src/main.rs:

```rust
use itertools::Itertools;
use std::fs::File;
use std::io::{BufRead, BufReader};

fn main() -> Result<(), failure::Error> {
    // The /usr/share/dict/words file is provided by the wbritish and wamerican packages on Ubuntu,
    // and by the miscfiles package on Gentoo.
    let file = File::open("/usr/share/dict/words")?;
    let reader = BufReader::new(file);

    for line in reader.lines() {
        let word = line?;

        let counts = word.chars()
            // Group sequences of the same character into runs.
            .group_by(|el| *el)
            // Consume the iterator.
            .into_iter()
            // Count how often the same character occurs per run.
            .map(|(key, group)| (key, group.count()))
            // We have to collect the results in order to apply a moving window.
            .collect::<Vec<(char, usize)>>();

        if counts
            // Look at consecutive triplets.
            .windows(3)
            // Check if all of them occur twice.
            .map(|counts| counts.iter().all(|(_, count)| *count == 2))
            // If any window has a triplet where all of them occur twice, then show the word.
            .any(|el| el) {
            println!("{}", word);
        }
    }

    Ok(())
}
```

Cargo.toml:

```toml
[package]
name = "bookkeeper"
version = "0.1.0"
authors = ["S.J.R. van Schaik <stephan@synkhronix.com>"]
edition = "2018"

[dependencies]
failure = "0.1"
itertools = "0.10"
```

Peer Review ID: 62040661 — enter this when you fill out your peer evaluation via gradescope

2 Set Theory [5 points]

Let X and Y be sets. Let $\mathcal{P}(X)$ denote the power set of X, i.e. the set of all subsets of X. There is a one-to-one correspondence, i.e. a bijection, between the sets A and B, where $A = X \to \mathcal{P}(Y)$ and $B = \mathcal{P}(X \times Y)$. Note that A is a set of functions, and that B can be viewed as a set of relations.

Demonstrate the correspondence between A and B by presenting an appropriate function and proving that it is a bijection. For example, you might construct a function $f : B \to A$ and prove that f is an injection and a surjection.

Let us construct a function $f : A \to B$ and prove that it is bijective. We can either prove this by: a) proving that the function f is both *injective* and *surjective* b) proving that the function f is invertible. More specifically, the type of the function f is $f : (X \to \mathcal{P}(Y)) \to \mathcal{P}(X \times Y)$. We choose f as follows:

$$f(a) \coloneqq \{(x, y) \mid y \in a(x)\}$$

a) **Injectivity**:

Let f be a function whose domain is a set A. The function f is said to be *injective* provided that for all a and b in A, whenever $f(a) = f(b)$, then $a = b$. That is, $f(a) = f(b)$ implies $a = b$.

Symbolically, we can express the above as: $\forall a, b \in A, f(a) = f(b) \Rightarrow a = b$

More specifically, our function f is *injective* if for all $a_1 \in A$ and $a_2 \in A$, whenever $f(a_1) = f(a_2)$, then $a_1 = a_2$. Let a_1 and a_2 be arbitrary elements of the set A, and assume $f(a_1) = f(a_2)$. Then, by definition of f:

$$\{(x, y) \mid y \in a_1(x)\} = \{(x, y) \mid y \in a_2(x)\}$$

We now proceed to use the *axiom of extensionality* in set theory, which states that two sets are equal if they have exactly the same elements. If we apply this axiom to the two aforementioned sets, then we find that for any (x, y), whenever $y \in a_1(x)$, we also have $y \in a_2(x)$. By applying the *axiom of extensionality* to $a_1(x)$ and $a_2(x)$, we find that they must be equal sets. This is because for all y, they either both contain that same y, or they both do not contain that same y. Therefore, for any x, $a_1(x) = a_2(x)$. Thus, as both functions agree on all arguments and by the definition of *function*, both a_1 and a_2 are equal functions. Therefore f is *injective*.

Surjectivity:

A function is said to be *surjective* provided that its image is equal to its codomain. Equivalently, a function f with domain A and codomain B is *surjective*, if for every b in B there exists at least one a in A with $f(a) = b$.

Symbolically, we can write this as follows: consider the function $f : A \to B$, then f is said to be surjective if: $\forall b \in B, \exists a \in A, f(a) = b$

Let b be an arbitrary element of B. Therefore, by the definition of B above, $b \in \mathcal{P}(X \times Y)$ where every element of b is of the form (x, y) with $x \in X$ and $y \in Y$. We now construct an a, such that $f(a) = b$. By defintion of f, $f(a) = \{(x, y) \mid y \in a(x)\}$. Thus, we pick our function a by letting $a(x) = \{(x, y) \mid (x, y) \in b\}$. Through substitution, we then find that $f(a) = \{(x, y) \mid y \in \{y \mid (y \mid (x, y) \in b\}\}$ simplifies to $f(a) = \{(x, y) \mid (x, y) \in b\}$. Since $f(a)$ is the set of elements that are exactly those elements found in b, by the *axiom of extensionality*, $f(a) = b$. Therefore the function f is *surjective*.

Bijectivity:

As we have proved that f is both *injective* and *surjective*, it is also *bijective* or invertible. Since there exists an invertible function $f : A \to B$, there is a one-to-one correspondence between A and B. Q.E.D.

Peer Review ID: 62040661 — enter this when you fill out your peer evaluation via gradescope

b) **Inverse**:

Let f be as in the previous solution:

$$f(a) \coloneqq \{(x, y) \mid y \in a(x)\} \tag{1}$$

We introduce a second function, g, that we will show to be the inverse of f. Since $g : B \to A$, and A is a set of functions, every $g(b)$ will be a function. We define g as follows:

$$(g(b))(x) \coloneqq \{y \mid (x, y) \in b\} \tag{2}$$

That is, $g(b)$ returns a function, which when it is presented with the argument x, returns the set $\{y \mid (x, y) \in b\}$.

By the definition of *invertible*, by showing that $g \circ f$, i.e. g composed with f or $g(f(x))$, is the identity function: $(g \circ f)(a) = a$, we can show that f and g are each other's inverses. Let a be an arbitrary element of A, such that a is a function $a : X \to \mathcal{P}(Y)$, that is a function mapping X to $\mathcal{P}(Y)$. To show that $(g \circ f)(a) = a$, we will show that they behave the same way on all inputs as they are both functions: $((g \circ f)(a))(x) = a(x)$.

Now we expand $(f \circ g)(x)$ by definition of $f(a)$ [eq. (1)]:

$$((g \circ f)(a))(x) = g(\{(x, y) \mid y \in a(x)\})(x)$$

Now we expand by definition of $(g(b))(x)$ [eq. (2)]:

$$((g \circ f)(a))(x) = \{y \mid (x, y) \in \{(x, y) \mid y \in a(x)\}\}$$

Through simplification, we have:

$$((g \circ f)(a))(x) = \{y \mid y \in a(x)\}$$

By the *axiom of extensionality*, we find that the set of all elements found in $a(x)$ is exactly $a(x)$ itself. Thus, we have:

$$((g \circ f)(a))(x) = a(x)$$

As $g \circ f$ is indeed the identity function, f and g must be each other's inverses. Therefore $f : A \to B$ is *intervertible*, which means that there is a one-to-one correspondence between A and B. Q.E.D.

Peer Review ID: 62040661 — enter this when you fill out your peer evaluation via gradescope

3 Model Checking [10 points]

Download the CPAChecker software model-checking tool using the instructions on the homework webpage. Read through enough of the manual to run the tool on the `tcas.i` testcases provided on the homework webpage. Check the three proprerties given. For each command, copy or screenshot the last ten non-empty lines (or all of the lines you have, if you have fewer than ten) of (standard, terminal) output from CPAChecker and include them as part of your answer to this question.

For this part of the assignment, we will be looking at a Traffic Collision Avoidance System or a Traffic Alert and Collision Avoidance System (TCAS). The goal of such a system is to reduce the chance of mid-air collision between aircraft. Therefore, it monitors the airspace around an aircraft for other aircraft equipped with a corresponding transponder, independent of air traffic control, and warns pilots of the presence of other transponder-equipped aircraft which may present a threat of mid-air collision.

Whenever an airplane approaches, the TCAS will first signal a traffic advisory to alert the pilot for a possible resolution advisory. For the purpose of this assignment, we only consider the regulatory advisories where the pilot has to climb or descend to avoid a possible mid-air collision. Once the other aircraft is no longer a threat, the TCAS signals a clear of conflict. The TCAS has to maintain an up and a down separation, which is how much space is available above and below of the aircraft relative to other aircraft(s) respectively.

We will now focus on lines 2118-2141 of the `tcas.i` file, where we check for the violations of certain properties when the system has to advice the pilot to either climb or descend:

```
2118        else if (need_upward_RA)
2119          {
2120
2121
2122                  property1b(alim) ;
2123                  property2b(alim) ;
2124                  property3b(alim) ;
2125                  property4b() ;
2126                  property5b() ;
2127
2128
2129              alt_sep = 1;
2130          }
2131          else if (need_downward_RA)
2132          {
2133
2134                  property1a(alim) ;
2135                  property2a(alim) ;
2136                  property3a(alim) ;
2137                  property4a() ;
2138                  property5a() ;
2139
2140              alt_sep = 2;
2141          }
```

In case the advice is to climb, the properties we care about are 1b and 2b. Property 1b is in violation when the up separation is less than a certain threshold, but when the down separation is greater than or equal to that threshold. This checks that we don't leave too little room above us, while we have a lot of room below us, in which case the TCAS should have guided us to fly a bit lower instead to maintain enough space above us. Property 1a is the inverse for when the advice is to descend. That is property 1a checks whether we don't have too little room below us, while we have a lot of space above us, in which case the TCAS should have guided us to fly a little bit higher instead.

Finally, property 2b is in violation if the up and down separation are below a certain threshold, but when the up separation is smaller than the down separation. In this situation, we already have little space above and below us, and the TCAS system would be guiding us to climb while we have less space above us than below us.

Peer Review ID: 62040661 — enter this when you fill out your peer evaluation via gradescope

The following sequence of commands check whether the properties 1a, 1b and 2b are in violation:

1. `./CPAchecker-2.0-unix/scripts/cpa.sh -predicateAnalysis -spec Property1a.spc tcas.i`:

```
Using refinement for predicate analysis with PredicateAbstractionRefinementStrategy strategy.
↪  (PredicateCPA:PredicateCPARefiner.<init>, INFO)

Starting analysis ... (CPAchecker.runAlgorithm, INFO)

Stopping analysis ... (CPAchecker.runAlgorithm, INFO)

Verification result: FALSE. Property violation (error label in line 1963) found by chosen configuration.
More details about the verification run can be found in the directory "./output".
Graphical representation included in the file "./output/Counterexample.1.html".
```

2. `./CPAchecker-2.0-unix/scripts/cpa.sh -predicateAnalysis -spec Property1b.spc tcas.i`:

```
Using refinement for predicate analysis with PredicateAbstractionRefinementStrategy strategy.
↪  (PredicateCPA:PredicateCPARefiner.<init>, INFO)

Starting analysis ... (CPAchecker.runAlgorithm, INFO)

Stopping analysis ... (CPAchecker.runAlgorithm, INFO)

Verification result: TRUE. No property violation found by chosen configuration.
More details about the verification run can be found in the directory "./output".
Graphical representation included in the file "./output/Report.html".
```

3. `./CPAchecker-2.0-unix/scripts/cpa.sh -predicateAnalysis -spec Property2b.spc tcas.i`:

```
Using refinement for predicate analysis with PredicateAbstractionRefinementStrategy strategy.
↪  (PredicateCPA:PredicateCPARefiner.<init>, INFO)

Starting analysis ... (CPAchecker.runAlgorithm, INFO)

Stopping analysis ... (CPAchecker.runAlgorithm, INFO)

Verification result: FALSE. Property violation (error label in line 1997) found by chosen configuration.
More details about the verification run can be found in the directory "./output".
Graphical representation included in the file "./output/Counterexample.1.html".
```

That is, property 1a and 2b are in violation, while property 1b is satisfied. The implications of the properties that are being violated are that for 1a) that the pilot may be told to descend while there is already little room available below the airplane, while there is plenty of space above of the airplane and that for 2b) that the pilot may be told to climb while there is little room left both above and below of the airplane, while the room left above is already than the room available below the pilot.

In general, the way these properties work is that if they are in violation, the code jumps to an error state and the CPAChecker tool checks whether that error state can be reached or not. While CPAChecker is definitely a good tool to use for software where software and system safety is paramount, as is the case with anything related to aviation, CPAChecker does put some burden on the programmer as the programmer has to check for these properties at the right places in their code as well as come up with properties for which they should test. Additionally, these properties should be narrowed down enough to the extent that they can distinguish particular scenarios.

Whether CPAChecker actually works well depends on a number of factors, such as how well does the specification and therefore the properties cover the set of problems that can occur. If we invert the problem, then one may consider using `tcas.i` as a tool to verify whether CPAChecker actually works accordingly, in which case the question becomes does the `tcas.i` test case actually cover all the use cases of CPAChecker? To be able to answer that question, we have to look at what edge cases we can intentionally and perhaps artificially implement in our test case to find issues with CPAChecker itself. However, in that case it would be beneficial if we evaluate other software model-checking tools to a) confirm that our program without those edge cases is correct, that b) our edge cases indeed do what we intended them to do and c) compare whether all those software model-checking tools come

to same conclusion, and if not, why that is not the case. In addition, it is always good to collect a wide variety of test cases instead of just `tcas.i` to verify software model-checking tools. This would not only allow to evaluate for correctness, but it would allow to evaluate different metrics such as how suitable are certain tools for the problem at hand, and how well do they perform.

An approach other than software model checking might be to come up with a paradigm where the programmer is restricted in such a way, that these properties can never be violated to begin with, and this is a very typical approach for a lot of Rust APIs. For instance, with multi-threading, you can restrict the API in such a way that locks can only be dropped/freed/unlocked once, which means that you could never violate a property of unlocking a lock more than once. Of course, this approach requires the developer to rewrite their entire software using that restricted API, but for new software implementations, this is an approach worth considering.

As there really is no clear cut solution, CPAChecker is definitely a worthwhile tool to consider when you want to achieve systems safety in avionics. However, if we invert the problem, the `tcas.i` test case on its own may not be sufficient to test tools like CPAChecker. It would instead be much more preferable to compare different tools and different test cases to evaluate such tools, and to be able to actually compare them in regards to different metrics like correctness, but also usability and performance.

1 HW0

- **0 pts** Correct