

Assignment 3 - Emirps

Rahil Agrawal z5165505

Aditya Karia z5163287

COMP2111 18s1

1 Task 1 -Syntactic Data Type Dict

We define a syntactic data type **Dict** that encapsulates a dictionary set W as follows :

$$\mathbf{Dict} = (init^{Dict}, (W, x : [pre_j^{Dict}, post_j^{Dict}]_{j \in J}))$$

which consists of an initialisation predicate $init^{Dict} = (W = \{\})$ and the following operations:

proc addword^{Dict}(value w) . $W : [True, W = W_0 \cup w]$

proc checkword^{Dict}(value W , value w , result b) . $b : [True, b = (W = \{w\})]$

proc delword^{Dict}(value w) . $W : [W \neq \{w\}, W = W_0 \setminus w]$

2 Task 2 - Refinement to DictA

We're refining this to a data type DictA where we replace W with a Trie t . From Ass3 2018 S1 Specification, "A *trie domain* is a prefixclosed finite subset of L^* . A *trie* is a function from a trie domain to Booleans. Given a trie t we write **domt** for its trie domain. Let T be the set of all tries."

The correspondance between the two data types is captured by the function $f : T \mapsto P(L^*)$ given by :

$$f(t) = \{w \in L^* \mid w \in \mathbf{domt} \wedge t(w) = 1\}$$

Also, from Ass3 2018 S1 Specification, "Formally, we write $v \leq w$ if word $v \in L^*$ is a prefix of $w \in L^*$, i.e., $\exists v' \in L^* (vv' = w)$. We write B for 0, 1 where 0 represents false and 1 true."

We define our With the aforementioned facts in mind, we define a concrete syntactic data type **DictA** that encapsulates a trie t as follows :

$$\mathbf{DictA} = (init^{DictA}, (t, x : [pre_j^{DictA}, post_j^{DictA}]_{j \in J}))$$

which consists of an initialisation predicate $init^{DictA} = (t = \{\})$ and the following operations:

$proc\ addword^{DictA}(\text{value } w) . t : [True, post(addword^{DictA})]$ where

$$\begin{aligned} post(addword^{DictA}) = \mathbf{dom}t &= \mathbf{dom}t_0 \cup \{v \in L^* \mid v \leq w\} \wedge t = t_0 \cup \\ &\{v \in L^* \mid v \in \mathbf{dom}t \wedge v < w \wedge t_0(v) = 1 \Rightarrow t(v) = 1 \\ &\quad v < w \wedge t_0(v) = 0 \Rightarrow t(v) = 0 \\ &\quad v = w \Rightarrow t(v) = 1\} \end{aligned}$$

$proc\ checkword^{DictA}(\text{value } t, \text{value } w, \text{result } b) . b : [True, b = (t(w) = 1)]$

$proc\ delword^{DictA}(\text{value } w) . t : [t \neq \{\}, t(w) = 0]$

That this indeed is a refinement requires checking the relevant proof obligations:

$$init^{DictA} \Rightarrow init^{Dict}[f(t)/w] \tag{1}$$

$$pre_j^{Dict}[f(t)/w] \Rightarrow pre_j^{DictA}, \text{ for } j \in J \tag{2}$$

$$pre_j^{Dict}[f(t_0), x_0/w, x] \wedge post_j^{DictA} \Rightarrow post_j^{Dict}[f(t_0), f(t)/w_0, w], \text{ for } j \in J \tag{3}$$

We begin with (1).

$$\begin{aligned} &init^{DictA}[f(t_0), x_0/w, x] = f(t) \neq \{\} \\ \Rightarrow &\langle \text{def. of } f \rangle \\ &f(t) = \{\} \Rightarrow \\ \Rightarrow &\langle \text{def. of } Dict \rangle \\ &init^{Dict}[f(t)/w] \end{aligned}$$

Condition (2) is only required to be proven when the concrete pre-condition is non-trivial (not True). This is only the case in $delword$.

$$\begin{aligned} &pre_{delword}^{Dict}[f(t_0), x_0/w, x] = f(t) \neq \{\} \\ \Rightarrow &\langle \text{def. of } f \rangle \\ &w \in \mathbf{dom}t \wedge t(w) = 1 \\ \Rightarrow &\langle \text{def. of } trie \rangle \\ &t \neq \{\} \\ \Rightarrow &\langle \text{def. of } pre_{delword}^{DictA} \rangle \\ &pre_{delword}^{DictA} \end{aligned}$$

Finally, condition (4) needs to be checked for all operations.

For addword, we prove

$$\begin{aligned}
& \textcolor{red}{pre}_{\text{addword}}^{Dict}[f(t)/w] \wedge \textcolor{blue}{post}_{\text{addword}}^{DictA} = True \wedge \mathbf{dom}t = \mathbf{dom}t_0 \cup \{v \in L^* \mid v \leq w\} \wedge t = t_0 \cup \\
& \quad \{v \in L^* \mid v \in \mathbf{dom}t \wedge v < w \wedge t_0(v) = 1 \Rightarrow t(v) = 1 \\
& \quad \quad v < w \wedge t_0(v) = 0 \Rightarrow t(v) = 0 \\
& \quad \quad v = w \Rightarrow t(v) = 1\} \\
& \Rightarrow \langle \text{Trie is the same as old trie but we added the prefixes of } w \rangle \\
& \quad \langle \text{without changing their old mappings} \rangle \\
& \quad \langle \text{New prefixes are mapped to 0 and } w \text{ is mapped to 1.} \rangle \\
& \quad \langle \text{This means we added a word } w \text{ if it did not already exist, def. of trie and } f \rangle \\
& \quad f(t) = f(t_0) \cup w \\
& \Rightarrow \langle \text{Definition of } \text{addword}^{Dict} \rangle \\
& \quad \textcolor{red}{post}_{\text{addword}^{Dict}}[f(t_0), f(t)/w_0, w]
\end{aligned}$$

For checkword, we prove

$$\begin{aligned}
& \textcolor{red}{pre}_{\text{checkword}}^{Dict}[f(t)/w] \wedge \textcolor{blue}{post}_{\text{checkword}}^{DictA} = True \wedge b = (t(w) = 1) \\
& \Rightarrow \langle b \text{ is 1 if } w \text{ is in } \mathbf{dom}t \text{ and trie maps } w \text{ to 1, 0 otherwise. def. of trie.} \rangle \\
& \Rightarrow \langle b = 1 \text{ means } w \text{ is in our set of words. def of } f \rangle \\
& \quad b = (w \in f(t)) \\
& \Rightarrow \langle \text{def. of } \text{checkword}^{Dict} \rangle \\
& \quad \textcolor{red}{post}_{\text{checkword}^{Dict}}[f(t_0), f(t)/w_0, w]
\end{aligned}$$

For delword, we prove

$$\begin{aligned}
& \textcolor{red}{pre}_{\text{delword}}^{Dict}[f(t)/w] \wedge \textcolor{blue}{post}_{\text{delword}}^{DictA} = W \neq \{\} \wedge b = (t(w) = 0) \\
& \Rightarrow \langle \text{def of } f \rangle \\
& \quad w \notin f(t) \\
& \Rightarrow \langle \text{Clearly} \rangle \\
& \quad f(t) = f(t_0) \setminus w \\
& \Rightarrow \langle \text{def. of } \text{delword}^{Dict} \rangle \\
& \quad \textcolor{red}{post}_{\text{delword}^{Dict}}[f(t_0), f(t)/w_0, w]
\end{aligned}$$

3 Task 3 - Derivation

```

proc FunctionName(value n, result r) ·  $\sqsubseteq n, r, x : [Pre, Post] \dashv(1)$ 
(1)  $\sqsubseteq$      $\langle \text{Rule} \rangle$ 
 $\sqsubseteq r, x : [NewPre, NewPost] \dashv(2)$ 
(2)  $\sqsubseteq$      $\langle \text{seq} \rangle$ 
 $\sqsubseteq i, x, r : [Pre, Blah] \dashv(3);$ 
 $\sqsubseteq i, x : [Blah, Post] \dashv(4)$ 
 $\sqsubseteq$      $\langle \text{ass} - (1) \rangle$ 
i := 1
x := 13
 $\sqsubseteq$      $\langle \text{seq} \rangle$ 
 $\sqsubseteq s, x : [pre(16), s = 0 \wedge x > 0] \dashv(18);$ 
 $\sqsubseteq a, i, r, s, x : [x > 0 \wedge s = 0, post(16)] \dashv(19)$ 
(5)  $\sqsubseteq$      $\langle \text{while} \rangle$ 
while j  $\neq$  r do
     $\sqsubseteq r, j : [Inv_2 \wedge j \neq r, Inv_2] \dashv(8)$ 
od;
(9)  $\sqsubseteq$      $\langle \text{if} \rangle$ 
if Gaurd
then  $\sqsubseteq a : [Gaurd \wedge pre(9), post(9)] \dashv(11)$ 
else  $\sqsubseteq a : [\text{Not } Gaurd \wedge pre(9), post(9)] \dashv(12)$ 
fi;

```

We gather the code for the procedure body of `blah`:

```
blah(r, n) :  
  var i := 1;  
  var x := 13;  
  r := 13;  
  while j ≠ r do  
    x := x + 1;  
    var a := 1;  
    isPrime(x, a);  
    if a = 1 then  
      var s := 0;  
      reversen(x, s);  
      var b := 1;  
      isPrime(s, b);  
      if b = 1 ∧ s ≠ x then  
        i := i + 1;  
        r := x;  
    od;
```

Also, we gather the code for the procedure body of `blah`:

```
isPrime(r, j) :  
  var j := 2;  
  while j ≠ r do  
    if (r mod j) = 0 then  
      a := 0;  
      j := j + 1;  
    od;
```

We have derived our code. However we need to prove **some** refinements.

3.1 Implication 1: $[BLAH, BLAH] \sqsubseteq BLAH$

To prove: $BLAH \Rightarrow BLAH$

Proof:

$$\begin{aligned} & LHS = BLAH \\ \Rightarrow & \langle BLAH \rangle \\ & BLAH \\ \Rightarrow & \langle BLAH \rangle \\ & BLAH \\ \Rightarrow & \langle BLAH \rangle \\ & BLAH \\ \Rightarrow & \langle BLAH \rangle \\ & BLAH \\ \Rightarrow & \langle BLAH \rangle \\ & RHS \end{aligned}$$

3.2 Implication 2: $[BLAH, BLAH] \sqsubseteq BLAH$

To prove: $BLAH \Rightarrow BLAH$

Proof:

$$\begin{aligned} & LHS = BLAH \\ \Rightarrow & \langle BLAH \rangle \\ & BLAH \\ \Rightarrow & \langle BLAH \rangle \\ & BLAH \\ \Rightarrow & \langle BLAH \rangle \\ & BLAH \\ \Rightarrow & \langle BLAH \rangle \\ & BLAH \\ \Rightarrow & \langle BLAH \rangle \\ & RHS \end{aligned}$$

4 Task 4 - C Code

```
1 #include "dict.h"  
2 #include <stdio.h>
```