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:

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

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).

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

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:

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

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:

$$
\begin{aligned}
& pre_{checkword^{Dict}}[^{f(t_0)}/_{W}] \wedge post_{checkword^{DictA}} \\
\Leftrightarrow \quad & \langle \text{Definition of } checkword^{DictA} \text{ and } checkword^{Dict} \rangle \\
& \text{TRUE}[^{f(t_0)}/_{W}] \wedge t = t_0 \wedge b = (\forall w' \leq w\,(w' \in \mathbf{dom}(t)) \wedge t(w) = 1) \\
\Rightarrow \quad & \langle \text{Definition of } f \rangle \\
& b = (w \in f(t) \wedge t(w) = 1) \wedge t = t_0 \\
\Leftrightarrow \quad & \langle \text{Definition of } checkword^{DictA} \text{ and } checkword^{Dict} \rangle \\
& post_{checkword^{Dict}}[^{f(t_0), f(t)}/_{W_0, W}]
\end{aligned}
$$

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

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 $POPWORD$

$$POPWORD(w, i) = w' \qquad where\ w' \leq w \wedge |w'| = i$$

Definition of the function call $popWord$

$$\textbf{func}\ popWord(\textbf{value}\ w, \textbf{value}\ i) : \textbf{word}\ \cdot$$
$$\textbf{var}\ w' \cdot \llcorner w' : [i \leq |w|, w' \leq w \wedge |w'| = i];\lrcorner_{(P1)}\textbf{return}\ w'$$

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 $doAddword(\textbf{var}\ w, \textbf{var}\ index)$ 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

$$\textbf{proc } \mathit{doAddword}^{DictA}(\textbf{value } w, \textbf{value } \mathit{index})$$

$$\llcorner t : \left[\begin{array}{l} 0 \leq \mathit{index} \leq |w| \wedge \forall w' \leq w \wedge |w'| \leq \mathit{index}\,(w' \in \mathbf{dom}(t))\,, \\ t = t_0 \backslash \{w \mapsto 0\} \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \mathbf{dom}(t) | w' \mapsto 0\}] \end{array} \right]_{\lrcorner(A2)}$$

Refinement of the procedure in doAddword

$(A2) \sqsubseteq \quad \langle \text{if} \rangle$

$\textbf{if } \mathit{index} < |w|$

$\textbf{then} \llcorner t : [\mathit{index} < |w| \wedge \mathit{pre}(A2), \mathit{post}(A2)]_{\lrcorner(A3-1)}$

$\textbf{else} \llcorner t : [\mathit{index} \geq |w| \wedge \mathit{pre}(A2), \mathit{post}(A2)]_{\lrcorner(A3-2)}$

$\textbf{fi}$

$(A3-1) \sqsubseteq \quad \langle \text{seq} \rangle$

$$\llcorner t : \left[\begin{array}{l} \mathit{index} \geq |w| \wedge \mathit{pre}(A2), \\ \mathit{index} \geq |w| \wedge \mathit{pre}(A2) \wedge \mathit{POPWORD}(w, \mathit{index}+1) \in \mathbf{dom}(t) \end{array} \right]_{\lrcorner(A3-3)}$$

$$\llcorner t : \left[\begin{array}{l} \mathit{index} \geq |w| \wedge \mathit{pre}(A2) \wedge \mathit{POPWORD}(w, \mathit{index}+1) \in \mathbf{dom}(t), \\ \mathit{post}(A2) \end{array} \right]_{\lrcorner(A3-4)}$$

$(A3-3) \sqsubseteq \quad \langle \text{if} \rangle$

$\textbf{if } (\mathit{POPWORD}(w, \mathit{index}+1) \in \mathbf{dom}(t))$

$$\textbf{then } \llcorner t : \left[\begin{array}{l} \mathit{pre}(A3-3) \wedge \mathit{POPWORD}(w, \mathit{index}+1) \in \mathbf{dom}(t), \\ \mathit{post}(A3-3) \end{array} \right]_{\lrcorner(A3-3-1)}$$

$$\textbf{else } \llcorner t : \left[\begin{array}{l} \mathit{pre}(A3-3) \wedge \mathit{POPWORD}(w, \mathit{index}+1) \notin \mathbf{dom}(t), \\ \mathit{post}(A3-3) \end{array} \right]_{\lrcorner(A3-3-2)}$$

$\textbf{fi}$

$(A3-3-1) \sqsubseteq \quad \langle \mathit{POPWORD}(w, \mathit{index}+1) \in \mathbf{dom}(t) \Rightarrow \mathit{post}(A3-3) = \text{TRUE} \rangle$

$\mathit{skip};$

$(A3-3-2) \sqsubseteq \quad \langle \text{Ass} \rangle$

$t := t \cup \{\mathit{popWord}(w, \mathit{index}) \mapsto 0\}$

$(A3-4) \sqsubseteq \quad \langle \text{proc} \rangle$

$\mathit{doAddword}(w, \mathit{index}+1);$

$(A3-2) \sqsubseteq \quad \langle \mathit{index} \geq |w| \wedge \mathit{index} \leq |w| \Rightarrow \mathit{index} = |w| \text{ and definition of } \mathit{post}(A2) \rangle$

$t := t_0 \backslash \{w \mapsto 0\} \cup \{w \mapsto 1\};$

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**
> $t := t_0 \backslash \{w \mapsto 0\} \cup \{w \mapsto 1\}$;
> **fi**

3.1.3 *doCheckword*

Similarly to *doAddword*, we would also develop a corresponding function call named $doCheckword(\mathbf{var}\ w, \mathbf{var}\ index)$ 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

$$\mathbf{func}\ doCheckword(\mathbf{value}\ w, \mathbf{value}\ index : \mathbb{N}) : \mathbb{B}\cdot$$

$$\mathbf{var}\ b \cdot \llcorner b : \left[\begin{array}{l} 0 \leq index \leq |w| \wedge \\ \forall w' \leq w \wedge |w'| \leq index\ (w' \in \mathbf{dom}(t))\,, \\ b = (\forall w' \leq w\ (w' \in \mathbf{dom}(t)) \wedge t(w) = 1) \end{array}\right] \lrcorner_{(C2)}; \mathbf{return}\ b$$

Refinement of the procedure in doCheckword

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

We gather the code of *doCheckword* below:

$$
\begin{aligned}
& \textbf{if } \ \mathit{index} < |w| \\
& \textbf{then} \\
& \qquad \textbf{if } \ \mathit{POPWORD}(w, \mathit{index}+1) \in \textbf{dom}(t) \\
& \qquad \textbf{then } \mathit{skip}; \\
& \qquad \textbf{else } \mathit{doCheckword}(w, \mathit{index}+1); \\
& \qquad \textbf{fi} \\
& \textbf{else } b := (t(w) = 1); \\
& \textbf{fi}
\end{aligned}
$$

3.1.4 *doDelword*

Similarly to *doCheckword*, we would also develop a corresponding function call named $\mathit{doDelword}(\textbf{var } w, \textbf{var } \mathit{index})$ 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

$$\textbf{proc } doDelword^{DictA}(\textbf{value } w, \textbf{value } index)$$
$$\llcorner t : \left[\begin{array}{l} 0 \leq index \leq |w| \wedge \forall w' \leq w \wedge |w'| \leq index \, (w' \in \textbf{dom}(t)) \, , \\ (w \notin \textbf{dom}(t) \vee t = t_0 : w \mapsto 0) \end{array} \right]_{\lrcorner(D2)}$$

Refinement of the procedure in doDelword

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

We gather the code of *doDelword* below:

if $index < |w|$
then
  if $POPWORD(w, index+1) \in \textbf{dom}(t)$
  then $doDelword(w, index+1)$;
  else *skip*;
  fielse $t := t : w \mapsto 0$;
fi

3.2 Refinement of *init*

From the spec we have:

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

3.3 Refinement of *addword*

From the spec[1] we have:

$$
\begin{array}{l}
\textbf{proc } \mathit{addword}^{\mathit{DictA}}(\textbf{value } w)\cdot \\
\quad \llcorner b, t : [\text{TRUE}, b = b_0 \wedge t = t_0 \cup \{w \mapsto 1\} \cup \{w' < w \wedge w' \notin \mathbf{dom}(t) | w' \mapsto 0\}]\lrcorner_{(A1)}
\end{array}
$$

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

3.4 Refinement of *checkword*

From the spec we have:

$$
\begin{array}{l}
\textbf{func } \mathit{checkword}^{\mathit{DictA}}(\textbf{value } w) : \mathbb{B}\cdot \\
\quad \textbf{var } b \cdot \llcorner b, t : [\text{TRUE}, b = (\forall w' \leq w\, (w' \in \mathbf{dom}(t)) \wedge t(w) = 1)];\lrcorner_{(C1)} \textbf{ return } b
\end{array}
$$

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

3.5 Refinement of *delword*

From the spec we have:

$$
\begin{array}{l}
\textbf{proc } \mathit{delword}^{\mathit{DictA}}(\textbf{value } w) \\
\quad \llcorner \cdot b, t : [\text{TRUE}, b = b_0 \wedge (w \notin \mathbf{dom}(t) \vee t := t : w \mapsto 0)]\lrcorner_{(D1)}
\end{array}
$$

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

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