Assignment 3 COMP2111 18s1

Trie Harder

Kai Engelhardt

Revision: 1.1 of Date: 2018/05/08 01:34:38

This assignment is worth 15 marks and due before the end of week 11, that is, Wednesday May 23rd, 12:59:59 local time Sydney. Assignments are done in pairs.

Problem Statement

A *dictionary* is finite set of words, where, for simplicity, a word is a finite sequence of letters over $L = \{\text{'a'}, \ldots, \text{'z'}\}$. For instance, the default American English dictionary on my machine contains 235.886 words. Spell checking a document amounts to comparing the (stems of the) words of the document with those in the set and warning about those that cannot be matched. Typical operations on a dictionary are

- *adding* a given word to the set and
- *checking* whether a given word is contained in the set.
- *deleting* a given word. (This is an interesting yet uncommon operation. The last words removed from the example dictionary were "freen" (a typo in the list), "freend" (an archaic spelling masking a common typo), and "corelation" and its derivatives.)

Consider the following sketch of a driver program that interacts with a dictionary.

```
...
var $W$ ·                                   % our dictionary set, $W \subseteq L^*$
    $W := \emptyset$                        % initialisation
    ...
    while ... do                            % command loop
        case ... of
        $\ldots W := W \cup \{w\}$          % this is addword$(w)$
        $\ldots b := (w \in W)$             % this is checkword$(w)$
        $\ldots \{w \in W\} W := W \setminus \{w\}$   % this is delword$(w)$
        esac
        ...
```

It resembles the provided file `testdict.c`. The by far most common operation on a dictionary is checking. Thus, checking needs to be fast. Running time in the order of the size of the dictionary is not acceptable. Ideally, we aim for a running time in the order of the length of the word to be checked. One suitable data structure for this application is known as a trie. A trie is a typical COMP2521 abstract data type. It's a tree in which each node represents a unique prefix of a word in the dictionary. Each node encodes (a) information about its successor nodes and (b) whether the prefix it encodes is a word in the dictionary. If you haven't done COMP2521 or equivalent yet, you may have to brush up a little on dynamic data structures. The data type declaration and the function prototypes are provided in `dict.h`.

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 $\mathbb{B}$ for $\{\mathbf{0}, \mathbf{1}\}$ where $\mathbf{0}$ represents 'false' and $\mathbf{1}$ 'true'. A *trie domain* is a prefix-closed finite subset of L^*. A *trie* is a function from a trie domain to Booleans. Given a trie t we write $\mathsf{dom}\, t$ for its trie domain. Let $\mathcal{T}$ be the set of all tries.

Example 1 $D = \{\epsilon, \text{'a'}, \text{'a''b'}, \text{'b'}, \text{'b''b'}\}$ is a trie domain and $t = \{\epsilon \mapsto \mathbf{0}, \text{'a'} \mapsto \mathbf{1}, \text{'a''b'} \mapsto \mathbf{1}, \text{'b'} \mapsto \mathbf{0}, \text{'b''b'} \mapsto \mathbf{1}\}$ is a trie with $\mathsf{dom}\, t = D$. It represents the set $\{\text{'a'}, \text{'a''b'}, \text{'b''b'}\}$ of words.

Tasks

1. Inspired by the program sketch above, describe a syntactic data type Dict with three operations: addword, checkword, and delword. The encapsulated state is a dictionary word set W.

2. Refine Dict to a second data type, DictA, in which W is replaced by a trie t. Describe the refined initialisation and operations. Prove that DictA is a data refinement of Dict.

3. Derive toy language code from the specifications of your DictA operations.
4. Translate your answers to 3 to C functions to match the prototypes given in `dict.h`. (One common difference is that you have to add a parameter to the procedures to represent a root node to indicate where you currently are in your operation. This allows for using a common recursion pattern for traversing trees (and tries) one node at a time.)
5. Describe your solutions to tasks 1–4 in a LaTeX document that your tutor enjoys reading. In more detail:
   - Argue informally how your two syntactic data types capture the requirements.
   - List, and discharge by proof, all proof obligations arising from the data refinements. (It may very well be that certain implications you have to prove for Reynolds reappear in refinement proofs.) Properly reference any outside sources you use.
   - Justify any changes made during the translation to C.

   Excessively verbose or unclear presentations will be marked down.

Deliverables

`dict.c` C source.

`dict.tex` is a LaTeX document with your names and student numbers mentioned in the `\author` command. It contains your task 5 solution.

Examples

Examples of the interaction with your submission are as follows. (Our shell prompt is $ and user input is coloured red.)

```
$ make testdict
$ ./testdict
a ant
a anti
a antifreeze
c an
'an' is not known
c anti
q byebye
$ time ./testdict < words.txt
'pseudolamellibranchiata' is not known

real 0m0.224s
```

```
user 0m0.125s
sys 0m0.088s
$
$ make dict.pdf
```

Submission Instructions

When submissions are enabled, the `give` command to be run is:

```
% 2111
% give cs2111 ass3 dict.c dict.tex
```

Do not submit any of the the provided files.

Only one member of the pair should submit. In case there a competing submission by both members of a team, only the most recent one will be marked.

```
$Log: ass3.tex,v $
Revision 1.1  2018/05/08 01:34:38  kaie
Initial revision
```