUE, t = t0 ∧ b = (∀ w' ≤ w (w' ∈ dom(t)) ∧ t(w) = 1)]; return b  
proc delwordDictA(word w) · b, t :  
  [TRUE, b = b0 ∧ (w ∉ dom(t) ∨ t = t0 : 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 $init^{Dict}$.

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

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:

$$\begin{aligned}
& pre_{addword}^{Dict}[f(t_0)/w] \wedge post_{addword}^{DictA} \\
\Leftrightarrow & \langle \text{Definition of } addword^{Dict} \text{ and } addword^{DictA} \rangle \\
& TRUE[f(t_0)/w] \wedge b = b_0 \wedge t = t_0 \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \mathbf{dom}(t) | w' \mapsto 0\} \\
\Rightarrow & \langle \text{Definition of } f \rangle \\
& f(t) = f(t_0 \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \mathbf{dom}(t) | w' \mapsto 0\}) \\
\Leftrightarrow & \langle \text{Logic} \rangle \\
& f(t) = f(t_0) \cup \{w\} \\
\Leftrightarrow & \langle \text{Definition of } addword^{Dict} \text{ and } addword^{DictA} \rangle \\
& post_{addword}^{Dict}[f(t_0), f(t)/w_0, w]
\end{aligned}$$

For the *checkword* we proof:

$$\begin{aligned}
& pre_{checkword}^{Dict}[f(t_0)/w] \wedge post_{checkword}^{DictA} \\
\Leftrightarrow & \langle \text{Definition of } checkword^{DictA} \text{ and } checkword^{Dict} \rangle \\
& TRUE[f(t_0)/w] \wedge t = t_0 \wedge b = (\forall w' \leq w (w' \in \mathbf{dom}(t)) \wedge t(w) = 1) \\
\Rightarrow & \langle \text{Definition of } f \rangle \\
& b = (w \in f(t) \wedge t(w) = 1) \wedge t = t_0 \\
\Leftrightarrow & \langle \text{Definition of } checkword^{DictA} \text{ and } checkword^{Dict} \rangle \\
& post_{checkword}^{Dict}[f(t_0), f(t)/w_0, w]
\end{aligned}$$

For the *delword* we proof:

$$\begin{aligned}
& pre_{delword}^{Dict}[f(t_0)/w] \wedge post_{delword}^{DictA} \\
\Leftrightarrow & \langle \text{Definition of } delword^{DictA} \text{ and } delword^{Dict} \rangle \\
& w \in f(t_0) \wedge b = b_0 \wedge (w \notin \mathbf{dom}(t) \vee t = t_0 : w \mapsto 0) \\
\Rightarrow & \langle \text{Definition of } f \rangle \\
& f(t) = f(t_0) \setminus \{w\} \\
\Leftrightarrow & \langle \text{Definition of } delword^{DictA} \text{ and } delword^{Dict} \rangle \\
& post_{delword}^{Dict}[f(t_0), f(t)/w_0, w]
\end{aligned}$$

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:

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

3.2 popWord

We would like to define a semantic function *popWord*(*var word*, *var index*) that returns a substring of the word *w* starting from the first letter to the *index*'th letter. A *popWord* is for us to have a easily use recursive calls, it is also a bridge between C and our Toy language¹. The purpose of this function is to pop the first given letters of a given word.

3.2.1 Math Function

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

3.2.2 Toy Language Function

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

3.3 addword

From the spec² we have:

$$\begin{aligned} & \mathbf{proc} \text{ addword}^{\text{DictA}}(\mathbf{value} \ w) \cdot \\ & \quad \llbracket b, t : [\text{TRUE}, b = b_0 \wedge t = t_0 \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \mathbf{dom}(t) | w' \mapsto 0\}] \rrbracket_{(A1)} \end{aligned}$$

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

¹This 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.

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

3.4 checkword

From the spec we have:

```
func checkwordDictA(value w) :  $\mathbb{B}$ .
  var b ·  $\sqsubseteq$  b, t : [TRUE, b = ( $\forall w' \leq w$  ( $w' \in \mathbf{dom}(t)$ )  $\wedge$   $t(w) = 1$ )] ;  $\neg(C1)$  return b
```

```
(C1)  $\sqsubseteq$        $\langle$ c-frame $\rangle$ 
           b : [TRUE, b = ( $\forall w' \leq w$  ( $w' \in \mathbf{dom}(t)$ )  $\wedge$   $t(w) = 1$ )] ;
            $\sqsubseteq$        $\langle$ proc,  $0 \leq |w|$  and  $\epsilon \in \mathbf{dom}(t)$  since  $init^{DictA}$  $\rangle$ 
           b := doCheckword(w, 0);
```

doCheckword() is a recursive function call that does the major checking work through the whole trie *t* with a variable *i* to locate which sub-trie it is checking in. The definition of the function and the refinement of its linking procedure is as follows.

3.4.1 Definition of doCheckword

```
func doCheckword(value w, value index :  $\mathbb{N}$ ) :  $\mathbb{B}$ .
  var b ·  $\sqsubseteq$  b :  $\left[ \begin{array}{l} 0 \leq index \leq |w| \wedge \\ \forall w' \leq w \wedge |w'| \leq index \ (w' \in \mathbf{dom}(t)), \\ b = (\forall w' \leq w \ (w' \in \mathbf{dom}(t)) \wedge t(w) = 1) \end{array} \right] \neg(C2); \text{return } b$ 
```

3.4.2 Refinement of the procedure in doCheckword

$$\begin{aligned}
(C2) &\sqsubseteq \langle \text{if} \rangle \\
&\quad \text{if } index < |w| \\
&\quad \text{then } \perp b : [index < |w| \wedge pre(C2), post(C2)] \neg(C3-1) \\
&\quad \text{else } \perp b : [index \geq |w| \wedge pre(C2), post(C2)] \neg(C3-2) \\
&\quad \text{fi} \\
(C3-1) &\sqsubseteq \langle \text{if} \rangle \\
&\quad \text{if } (POPWORD(w, index + 1) \in \mathbf{dom}(t)) \\
&\quad \text{then } \perp b : \left[\begin{array}{l} pre(C3-1) \wedge POPWORD(w, index + 1) \in \mathbf{dom}(t), \\ post(C2) \end{array} \right] \neg(C3-1-1) \\
&\quad \text{else } \perp b : \left[\begin{array}{l} pre(C3-1) \wedge POPWORD(w, index + 1) \notin \mathbf{dom}(t), \\ post(C2) \end{array} \right] \neg(C3-1-2) \\
&\quad \text{fi} \\
(C3-1-1) &\sqsubseteq \langle \text{ass, func} \rangle \\
&\quad b := doCheckword(w, index + 1); \\
(C3-1-2) &\sqsubseteq \langle \text{Ass and } POPWORD(w, index + 1) \notin \mathbf{dom}(t) \Rightarrow b = \text{FALSE} \rangle \\
&\quad b := \text{FALSE} \\
(C3-2) &\sqsubseteq \langle index \geq |w| \wedge index \leq |w| \Rightarrow index = |w| \text{ and definition of } post(C2) \rangle \\
&\quad b := (t(w) = 1)
\end{aligned}$$

3.5 delword

From the spec we have:

$$\begin{aligned}
&\mathbf{proc} \, delword^{DictA}(\mathbf{value} \, w) \\
&\quad \perp b, t : [\text{TRUE}, b = b_0 \wedge (w \notin \mathbf{dom}(t) \vee t = t_0 : w \mapsto 0)] \neg(D1)
\end{aligned}$$

$$\begin{aligned}
(D1) &\sqsubseteq \langle \text{c-frame} \rangle \\
&\quad t : [\text{TRUE}, (w \notin \mathbf{dom}(t) \vee t = t_0 : w \mapsto 0)] \\
&\sqsubseteq \langle \text{proc}, 0 \leq |w| \text{ and } \epsilon \in \mathbf{dom}(t) \text{ since } init^{DictA} \rangle \\
&\quad doDelword(w, 0);
\end{aligned}$$

3.5.1 Definition of doDelword

$$\begin{aligned}
&\mathbf{proc} \, doDelword^{DictA}(\mathbf{value} \, w, \mathbf{value} \, index) \\
&\quad \perp t : [\text{TRUE}, (w \notin \mathbf{dom}(t) \vee t = t_0 : w \mapsto 0)] \neg(D2)
\end{aligned}$$

3.5.2 Refinement of the procedure in doDelword

$$\begin{aligned}
(D2) &\sqsubseteq \langle \text{if} \rangle \\
&\quad \text{if } index < |w| \\
&\quad \text{then } \sqcup b : [index < |w| \wedge pre(D2), post(D2)] \neg(D3-1) \\
&\quad \text{else } \sqcup b : [index \geq |w| \wedge pre(D2), post(D2)] \neg(C3-2) \\
&\quad \text{fi} \\
(D3-1) &\sqsubseteq \langle \text{if} \rangle \\
&\quad \text{if } (POPWORD(w, index + 1) \in \mathbf{dom}(t)) \\
&\quad \text{then } \sqcup b : \left[\begin{array}{l} pre(D3-1) \wedge POPWORD(w, index + 1) \in \mathbf{dom}(t), \\ post(D2) \end{array} \right] \neg(D3-1-1) \\
&\quad \text{else } \sqcup b : \left[\begin{array}{l} pre(D3-1) \wedge POPWORD(w, index + 1) \notin \mathbf{dom}(t), \\ post(D2) \end{array} \right] \neg(D3-1-2) \\
&\quad \text{fi} \\
(D3-1-1) &\sqsubseteq \langle \text{ass, func} \rangle \\
&\quad doDelword(w, index + 1); \\
(D3-1-2) &\sqsubseteq \langle \text{Ass and } POPWORD(w, index + 1) \notin \mathbf{dom}(t) \Rightarrow post(D2) = \text{TRUE} \rangle \\
&\quad skip; \\
(C3-2) &\sqsubseteq \langle index \geq |w| \wedge index \leq |w| \Rightarrow index = |w| \text{ and definition of } post(D2) \rangle \\
&\quad t := t : w \mapsto 0;
\end{aligned}$$