Proboszcz Jakub Projekt PIPR – dokumentacja

1. Cel i opis projektu

Program pozwala użytkownikowi na symulację rejestru zbudowanego z dowolnej liczby przerzutników połączonych ze sobą.

Wejściem do przerzutnika może być dowolnie wiele wyjść przerzutników, poddanych wybranej operacji logicznej (OR/AND/XOR/NAND/NOR/NOT). Każdy przerzutnik musi mieć co najmniej jedno wejście. Przerzutnik z operacją NOT musi mieć dokładnie jedno wejście.

W ten sposób powstaje generator ciągów bitów.

Dla podanych w pliku wejściowym: danych rejestru oraz startowej listy stanów przerzutników program generuje listę ciągów bitów.

Ponadto, program oblicza stopień wykorzystania przestrzeni (ile różnych ciągów wygenerowano w stosunku do liczby możliwych różnych ciągów dla rejestru z taką liczbą przerzutników) oraz średnią różnorodność ciągów (średnią liczbę różniących się między sobą sąsiednich bitów ciągu).

Program przyjmuje argumenty pozwalające ustalić liczbę kroków do wykonania na rejestrze oraz decydujące o zapisaniu lub wyświetleniu wyników.

2. Instrukcja użytkownika

Użycie programu polega na uruchomieniu głównego pliku programu (main.py) przez intepreter Pythona (wersja: Python 3.8.10) z odpowiednimi argumentami.

Wymagane argumenty:
- SOURCE – ścieżka do pliku wejściowego (patrz: 3. Format pliku wejściowego)
- jeden z argumentów: --steps STEPS, --until-looped (patrz Opcjonalne argumenty)

Opcjonalne argumenty:
- -h, --help – program tylko wypisze informacje o przyjmowanych argumentach
- -s STEPS, --steps STEPS – ilość kroków, które program ma wykonać podczas generowania ciągów (wygenerowane zostanie STEPS+1 ciągów, wliczając ciągi startowy i końcowy) (argument wzajemnie wykluczający się z --until-looped)
- -l, --until-looped – z tą opcją program będzie generował ciągi do momentu, kiedy natrafi na ciąg, który już wcześniej wystąpił (oznacza to, że nastąpiło zapętlenie) (argument wzajemnie wykluczający się z --steps STEPS)
- -sv SAVE, --save SAVE – z tą opcją program zapisze wyniki programu do pliku wyjściowego. (patrz 4. Format pliku wyjściowego). SAVE to ścieżka do pliku wyjściowego. Jeśli podana ścieżka prowadzi do już istniejącego pliku, zostanie on nadpisany.
- -sh, --show – z tą opcją program wypisze wyniki w wyjściu standardowym. Wypisane zostaną: wygenerowane ciągi (razem z ciągiem startowym oraz końcowym), stopień wykorzystania przestrzeni w % oraz średnia różnorodność ciągów (obliczone dla wygenerowanych ciągów).

Przykładowa komenda uruchamiająca program (system: Windows 10, WSL, Ubuntu):

python3 main.py example.json -l -sv results.json -sh

Program załaduje dane rejestru z pliku example.json, będzie generował ciągi aż natrafi na pętlę, po czym zapisze wyniki do pliku results.json oraz je wypisze.

W programie użyto modułów sys, json, argparse oraz copy. Są one częścią standardowej biblioteki Pythona, która powinna być zainstalowana, żeby program działał poprawnie.

3. Format pliku wejściowego

Plik wejściowy powinien być w formacie JSON. Powinien zawierać słownik z dwoma elementami:

• "flip_flop_functions" – lista słowników.

Każdy z tych słowników reprezentuje wejście jednego z przerzutników, numerowanych od zera (pierwszy element listy opisuje przerzutnik o indeksie 0, drugi element - przerzutnik o indeksie 1 itd.).

Każdy z tych słowników musi zawierać dwa elementy:

• "operation" – napis reprezentujący operację, która ma zostać wykonana na wejściach przerzutnika. Operacja powinna być napisana w całości małymi literami. Akceptowane operacje: or, and, xor, nand, nor, not.
Dla tylko jednego wejścia operacje or, and oraz xor nie zmodyfikują wejścia. Operacje nor oraz nand zachowają się jak operacja not.
• "input_indexes" – lista liczb całkowitych nieujemnych. Jest to lista indeksów przerzutników, których wyjścia mają być podłączone do wejścia opisywanego przerzutnika. Należy zwrócić uwagę na to, że przerzutniki indeksowane są od zera.
Ta lista powinna zawsze zawierać co najmniej jeden element.
Dla operacji not ta lista powinna zawierać dokładnie jeden element.

• "starting_state" – lista liczb całkowitych 0 i 1. Reprezentuje ona początkowy stan poszczególnych przerzutników rejestru. 0 oznacza stan niski, 1 stan wysoki. Pierwszy element listy to stan przerzutnika o indeksie 0, drugi element listy to stan przerzutnika o indeksie 1 itd. Lista ta musi mieć tyle elementów ile rejestr ma przerzutników.

Przykładowy plik wejściowy to example.json w głównym katalogu projektu.

4. Format pliku wyjściowego

Plik wyjściowy jest w formacie JSON. Zawiera słownik z trzema elementami:

• "sequences" – lista list liczb całkowitych 0 i 1. Jest to lista ciągów wygenerowanych przez program. Zawiera startowy, jak i końcowy stan rejestru.

• "space_usage" – liczba rzeczywista. Jest to stopień wykorzystania przestrzeni, obliczony dla wygenerowanych ciągów.

• "average_sequence_diversity" – liczba rzeczywista. Jest to średnia różnorodność ciągu, obliczona dla wygenerowanych ciągów.

Przykładowy plik wyjściowy to results.json w głównym katalogu projektu.

5. Struktura programu

Program złożony jest z trzech klas:

• Logic_Function – klasa, której obiekt reprezentuje funkcję logiczną obliczającą wejście do przerzutnika na podstawie przypisanych do niego wyjść przerzutników. Może zwrócić wynik funkcji logicznej dla podanego stanu rejestru.

• Register – klasa, której obiekt reprezentuje rejestr symulowany przez program. Najważniejszą częścią jej funkcjonalności jest przejście do następnego stanu rejestru (wykonanie jednego kroku). Jest też w stanie wykryć zapętlenie rejestru.

• Register_Manager – klasa, której obiekt wykonuje operacje na rejestrze (obiekcie klasy Register), aby uzyskać wyniki programu: listę wygenerowanych ciągów, stopień wykorzystania przestrzeni oraz średnią różnorodność ciągów.

Moduły zawarte w programie:

• main.py – główna funkcja programu, otwierająca pliki i wypisująca informacje
• exceptions.py – deklaracja wyjątków używanych w programie
• exceptioninfo.py – funkcja uzyskująca komunikat o błędzie na podstawie wyjątku
• logicfunction.py – klasa Logic_Function i powiązane funkcje
• register.py – klasa Register i powiązana funkcja
• registermanager.py – klasa Register_Manager i powiązane funkcje
• iofunctions.py – funkcje zajmujące się komunikacją programu z otoczeniem

6. Refleksje

Zostało wykonane: napisanie kodu programu, przygotowanie przykładowego pliku wejściowego i wyjściowego, napisanie testów jednostkowych do programu, opisanie każdej klasy i funkcji programu docstringami oraz opisanie całości programu dokumentacją.

W pierwotnym koncepcie rozwiązania były tylko dwie klasy - Logic_Function oraz Register. Ponadto, zapętlenie rejestru było wykrywane tylko w sytuacji, gdy rejestr powrócił do stanu początkowego. W związku z brakiem problemów z czasem wykonywania programu, wykrywanie zapętleń zostało rozszerzone do wszystkich zapętleń. Została również dodana klasa Register_Manager, w celu lepszego zgrupowania pozaklasowych funkcji w programie.

Problemem podczas wykonywania rozwiązania było ustalenie importów między plikami, szczególnie w przypadku testów oraz powiązane z tym nietypowe zachowanie rozszerzenia pytest. Zostało to jednak rozwiązane.

Powierzone zadanie zostało w całości wykonane. Program jest w stanie poprawnie przetworzyć wszystkie rejestry zgodne z założeniami zadania, a także niektóre inne. Funkcjonalność programu została zatem rozszerzona w stosunku do treści zadania.