

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 := (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 $init^{Dict}$.

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

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:

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

3 Task 3

3.1 Pre-defined function calls

Before refining the operations in *DictA* into toy language, we would like to first define some useful function calls that helps building our later refinement more close to the real c program constructions.

3.1.1 *POPWORD*

The semantic function *POPWORD* returns a substring w' of the word w starting from the first letter to the *index*'th letter of w . The definition of the function and corresponding function call is as below:

Definition of *POPWORD*

$$POPWORD(w, i) = w' \quad \text{where } w' \leq w \wedge |w'| = i$$

Definition of the function call *popWord*

```
func popWord(value w, value i) : word ·  
  var w' ·  $\sqsubseteq w' : [i \leq |w|, w' \leq w \wedge |w'| = i]; \neg(P1) \mathbf{return} w'$ 
```

Refinement of the procedure in *popWord*

$$(P1) \sqsubseteq \quad \langle |w[0..i-1]| = i \wedge w[0..i-1] \leq w \rangle \\ w' := w[0..i-1]$$

This function helps us to build our refinement as close to the trie construction in c programs as possible. (In c programs, words are splitted into prefixes in increasing length and checked in the trie recursively.)

3.1.2 *doAddword*

Using the previous definition of *POPWORD*, we would like to develop a corresponding function call named *doAddword*(**var** w, **var** index) that allows using a variable i to locate which prefix of w (or say which sub-trie of t) the procedure currently is checking at during the entire word-adding operation.

All the prefixes of w should exist in t eventually, so if the current prefix *POPWORD*(w, i) does not exist, *doAddword* will add the new prefix into t and continue searching for the next prefix *POPWORD*($w, i+1$). Considering this and the fact that t always initializes with ϵ included, it is guaranteed that every *doAddword* operation at i level will already have all prefixes of w with length no longer than i exist in t , which satisfies the precondition of the function itself. The definition of the function and the refinement of its linking procedure is as follows.

Definition of doAddword

proc $doAddword^{DictA}(\text{value } w, \text{value } index)$
 $\sqsubseteq t : \left[\begin{array}{l} 0 \leq index \leq |w| \wedge \forall w' \leq w \wedge |w'| \leq index (w' \in \mathbf{dom}(t)), \\ t = t_0 \setminus \{w \mapsto 0\} \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \mathbf{dom}(t) | w' \mapsto 0\} \end{array} \right] \neg(A2)$

Refinement of the procedure in doAddword

$(A2) \sqsubseteq \langle \text{if} \rangle$
if $index < |w|$
then $\sqsubseteq t : [index < |w| \wedge pre(A2), post(A2)] \neg(A3-1)$
else $\sqsubseteq t : [index \geq |w| \wedge pre(A2), post(A2)] \neg(A3-2)$
fi
 $(A3-1) \sqsubseteq \langle \text{seq} \rangle$
 $\sqsubseteq t : \left[\begin{array}{l} index \geq |w| \wedge pre(A2), \\ index \geq |w| \wedge pre(A2) \wedge POPWORD(w, index + 1) \in \mathbf{dom}(t) \end{array} \right] \neg(A3-3)$
 $\sqsubseteq t : \left[\begin{array}{l} index \geq |w| \wedge pre(A2) \wedge POPWORD(w, index + 1) \in \mathbf{dom}(t), \\ post(A2) \end{array} \right] \neg(A3-4)$
 $(A3-3) \sqsubseteq \langle \text{if} \rangle$
if $(POPWORD(w, index + 1) \in \mathbf{dom}(t))$
then $\sqsubseteq t : \left[\begin{array}{l} pre(A3-3) \wedge POPWORD(w, index + 1) \in \mathbf{dom}(t), \\ post(A3-3) \end{array} \right] \neg(A3-3-1)$
else $\sqsubseteq t : \left[\begin{array}{l} pre(A3-3) \wedge POPWORD(w, index + 1) \notin \mathbf{dom}(t), \\ post(A3-3) \end{array} \right] \neg(A3-3-2)$
fi
 $(A3-3-1) \sqsubseteq \langle POPWORD(w, index + 1) \in \mathbf{dom}(t) \Rightarrow post(A3-3) = \text{TRUE} \rangle$
 $\text{skip};$
 $(A3-3-2) \sqsubseteq \langle \text{Ass} \rangle$
 $t := t \cup \{popWord(w, index) \mapsto 0\}$
 $(A3-4) \sqsubseteq \langle \text{proc} \rangle$
 $doAddword(w, index + 1);$
 $(A3-2) \sqsubseteq \langle index \geq |w| \wedge index \leq |w| \Rightarrow index = |w| \text{ and definition of } post(A2) \rangle$
 $t := t_0 \setminus \{w \mapsto 0\} \cup \{w \mapsto 1\};$

We gather the code for the body of *doAddword*

```

if  $index < |w|$ 
then
  if  $POPWORD(w, index + 1) \in \mathbf{dom}(t)$ 
  then skip;
  else
     $t := t \cup \{popWord(w, index) \mapsto 0\}$ 
  fi
  doAddword( $w, index + 1$ );
else
   $t := t_0 \setminus \{w \mapsto 0\} \cup \{w \mapsto 1\}$ ;
fi

```

3.1.3 *doCheckword*

Similarly to *doAddword*, we would also develop a corresponding function call named *doCheckword*(**var** w , **var** $index$) which will search the complete word w in t based on the precondition that all prefixes of w with length no longer than $index$ are guaranteed to exist in t . The definition of the function and the refinement of its linking procedure is as follows.

Definition of *doCheckword*

```

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

Refinement of the procedure in doCheckword

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

3.1.4 doDelword

Similarly to *doCheckword*, we would also develop a corresponding function call named *doDelword*(**var** *w*, **var** *index*) which will set the result of *t(w)* as 0 based on the precondition that all prefixes of *w* with length no longer than *index* are guaranteed to exist in *t*. The definition of the function and the refinement of its linking procedure is as follows.

Definition of doDelword

$$\begin{aligned}
&\mathbf{proc} \ doDelword^{DictA}(\mathbf{value} \ w, \mathbf{value} \ index) \\
&\quad \perp \quad t : [0 \leq index \leq |w| \wedge \forall w' \leq w \wedge |w'| \leq index \ (w' \in \mathbf{dom}(t)), \neg(D2) \\
&\quad \quad (w \notin \mathbf{dom}(t) \vee t = t_0 : w \mapsto 0)]
\end{aligned}$$

Refinement of the procedure in `doDelword`

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

3.2 Refinement of `init`

From the spec we have:

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

3.3 Refinement of `addword`

From the spec¹ we have:

$$\begin{aligned}
&\mathbf{proc } addword^{DictA}(\mathbf{value } w). \\
&\quad \perp b, t : [\text{TRUE}, b = b_0 \wedge t = t_0 \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \mathbf{dom}(t) | w' \mapsto 0\}] \neg(A1) \\
(A1) &\sqsubseteq \langle \text{c-frame} \rangle \\
&\quad t : [\text{TRUE}, t = t_0 \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \mathbf{dom}(t) | w' \mapsto 0\}] \\
&\sqsubseteq \langle \text{proc, } 0 \leq |w| \text{ and } \epsilon \in \mathbf{dom}(t) \text{ since } init^{DictA} \rangle \\
&\quad doAddword(w, 0)
\end{aligned}$$

¹Definition of this is in the Assignment 3 requirements of cs2111.

3.4 Refinement of *checkword*

From the spec we have:

func $checkword^{DictA}(\mathbf{value} \ w) : \mathbb{B}.$
var $b \cdot \sqcup b, t : [\text{TRUE}, b = (\forall w' \leq w \ (w' \in \mathbf{dom}(t)) \wedge t(w) = 1)]; \sqcup_{(C1)} \mathbf{return} \ b$

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

3.5 Refinement of *delword*

From the spec we have:

proc $delword^{DictA}(\mathbf{value} \ w)$
 $\sqcup \cdot b, t : [\text{TRUE}, b = b_0 \wedge (w \notin \mathbf{dom}(t) \vee t := t : w \mapsto 0)] \sqcup_{(D1)}$

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