Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа программной инженерии

**Отчет**

Дисциплина: Объектно-ориентированное моделирование технических систем

Тема: Описание работы с ПО BlankFill

Выполнили:

студент гр. 5140904/40102 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Устинов И. В.

(подпись)

студент гр. 5140904/40101 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кушнарева С. В.

(подпись)

студент гр. 5140904/40101 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Магомедов Т. З.

(подпись)

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Шошмина И. В.

(подпись)

Санкт-Петербург   
2024

Содержание

[Введение 3](#_Toc191812213)

[1. Установка 4](#_Toc191812214)

[1. Установка Rust 4](#_Toc191812215)

[2. Клонирование репозитория 4](#_Toc191812216)

[3. Сборка и запуск 4](#_Toc191812217)

[2. Разбор примеров 5](#_Toc191812218)

[1. Преобразование ФИО 5](#_Toc191812219)

[2. Форматирование дат 5](#_Toc191812220)

[3. Преобразование телефонных номеров 6](#_Toc191812221)

[4. Маскировка номера телефона 7](#_Toc191812222)

[5. Форматирование номера карты 7](#_Toc191812223)

[6. Форматирование времени 8](#_Toc191812224)

[7. Постепенное наращивание строки 9](#_Toc191812225)

[8. Форматирование адреса 9](#_Toc191812226)

[9. JSON преобразование 10](#_Toc191812227)

[3. Демонстрация работы 12](#_Toc191812228)

[Заключение 13](#_Toc191812229)

# Введение

BlinkFill – это инструмент для синтеза программ, который используется для автоматического преобразования данных на основе примеров. Он позволяет пользователям автоматически генерировать правила преобразования строк без необходимости явного программирования. Данный инструмент полезен в задачах обработки текстовых данных, автоматизации форматирования и преобразования строк в различных областях, включая работу с базами данных, текстовыми файлами и табличными процессорами.

BlinkFill основан на алгоритме BlinkFill (Singh '16, VLDB), который использует методы синтеза программ для анализа входных и выходных данных и поиска наиболее вероятного правила преобразования. Подобные технологии используются в популярных инструментах, таких как Flash Fill в Microsoft Excel.

Алгоритм BlinkFill (Singh '16, VLDB) был представлен в работе Армаана Сингха в 2016 году на конференции Very Large Data Bases (VLDB). Этот метод основан на обучении по примерам: пользователь вводит несколько входных и выходных значений, а система автоматически выявляет закономерности и строит программу для преобразования новых данных.

Rust – это современный системный язык программирования, ориентированный на безопасность памяти, высокую производительность и параллельное выполнение. Его преимущества делают его идеальным для задач синтеза программ, где важна надежность.

Cargo – это менеджер пакетов и инструмент сборки для Rust. Он позволяет:

* устанавливать зависимости,
* компилировать код,
* запускать тесты и исполняемые файлы.

# Установка

Инструмент BlinkFill реализован на языке программирования **Rust**, поэтому для его работы необходимо установить соответствующую среду разработки.

## Установка Rust

Если Rust еще не установлен, выполните в терминале следующие команды для его установки:

1. Скачать и запустить установочный скрипт **rustup**, который является официальным менеджером версий для **Rust:**

curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

1. Обновить текущую среду командной строки, чтобы можно было сразу использовать cargo и rustc без перезагрузки терминала.

source $HOME/.cargo/env

Эти команды установят **Rust** и его менеджер пакетов **Cargo**.

## Клонирование репозитория

Для получения исходного кода BlinkFill необходимо выполнить команды:

1. Для клонирования проекта с GitHub:

git clone https://github.com/anishathalye/synox.git

1. Для перемещения в директорию проекта:

cd synox

## Сборка и запуск

Перед запуском нужно убедиться, что все зависимости установлены корректно. Запустить BlinkFill можно с помощью команды:

cargo run --bin blinkfill\_test

Если все установлено правильно, программа должна запуститься и выполнить тестовый пример, который находится в методе main файла main.rs.

# Разбор примеров

## Преобразование ФИО

**Задача:** преобразовать полное имя в инициалы и фамилию.

**Входные данные:**

John Doe

Alice Smith

Bob Johnson

**Выходные данные:**

J. Doe

Smith

Johnson

**Код на Rust:**

use synox::{StringProgram, blinkfill};

fn main() {

let unpaired: &[Vec<&str>] = &[];

let examples = &[(vec!["John Doe"], "J. Doe"),

(vec!["Alice Smith"], "A. Smith")];

let prog = blinkfill::learn(unpaired, examples).unwrap();

let result = prog.run(&["Bob Johnson"]).unwrap();

println!("{}", result); // Должно вывести: "B. Johnson"

}

## Форматирование дат

**Задача:** преобразовать даты из формата "ГГГГ-ММ-ДД" в "ДД.ММ.ГГГГ".

**Входные данные:**

2025-02-22

2024-12-05

2023-07-19

**Выходные данные:**

22.02.2025

05.12.2024

19.07.2023

**Пример кода на Rust:**

use synox::{StringProgram, blinkfill};

fn main() {

let unpaired: &[Vec<&str>] = &[];

let examples = &[(vec!["2025-02-22"], "22.02.2025"),

(vec!["2024-12-05"], "05.12.2024")];

let prog = blinkfill::learn(unpaired, examples).unwrap();

let result = prog.run(&["2023-07-19"]).unwrap();

println!("{}", result); // Должно вывести: "19.07.2023"

}

## Преобразование телефонных номеров

**Задача:** преобразовать номера телефонов из формата "1234567890" в "(123) 456-7890".

**Входные данные:**

1234567890

9876543210

5551234567

**Выходные данные:**

(123) 456-7890

(987) 654-3210

(555) 123-4567

**Пример кода на Rust:**

use synox::{StringProgram, blinkfill};

fn main() {

let unpaired: &[Vec<&str>] = &[];

let examples = &[(vec!["1234567890"], "(123) 456-7890"),

(vec!["9876543210"], "(987) 654-3210")];

let prog = blinkfill::learn(unpaired, examples).unwrap();

let result = prog.run(&["5551234567"]).unwrap();

println!("{}", result); // Должно вывести: "(555) 123-4567"

}

## Маскировка номера телефона

**Задача:** замаскировать 4 цифры номера телефона в символ «\*»

**Входные данные:**

89217065716

89001234567

89506781234

**Выходные данные:**

8950\*\*\*\*234

**Пример кода на Rust:**

use synox::{StringProgram, blinkfill};

fn main() {

let unpaired: &[Vec<&str>] = &[];

let examples = &[ (vec!["89217065716"], "8921\*\*\*\*716"),

(vec!["89001234567"], "8900\*\*\*\*567") ];

let prog = blinkfill::learn(unpaired, examples).unwrap();

let result = prog.run(&["89506781234"]).unwrap();

println!("{}", result); // Должно вывести:"8950\*\*\*\*234"

}

## Форматирование номера карты

**Задача:** представить номер карты в привычном формате (4 группы по 4 цифры ХХХХ ХХХХ ХХХХ ХХХХ)

**Входные данные:**

234567812345678

8765432187654321

9876543212345678

**Выходные данные:**

9876 5432 1234 5678

**Пример кода на Rust:**

use synox::{StringProgram, blinkfill};

fn main() {

let unpaired: &[Vec<&str>] = &[];

let examples = &[

(vec!["1234567812345678"], "1234 5678 1234 5678"),

(vec!["8765432187654321"], "8765 4321 8765 4321")

];

let prog = blinkfill::learn(unpaired, examples).unwrap();

let result = prog.run(&["9876543212345678"]).unwrap();

println!("{}", result); //Вывод: "9876 5432 1234 5678"

}

## Форматирование времени

**Задача:** преобразовать время в формате ЧЧ:ММ:СС

**Входные данные:**

143025

084514

143040

083015

235959

120000

101535

**Выходные данные:**

10:13:35

**Пример кода на Rust:**

use synox::{StringProgram, blinkfill};

fn main() {

let unpaired: &[Vec<&str>] = &[];

let examples = &[

(vec!["143025"], "14:30:25"),

(vec!["084514"], "08:45:14"),

(vec!["143040"], "14:30:40"),

(vec!["083015"], "08:30:15"),

(vec!["235959"], "23:59:59"),

(vec!["120000"], "12:00:00"),

];

let prog = blinkfill::learn(unpaired, examples).unwrap();

let result = prog.run(&["101535"]).unwrap();

println!("{}", result); // Вывод результата: "10:15:30"

}

## Постепенное наращивание строки

**Задача:** вывод всех префиксов входной строки заданной длины.

**Входные данные:**

hello

camel

house

**Выходные данные:**

h he hel hell hello

c ca cam came camel

h ho hou hous house

**Пример кода на Rust:**

use synox::{StringProgram, blinkfill};

fn main() {

let unpaired: &[Vec<&str>] = &[];

let examples = &[(vec!["hello"], "h he hel hell hello"),

(vec!["camel"], "c ca cam came camel")];

let prog = blinkfill::learn(unpaired, examples).unwrap();

let result = prog.run(&["house"]).unwrap();

println!("{}", result);

}

## Форматирование адреса

**Задача:** представить введенный адрес в отформатированном виде

**Входные данные:**

Russia, Moscow, Lenin, 15, 105

Germany, Berlin, Dirken, 4, 64

America, Minnesota, Wash, 21, 2

Australia, Canberra, Kang, 67, 6

**Выходные данные:**

st. Lenin, h. 15, apt. 105, Moscow, Russia

st. Dirken, h. 4, apt. 64, Berlin, Germany

st. Wash, h. 21, apt. 2, Minnesota, America

st. Kang, h. 67, apt. 6, Canberra, Australia

**Пример кода на Rust:**

use synox::{StringProgram, blinkfill};

fn main() {

let unpaired: &[Vec<&str>] = &[];

let examples = &[(vec!["Russia, Moscow, Lenin, 15, 105"], "st. Lenin, h. 15, apt. 105, Moscow, Russia"),

(vec!["Germany, Berlin, Dirken, 4, 64"], "st. Dirken, h. 4, apt. 64, Berlin, Germany"),

(vec!["America, Minnesota, Wash, 21, 2"], "st. Wash, h. 21, apt. 2, Minnesota, America")];

let prog = blinkfill::learn(unpaired, examples).unwrap();

let result = prog.run(&["Australia, Canberra, Kang, 67, 6"]).unwrap();

println!("{}", result);

}

## JSON преобразование

**Задача:** преобразовать заданные поля в JSON

**Входные данные:**

John, 30, banana

Jaden, 54, berry

Mikhail, 22, tomato

Donald, 78, cabbage

Timur, 23, kiwi

**Выходные данные:**

{ name: John, age: 30, good: banana }

{ name: Jaden, age: 54, good: berry }

{ name: Mikhail, age: 22, good: tomato }

{ name: Donald, age: 78, good: cabbage }

{ name: Timur, age: 23, good: kiwi }

**Пример кода на Rust:**

fn main() {

let unpaired: &[Vec<&str>] = &[];

let examples = &[(vec!["John, 30, cucumber"], "{ name: John, age: 30, good: cucumber }"),

(vec!["Jaden, 54, berry"], "{ name: Jaden, age: 54, good: berry }"),

(vec!["Mikhail, 22, tomato"], "{ name: Mikhail, age: 22, good: tomato }"),

(vec!["Donald, 78, cabbage"], "{ name: Donald, age: 78, good: cabbage }")];

let prog = blinkfill::learn(unpaired, examples).unwrap();

let result = prog.run(&["Timur, 23, kiwi"]).unwrap();

println!("{}", result);

}

# Демонстрация работы

Для демонстрации работы BlinkFill необходимо:

* Ввести примеры входных данных в файл main.rs.
* Запустить инструмент с помощью

cargo run --bin blinkfill\_test.

* Наблюдать, как BlinkFill автоматически анализирует примеры и формирует правила преобразования.
* Подать новые входные данные и проверить, насколько корректно инструмент применяет найденные правила.

Если все шаги выполнены корректно, программа успешно выведет преобразованные строки.

Объяснение команды cargo run --bin blinkfill\_test.

cargo run – команда для запуска проекта.

--bin blinkfill\_test – указывает, какой исполняемый файл нужно запустить.

Хотя основной код находится в main.rs, Rust использует систему пакетов Cargo, где можно определять несколько исполняемых файлов. В файле Cargo.toml указывается, что blinkfill\_test — это отдельный бинарный файл, который компилируется и исполняется отдельно от библиотеки synox.

# Заключение

BlinkFill является мощным инструментом для автоматического синтеза программ преобразования строк. Он позволяет обобщать примеры и генерировать код, который выполняет требуемые преобразования без явного программирования. Это делает его полезным инструментом для автоматизации обработки данных, особенно в случае повторяющихся задач по форматированию. В ходе экспериментов были выявлены следующие ключевые особенности:

* Гибкость. Поддерживает преобразование имен, дат, телефонных номеров и других строковых данных.
* Простота использования. Достаточно предоставить несколько примеров, и система сама построит нужную программу.
* Быстродействие. Реализован на Rust, что обеспечивает высокую производительность.
* Автоматизация. Снижает необходимость вручную писать код для повторяющихся операций обработки данных.

Таким образом, BlinkFill можно применять в бизнес-аналитике, обработке данных, автоматизированном форматировании и других сценариях, где требуется быстрое преобразование строковых данных.