UMCS
UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
W LUBLINIE
Wydział Matematyki, Fizyki i Informatyki

Kierunek: **Informatyka**

Viktar Zhdanovich
Nr albumu: 300187

Projekt i implementacja interfejsu graficznego SAT-solvera

Design and implementation of a SAT-solver graphical interface

Praca licencjacka
napisana w Katedrze Systemów Inteligentnych
pod kierunkiem dr. Jacka Krzaczkowskiego

Lublin 2023

Spis treści

Wstęp

Graficzny interfejs użytkownika (ang. graphical user interface, GUI) jest typem interfejsu, za pomocą którego użytkownik wchodzi w interakcję z komputerem, programem. Alternatywą do interfejsów graficznych są interfejsy konsolowe (ang. command line interface, CLI). Wadą takiego sposobu korzystania z programu jest to, że trzeba pamiętać komendy i je ręcznie wpisywać. Z tego powodu, że to jest uciążliwe dla większości ludzi, popularne stały się interfejsy graficzne, które są bardzo ważną częścią prawie każdej współczesnej aplikacji. Brak interfejsu podwyższa poziom przygotowania koniecznego do korzystania z takiej aplikacji. Dobrze przemyślany, atrakcyjny, niezawodny, pomagający użytkownikowi interfejs na pewno poszerza grono użytkowników programu, co może m.in dobrze wpłynąć na rozwój przedsiębiorstwa lub sukces twórcy.

Celem pracy było zaprojektowanie i implementacja webowego interfejsu graficznego SAT-solvera.

Rozdział pierwszy zawiera wprowadzenie do tematyki pracy. W drugim rozdziale opisano projekt aplikacji. W trzecim rozdziale przedstawiono szczegóły implementacji niektórych części aplikacji.

Rozdział 1

Wprowadzenie

1.1 CNF

Zmienna zdaniowa to zmienna, która może przyjmować wartość *true* lub *false*.

Literałem nazywamy pojedynczą zmienną zdaniową lub pojedynczą zanegowaną zmienną zdaniową. Przykłady literałów:

$$p, q, \neg p$$

Klauzula jest zbiorem literałów połączonych alternatywą. Przykład klauzuli:

$$(x \vee y \vee z).$$

Koniunkcyjna postać normalna (ang. conjunctive normal form) danej formuły logicznej to równoważna jej formuła zapisana w postaci koniunkcji klauzul. Przykład formuły w postaci CNF:

$$(x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_1)$$

Każdą formułę logiczną można przedstawić równoważnie w postaci CNF. Dla jednej formuły logicznej może istnieć kilka równoważnych jej formuł w CNF.

1.2 Problem spełnialności formuł logicznych

Problem spełnialności formuł logicznych jest ważnym problemem obliczeniowym w teorii złożoności. Wejściem problemu jest formuła logiczna w postaci CNF. Problem jest decyzyjny i polega na znalezieniu odpowiedzi na pytanie, czy istnieje wartościowanie, tzn. czy da się znaleźć takie wartości zmiennych zdaniowych formuły, dla których formuła jest spełniona. Formułę, która posiada takie wartościowanie nazywamy formułą *spełnialną* (ang. satisfiable), a taka, która takiego wartościowania nie posiada, odpowiednio *niespełnialną* (ang. unsatisfiable).

Na przykład, formuła

$$(x_1 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg x_1)$$

jest spełnialna dlatego, że istnieje wartościowanie, które ją spełnia, na przykład:

$$x_1 = false, x_2 = false, x_3 = false$$

1.3 SAT-solvery

SAT-solver jest programem, mającym na celu rozwiązanie problemu spełnialności formuł logicznych. Na wejściu program przyjmuje formulę w postaci CNF, a na wyjściu zwraca odpowiedź, czy podana formuła jest lub nie jest spełnialna.

Większość SAT-solverów zwraca nie tylko informację o spełnialności, ale też przykładowe wartościowanie spełniające, jeśli formuła jest spełnialna, lub minimalny zbiór niespełnialnych klauzul, jeśli formuła jest niespełnialna.

SAT-solvery mają wiele udanych zastosowań w różnych dziedzinach, takich jak np. sztuczna inteligencja, automatyzacja projektowania elektronicznego (ang. Electronic Design Automation) [1].

Istnieją różne rodzaje SAT-solverów, które rozwiązują inne odmiany problemu spełnialności formuł logicznych:

- MAX-SAT - szuka maksymalnego podzbioru klauzul formuły, które dają formułę spełnialną.
- Partial MAX-SAT [1] - dzieli formułę na klauzule miękkie i twarde, celem jest znalezienie wartościowania, które spełnia wszystkie klauzule twarde i maksymalizuje liczbę spełnionych klauzul miękkich.

1.4 Format DIMACS CNF

DIMACS CNF [2] to format tekstowy reprezentujący formułę w koniunkcyjnej postaci normalnej.

Reguły których trzeba pilnować, żeby poprawnie stworzyć plik z formułą:

1. Linie zaczynające się od znaku c zawierają komentarze.
2. Linia zaczynająca się od znaku p jest definicją formuły i wygląda następująco: p *cnf* l k, gdzie l i k są liczbami dodatnimi, l reprezentuje liczbę zmiennych formuły, a k liczbę klauzul.
3. Klauzule są umieszczane po definicji formuły. Każda klauzula jest zakodowana jako sekwencja liczb dziesiętnych odseparowanymi spacjami. Liczby dodatnie oznaczają zmienne niezanegowane, a liczby ujemne zmienne zanegowane.
4. Każda linia zawiera dokładnie jedną klauzulę i ma kończyć się symbolem 0.
5. Zostawianie pustych linii w formule nie jest dozwolone.

Niepilnowanie tych reguł może doprowadzić do błędnych wyników lub w ogóle do zawieszenia się SAT-solvera.

Przykładowo formuła w CNF

$$(x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2) \wedge (x_3 \vee x_4 \vee \neg x_1) \wedge (x_2 \vee x_3 \vee \neg x_1)$$

może być zapisana w DIMACS CNF jako:

```
p cnf 4 4
1 2 3 4 0
-1 2 0
3 4 -1 0
2 3 -1 0
```

1.5 Cel i zakres pracy

Celem pracy było stworzenie intuicyjnego i dostępnego interfejsu graficznego dla SAT-solvera. Dostępność miała zostać zapewniona poprzez stworzenie aplikacji webowej, która nie wymaga od użytkownika żadnych niepotrzebnych instalacji, działa prawie wszędzie, a do uruchomienia potrzebny jest tylko komputer i internet. Intuicyjność miała zostać zapewniona poprzez przemyślany interfejs, który w razie problemów pomaga użytkownikowi w ich rozwiązaniu za pomocą odpowiednich komunikatów i sugestii.

Wśród szczegółowych wymagań znalazły się:

1. Edytor plików w formacie DIMACS CNF, który wskazuje błędy w formule oraz pomaga użytkownikowi je naprawić.
2. Przekształcanie formuły z DIMACS CNF do CNF oraz umożliwienie modyfikacji formuły w tej postaci.
3. Możliwość zliczania i znalezienia wszystkich wartościowań spełniających formulę.
4. Możliwość łączenia formuł.

Rozdział 2

Projekt

2.1 Funkcjonalności

2.1.1 Obsługa błędów

Wyróżniono następujące błędy w formule w postaci DIMACS CNF, każdemu z nich z góry przepisano unikatowy kod błędu:

- *Kod 0*: Nie można zostawiać pustych linii.
- *Kod 1*: Definicja formuły może być niepoprawna.
- *Kod 2*: Każda klauzula ma kończyć się zerem.
- *Kod 3*: Klauzula może być niepoprawna.
- *Kod 4*: W pliku nie może być więcej niż jedna definicja formuły.
- *Kod 5*: Zmienna w klauzuli nie jest zadeklarowana w definicji.

Dla każdego z wymienionych powyżej błędów zostaną dodane indywidualne sugestie do poprawienia:

- *Kod 0*: Usuwanie lub edycja linii.
- *Kod 1*: Edycja linii z definicją.
- *Kod 2*: Dopisywane zera na końcu linii.
- *Kod 3*: Usuwanie lub edycja linii z klauzulą.
- *Kod 4*: Wymiana istniejącej lub usuwanie linii z definicją.
- *Kod 5*: Edycja linii z klauzulą.

Błędów może być dużo, więc dodana zostanie możliwość automatycznego poprawiania formuły. Przy tym w formule nastąpią zmiany:

1. Na końcach linii będą dopisane zera.
2. Liczby zmiennych i klauzul w nagłówku będą poprawione na odpowiednie.

3. Puste linie i niepoprawne klauzule będą usuwane.

4. Komentarze będą usuwane.

Ze względu na to, że formuła może zostać uszkodzona, użytkownik będzie poproszony o potwierdzenie chęci zrobienia automatycznego poprawienia.

Zostanie stworzony komponent z tabelą do wyświetlania wszystkich błędów, użytkownik będzie mógł szybko je przeglądać i naprawiać. Błędy w tabeli będą umieszczane w kolejności ich wykrycia.

W edytorze będą podświetlane linie z błędami oraz będzie dana możliwość szczegółowego przejrzenie w tym samym miejscu informacji o konkretnym błędzie oraz możliwość natychmiastowego poprawienia go.

2.1.2 Praca z SAT-solverem

Zostaną dodane podstawowe funkcje dotyczące pracy z SAT-solverem. Będzie można:

1. Dowiedzieć się, czy formuła jest lub nie jest spełnialna.

2. Znaleźć pojedyncze wartościowanie spełniające formułę.

3. Spróbować znaleźć kolejne wartościowanie, jeśli formuła jest spełnialna.

4. Uruchomić i w dowolnej chwili skończyć proces znajdowania wszystkich wartościowań spełniających.

5. Wybrać SAT-solver.

2.1.3 Możliwości edytora

Zostaną dodane następujące funkcje aplikacji dotyczące pracy z edytorem. Będzie można:

1. Pozbyć się zduplikowanych klauzul w formule. Będą usuwane nie tylko dokładnie takie same klauzule, ale także równoważne, ze względu na to, że alternatywa jest przemienna.

2. Zapisać do pliku z rozszerzeniem *txt* formułę w formacie DIMACS CNF.

3. Wczytać formułę w postaci DIMACS CNF z pliku z rozszerzeniem *cnf* lub *txt*.

2.2 Komponenty

Komponenty pozwalają podzielić interfejs graficzny na niezależne, pozwalające na ponowne użycie części i myśleć o każdej z nich osobno.

Biorąc pod uwagę funkcjonalności aplikacji, zaplanowano następujące komponenty:

- Edytor.
- Błędy.
- Wartościowania spełniające formułę.
- Formuła w postaci CNF.
- Łączenie formuł.

2.2.1 Edytor

Rysunek 2.1: Komponent edytora plików DIMACS CNF wraz z komponentem do pracy z SAT-solverem

Na rysunku 2.1 większą część zajmuje pole tekstowe, w którym według projektu będzie modyfikowana formuła. Pod numerem 8 umieszczony jest pasek przewijania, który pomoże przemieszczać się po formułach z dużą liczbą klauzul. Po lewej stronie, pod numerem 9, znajduje się słupek "przypisujący"numer linii do jej zawartości, będzie potrzebny do wskazywania błędów.

Po prawej stronie zostanie dodany obszar z przyciskami do zarządzania edytorem:

- Nr 4 odpowiada za załadowanie formuły do edytora z pliku.
- Nr 5 odpowiada za zapisanie formuły do pliku.

- Nr 6 odpowiada za automatyczne poprawienie błędów.
- Nr 7 odpowiada za usuwanie zduplikowanych klauzul.

Żeby znaczenie ikon przycisków z obszaru do zarządzania edytorem było jasne, zostanie dodana możliwość zobaczenia w postaci tekstowej informacji o znaczeniu przycisku. Po najechaniu myszką na każdy z przycisków pod nimi będzie pojawiała się podpowiedź (ang. tooltip) z tekstem. Na przykład pod przyciskiem nr 1 pojawi się podpowiedź "Upload formula".

Na dole będzie znajdował się komponent do pracy z SAT-solverem składający się z czterech przycisków:

- Nr 1 odpowiada za wyszukiwanie wszystkich wartościowań. Wyszukiwanie można będzie przerwać w dowolnej chwili. Po przerwaniu wyszukiwania, można będzie je ponownie zacząć od miejsca w którym przerwano, w tym przypadku na przycisku będzie znajdował się tekst "Find other solutions".
- Nr 2 odpowiada za sprawdzenie, czy formuła jest lub nie jest spełnialna. Jeśli jest spełnialna zwracane będzie przykładowe wartościowanie.
- Nr 3 odpowiada za znalezienie kolejnego wartościowania, jeśli takie istnieje. Będzie nieaktywny, jeśli formula będzie niespełnialna.
- Nr 4 odpowiada za wybór SAT-solvera.

Jeśli formuła w edytorze będzie zawierała błędy - uruchomienie SAT-solvera nie będzie możliwe. Wszystkie wymienione przyciski oprócz nr 4 będą nieaktywne.

Rysunek 2.2: Komponent edytora z formułą, w której zostały wykryte błędy

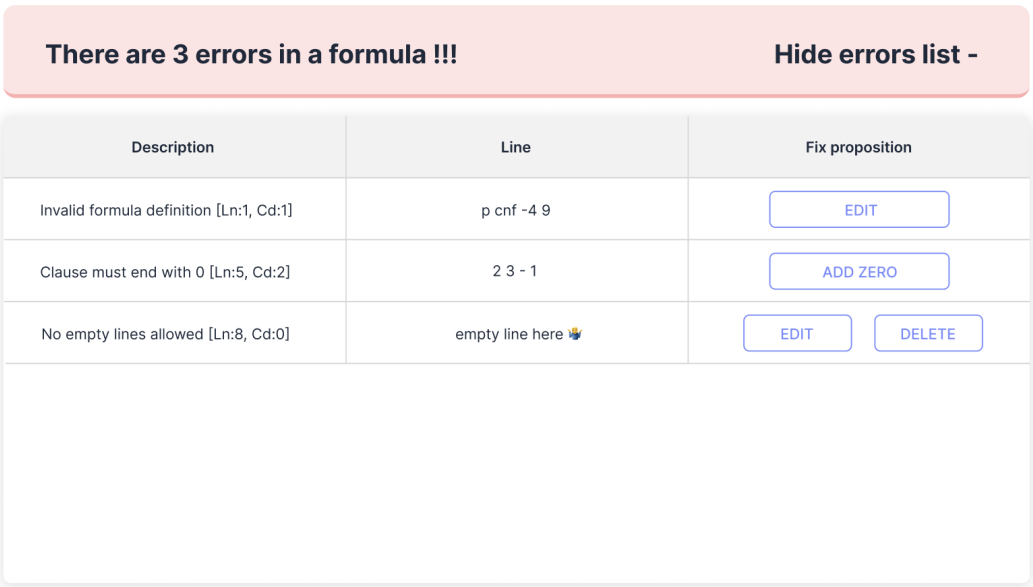
Na rysunku 2.2 można zobaczyć jak będzie wyglądał edytor z formułą w której zostały wykryte błędy. Takie linie będą podświetlanie czerwonym kolorem. Po najechaniu myszką na linię z błędem będzie pojawiała się informacja, w której w pierwszej linii będzie znajdowała się treść uszkodzonej linii, w drugiej opis błędu, a w trzeciej propozycja

sposobu naprawienia w postaci przycisku. W tym przykładzie widać, że zaproponowano dodanie zera na końcu klauzuli.

Według projektu edytor znajduje się na samej górze strony.

2.2.2 Błędy

Według projektu, komponent na rysunku 2.3 jest umieszczony pomiędzy edytorem a przyciskami do pracy z SAT-solverem. Komponentu nie będzie na stronie, kiedy formuła, z którą będzie pracował użytkownik nie będzie zawierała błędów. Będzie miał dwa stany: otwarty i zamknięty. Domyślnie będzie ustawiony stan zamknięty. Naciśnięcie na czerwony prostokąt będzie zmieniało stan komponentu z zamkniętego na otwarty i na odwrót. Niezależne od stanu otwarcia na górze zawsze będzie znajdował się czerwony prostokąt, w którym z lewej strony będzie napisane ile błędów jest w formule. Poniżej nagłówka będzie umieszczona tabela zawierająca wszystkie błędy. Będą w niej trzy kolumny: w pierwszej będzie znajdował się opis błędu, w drugiej zawartości uszkodzonej linii, a w trzeciej będą przyciski, dzięki którym użytkownik będzie mógł szybko naprawić błąd nie szukając go w edytorze.



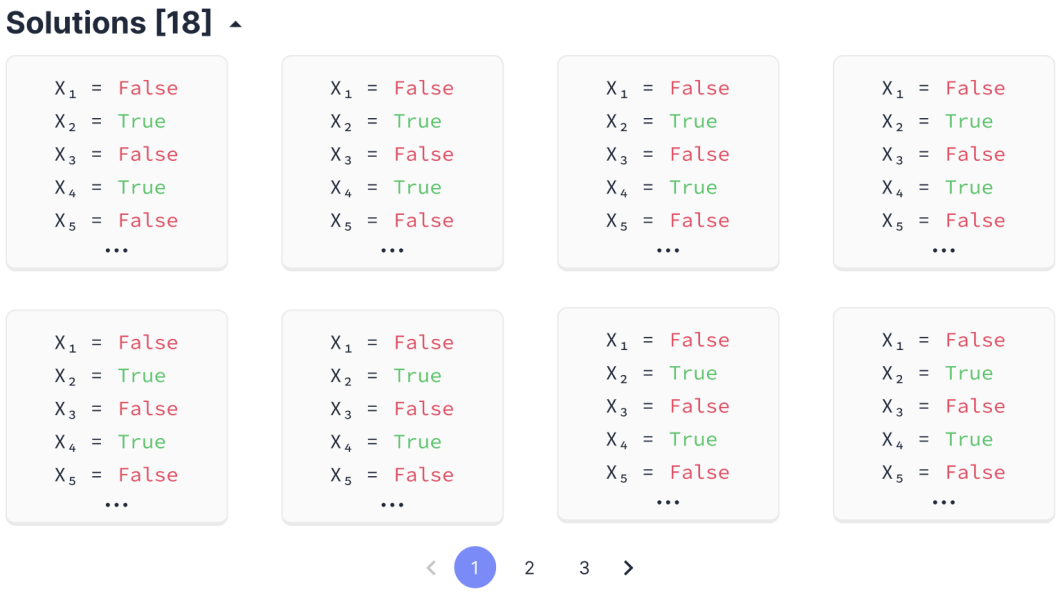
Rysunek 2.3: Komponent, w którym wyświetlane są wszystkie znalezione błędy

2.2.3 Wartościowania spełniające formułę

Na górze komponentu (patrz rysunek 2.4) będzie znajdował się nagłówek w którym będzie napisane ile wartościowań spełniających formułę znaleziono. Naciśnięcie na nagłówek będzie zmieniało stan komponentu z zamkniętego na otwarty i na odwrót. Na dole zostanie umieszczony komponent pozwalający przełączać strony z rozwiązaniami. Rozdzielenie wartościowań spełniających formułę na strony wprowadzono ze względów wydajnościowych. Na każdej stronie będzie znajdowała się stała ilość elementów.

W projekcie wartościowania spełniające formułę są wyświetlane w siatce z 4 kolumnami i 2 wierszami. Liczba kolumn i wierszy może się różnić w implementacji. W projekcie

przedstawiony został wygląd komponentu niezajmujący dużo miejsca na ekranie. Każde z wartościowań spełniających formułę jest oddzielnym komponentem.



Rysunek 2.4: Komponent, w którym wyświetlane są wartościowania spełniające formułę z edytora

Rysunek 2.5: Komponent wyświetlający całe wartościowanie

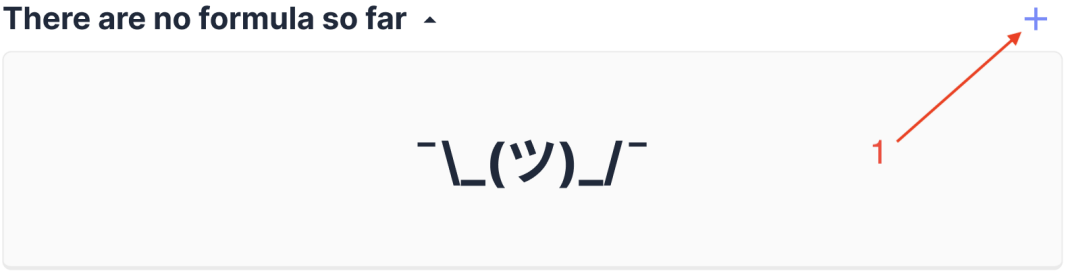
Wyświetlanie całego wartościowania będzie odbywało się tylko na życzenie użytkownika. Kiedy użytkownik będzie naciskał na dowolne z wartościowań, będzie otwierało się okno pozwalające zobaczyć całość. Widać to na rysunku 2.9. Znajduje się tu przycisk "Save to a file in a 1-0 form", który pozwoli zapisać wartościowanie spełniające formułę do pliku z rozszerzeniem *txt* w postaci słupka zer i jedynek, gdzie każda nowa linia odpowiada za indeks (indeksowanie zaczyna się od liczby 1) zmiennej w formule, a za wartość odpowiada liczba znajdująca się w tej linii. Liczba 0 będzie oznaczała wartość logiczną *false*, a liczba 1 będzie oznaczała wartość logiczną *true*.

Dopóki użytkownik nie znajdzie co najmniej jednego wartościowania spełniającego formulę, komponent będzie znajdował się w stanie mówiącym o braku rozwiązań.

2.2.4 Formuła w postaci CNF

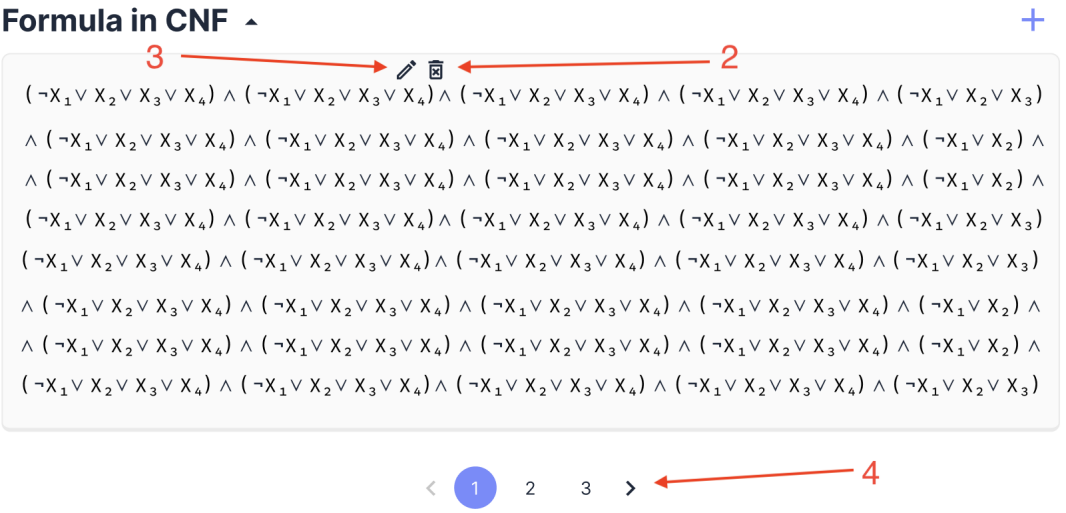
Według projektu, komponent będzie dawał możliwość zobaczenia i edytowania formuły inaczej, niż w edytorze DIMACS, a mianowicie w zwykłej postaci CNF.

Na rysunku 2.6 komponent znajduje się w stanie kiedy formuła jeszcze nie jest dodana. Kiedy użytkownik będzie naciskał na nagółwek komponentu będzie zmieniał jego stan z otwartego na zamknięty i na odwrót. Komponent domyślnie będzie znajdował się w stanie otwartym. Przycisk nr 1 będzie pozwalał ręcznie dodawać klauzule do formuły. Pojawi się do tego odpowiednie pole z miejscem do wpisywania wraz z instrukcją, w której zostanie wytłumaczone jak to prawidłowo zrobić.



Rysunek 2.6: Komponent z formułą w stanie, kiedy formuła jeszcze nie została dodana

Na rysunku 2.7 komponent znajduje się w stanie, kiedy formuła już została dodana. Przyciski nr 2 i nr 3 pozwolą modyfikować wybraną klauzulę. Będą pojawiać się nad każdą klauzulą po najechaniu na nią myszką. Przycisk nr 2 będzie pozawalał usunąć klauzulę, po naciśnięciu na niego użytkownik zostanie zapytany, czy na pewno chce to zrobić. W razie pozytywnej odpowiedzi klauzula zostanie usunięta. Po naciśnięciu na przycisk nr 3 użytkownik dostanie możliwość modyfikacji zawartości klauzuli.



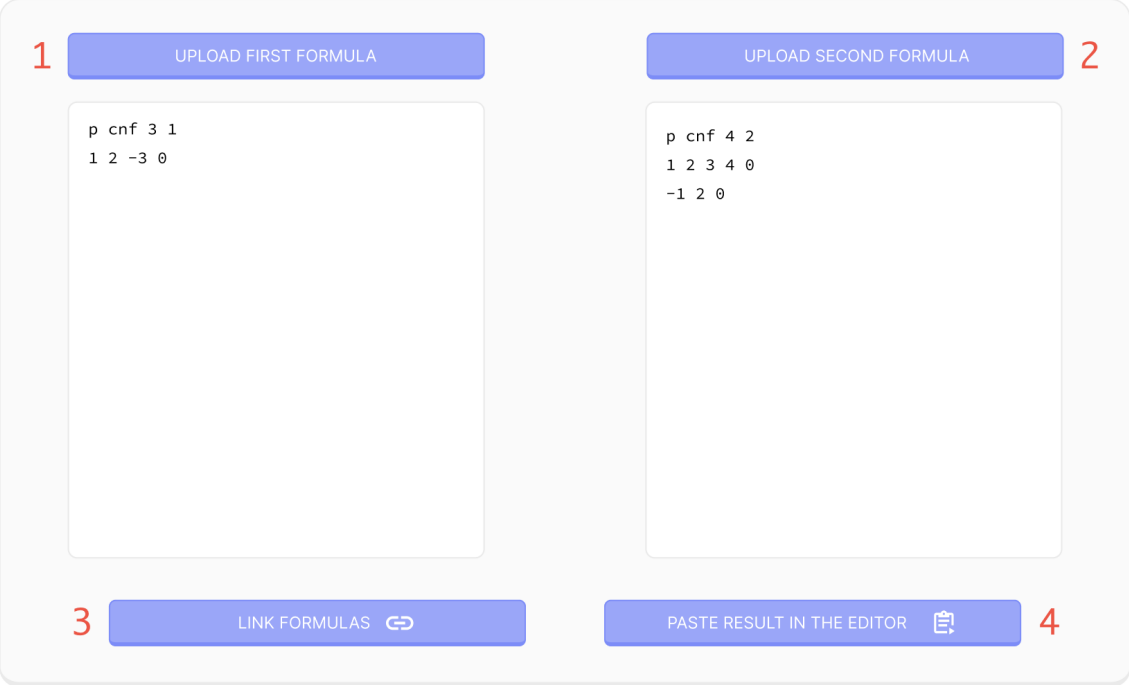
Rysunek 2.7: Komponent z formułą

Komponent nr 4 da możliwość przemieszczania się po formule. Podobnie jak w przypadku wartościowań spełniających formułę, rozdzielono dane na strony. Ilość klauzul wy-

świetlanych na jednej stronie będzie stała: domyślnie na jednej stronie maksymalnie będzie wyświetlane 115 klauzul. Takie rozdzielenie dobrze wpłynie na wydajność aplikacji dlatego, że przeglądarka użytkownika nie będzie wyświetlać całą formułę naraz, a tylko odpowiednią jej część.

2.2.5 Łączenie formuł

Komponent na rysunku 2.8 zawiera dwa pola tekstowe dla formuł, z możliwością modyfikowania oraz cztery przyciski. Przyciski nr 1 i nr 2 pozwolą załadować formuły z plików do pól tekstowych. Kiedy dwie formuły zostaną załadowane, przycisk nr 3 stanie się aktywny i użytkownik dostanie możliwość naciśnięcia go i połączenia formuł. Jeśli nie wystąpią błędy podczas łączenia, pojawi się odpowiedni komunikat, że formuły zostały pomyślnie połączone oraz zostanie odblokowany przycisk nr 4. Kiedy użytkownik naciśnie na ten przycisk, to zostanie przekierowany na stronę z edytorem plików DIMACS CNF, w który już będzie wklejony rezultat łączenia.



Rysunek 2.8: Komponent łączący formuły

Żeby nie obciążać aplikacji i logicznie podzielić ją na zadania, które wykonuje, w projekcie przydzielono temu komponentowi całą oddzielną stronę mająca nazwę "Formulas linker". Reszta omówionych powyżej komponentów będzie znajdowała się na stronie "Home". Użytkownikowi będzie dana możliwość przemieszczania się między stronami.

Rozdział 3

Implementacja

Kliencka część aplikacji została stworzona za pomocą darmowej i otwartej biblioteki **React** [3] dla języka JavaScript przeznaczonej do tworzenia interfejsów graficznych. Serwerowa część została stworzona za pomocą darmowego i otwartego frameworku dla języka Python **FastAPI** [4]. Na serwerze został zainstalowany pakiet **PySAT** [5], który dał możliwość korzystania z trzynastu zaimplementowanych SAT-solverów. Komunikację pomiędzy klientem a serwerem zapewniono poprzez zapytania HTTP.

Serwerowa część aplikacji odpowiada za następujące zadania:

- Stwierdzenie czy formuła jest lub nie jest spełnialna i zwracanie przykładowego wartościowania spełniającego formułę.
- Znalezienie kolejnego wartościowania spełniającego formułę.
- Usuwanie zduplikowanych klauzul z formuły.
- Automatyczne naprawienie formuły.
- Łączenie formuł.
- Sprawdzenie rzeczywistej liczby klauzul i liczby zadeklarowanych klauzul w formule. W przypadku nierówności tych liczb wyrzucenie błędu.
- Sprawdzenie, czy definicja formuły istnieje. W przypadku jej nieobecności wyrzucenie błędu.

Kliencka część aplikacji odpowiada za następujące zadania:

- Dostarczenie użytkownikowi interfejsu graficznego, np. interfejsu do ręcznego poprawienie błędów w formule itd.
- Wizualizacja danych zwróconych z serwera.
- Sprawdzanie na bieżąco w edytorze błędów w formule.

3.1 Typy danych

3.1.1 Literał

Literał jest podstawowym elementem, z którego składa się formuła. Literał w implementacji jest liczbą całkowitą i ma typ danych *number*. Taka liczba oznacza indeks literału w formule, np. liczba 1 oznacza literał x_1, a liczba -2 oznacza literał $\neg x_2$.

3.1.2 Klauzula

Każda klauzula jest obiektem o następującym typie danych:

```
interface IClause {
    id: number;
    variables: number[];
}
```

- **id** to unikatowy identyfikator klauzuli. Identyfikator jest potrzebny, żeby prawidłowo znajdować klauzulę do zmodyfikowania lub usunięcia.
- **variables** to tablica przechowująca literały klauzuli, np. tablica [-1, 2, 3] oznacza $(\neg x_1 \vee x_2 \vee x_3)$

3.1.3 Formuła

Formuła składa się z klauzul, więc logiczne jest przedstawienie jej jako tablicę obiektów typu *IClause*.

Na przykład następująca struktura w języku JavaScript:

```
[
    {
        id: 0,
        variables: [1, 2, 3]
    },
    {
        id: 1,
        variables: [-1, -2]
    },
    {
        id: 2,
        variables: [1, -3]
    }
]
```

odpowiada następującej formule:

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_3)$$

3.1.4 Błędy

Każdy błąd jest obiektem o następującym typie danych:

```
interface IError {
    line: number;
    errorCode: 0 | 1 | 2 | 3 | 4 | 5;
    description: string;
    damaged: string;
}
```

- **line** to linia wystąpienia błędu.
- **errorCode** to kod błędu.
- **description** to opis błędu.
- **damaged** to zawartość linii z błędem.

Wszystkie błędy są przechowywane w tablicy przechowującej obiekty typu *IError*.

3.1.5 Wartościowania spełniające formułę

Pojedyncze wartościowanie spełniające formułę jest tablicą liczb całkowitych, w postaci, w jakiej je zwraca SAT-solver, np. tablica [1, 2, -3] oznacza następujące wartościowanie:

$$x_1 = 1, x_2 = 1, x_3 = -1$$

Wszystkie wartościowania spełniające formułę są przechowywane w tablicy tablic liczb całkowitych.

3.2 Implementacja edytora

Podstawą edytora jest wieloliniowe pole tekstowe z wbudowaną możliwością przewijania zawartości po przekroczeniu zadeklarowanej wysokości. Takie pole jest zwykłym tagiem HTML nazywającym się *textarea*. Ma taką samą stałą wysokość, jak i kontener edytora: 545 pikseli.

Z lewej strony w edytorze znajduje się komponent pilnujący liczby linii w wieloliniowym polu tekstowym. Jest kontenerem dla elementów zawierających tekst z indeksem linii. Żeby prawidłowo go wyświetlić, trzeba wiedzieć ile linii jest w pliku z formułą. Liczba linii jest przeliczana za każdym razem, kiedy zmieni się liczba linii w polu tekstowym, a nie sama zawartość pliku, tzn. klauzula, definicja formuły itd. Kontener ma taką samą wysokość jak wieloliniowe pole tekstowe. Dodana została możliwość przewijania zawartości kontenera, kiedy zadeklarowana wysokość będzie przekroczona.

Kontener z błędami w edytorze ma taką samą szerokość, jak i wieloliniowe pole tekstowe. Znajduje się warstwę wyżej, dokładnie nad polem tekstowym (ma większą wartość właściwości CSS *z-index*). Każdy błąd wewnątrz kontenera jest oddzielnym komponentem i ma pozycjonowanie relatywne (ang. relative), za co odpowiada właściwości CSS *position* z wartością *relative*. Relatywna pozycja oznacza, że element będzie pozycjonowany względem samego siebie. Położenie każdego elementu posiadającego pozycję relatywną może być sterowane za pomocą właściwości *top*, *bottom*, *right*, *left*. Zmiana położenia elementu za pomocą powyższych właściwości do sterowania nie wpływa na położenie pozostałych elementów. W miejscu sprzed przemieszczeniem zawsze pozostaje "dziura" po elemencie, dokładnie jak gdyby ten element nie był w żaden sposób przemieszczony, ale znajdował się w tym samym miejscu i zajmował przestrzeń.

3.2.1 Obliczanie położenia linii z błędem

Położenie każdego błędu w kontenerze jest obliczane oddzielnie i ustawiane za pomocą właściwości CSS *top*, która w przypadku pozycji relatywnej oznacza o ile górna krawędź elementu jest przesunięta poniżej jego normalnej pozycji. Właściwość *top* może przyjmować wartości ujemne. Zwiększenie wartości właściwości *top* będzie oznaczało przemieszczenie elementu do dołu, a zmniejszanie do góry. Poniżej w schematyczny sposób przedstawiono działanie kontenera z błędami.

Domyślnie elementy w kontenerze są umieszczane element pod elementem. Każdy nowo dodany element znajduje się w swojej normalnej pozycji i ma wartość właściwości *top* ustawioną na 0 pikseli:

```
---------kontener---------
(1) err[0] -> top: 0px
(2) err[1] -> top: 0px
(3) err[2] -> top: 0px
---------kontener---------
```

1. Każdy element w kontenerze nazywa się *err* (w rzeczywistości jest czerwonym prostokątem podświetlającym linię z błędem). Ma wysokość 20 pikseli.

2. Liczba znajdująca się w okrągłych nawiasach oznacza numer linii w edytorze.

3. Liczba znajdująca się w kwadratowych nawiasach oznacza indeks błędu w tablicy z błędami.

4. Strzałka w prawo wskazuje na wartość właściwości CSS *top* tego elementu. *px* oznacza pikseli.

5. Symbol "&" oznacza, że w linii, dokładnie na tym samym miejscu znajdują się 2 elementy. Te elementy nakładają się na siebie.

Jeśli dodać do kontenera kolejny element z wartością właściwości *top* ustawioną na 20 pikseli, kontener będzie wyglądał następująco:

```
---------kontener---------
(1) err[0] -> top: 0px
(2) err[1] -> top: 0px
(3) err[2] -> top: 0px
(4) "dziura" po err[3] o wysokości 20 pikseli
(5) err[3] -> top: 20px
---------kontener---------
```

Po przemieszczeniu, w linii nr 4 na normalnej pozycji elementu z indeksem 3 pojawi się "dziura". Jeśli dodać do kontenera następny element i nie ustawiać wartość jego właściwości *top* sytuacja będzie wyglądała następująco:

```
---------kontener---------
(1) err[0] -> top: 0px
(2) err[1] -> top: 0px
(3) err[2] -> top: 0px
(4) "dziura" po err[3]
(5) err[3] & err[4] -> top: 20px & top: 0px
---------kontener---------
```

Widać, że nowy element z indeksem 4 znajduje się na tym samym miejscu, co i element z indeksem 3, przykrywając go. Żeby element z indeksem 4 znalazł się na linii nr 6 trzeba ustawić wartość jego właściwości *top* na 20 pikseli:

```
---------kontener---------
(1) err[0] -> top: 0px
(2) err[1] -> top: 0px
(3) err[2] -> top: 0px
(4) "dziura" po err[3]
(5) err[3] -> top: 20px & "dziura" po err[4]
(6) err[4] -> top: 20px
---------kontener---------
```

Żeby element z indeksem 4 znalazł się na linii nr 4 trzeba ustawić wartość jego właściwości *top* na -20 pikseli:

```
---------kontener---------
(1) err[0] -> top: 0px
(2) err[1] -> top: 0px
(3) err[2] -> top: 0px
(4) err[4] -> top: -20px & "dziura" po err[3]
(5) err[3] -> top: 20px & "dziura" po err[4]
---------kontener---------
```

Biorąc pod uwagę powyższe zachowanie kontenera z elementami mającymi pozycję relatywną, wyprowadzono następująca formułę do obliczania położenia linii z błędem w pikselach:

$$(line - index - 1) * errorComponentHeight$$

gdzie $line \geqslant 1$ to rzeczywista linia wystąpienia błędu, $index \geqslant 0$ jest indeksem błędu w tablicy z błędami, a $errorComponentHeight = 20$ jest stałą wysokością czerwonej linii podświetlającej błąd w pikselach.

3.2.2 Synchronizacja działania komponentów edytora

Właściwość *scrollTop* pobiera lub ustawia liczbę pikseli, o którą zawartość elementu została przewinięta. Jeśli zawartość elementu nie może być przewinięta pionowo, właściwość scrollTop elementu zawsze równa się 0.

Domyślnie przewijanie wieloliniowego pola tekstowego, kontenera z numerami linii i kontenera z błędami nie synchronizuje się w żaden sposób: przewijają się całkiem oddzielnie. Potrzebne zachowanie można zapewnić za pomocą właściwości *scrollTop* elementów. Pseudokod tego zdarzenia wygląda następująco:

```
-> 1. kiedy przewijane jest wieloliniowe pole tekstowe <textarea/>:
-> 2.   linesContainer.scrollTop = textarea.scrollTop
-> 3.   errorsContainer.scrollTop = textarea.scrollTop
```

Aby synchronizacja przewijania działała prawidłowo należy również synchronizować wygląd poszczególnych komponentów. W ostatecznej wersji programu w tym celu dokonano następujące konfiguracji stylów:

1. Wysokości oddzielnych elementów edytora takich jak: krotka z numerem linii, czerwona linia podświetlająca błąd, linia w wieloliniowym polu tekstowym ustawione są na 20 pikseli. Odpowiada za to właściwość CSS *height*.

2. Rozmiary czcionek w krotce z numerem linii oraz w wieloliniowym polu tekstowym są ustawione na 16 pikseli. Odpowiada za to właściwość CSS *font-size*.

3. Wysokości linii w krotce z numerem linii i wieloliniowym polu tekstowym są ustawione na wartość bez jednostki równą 1.25. Odpowiada za to właściwość CSS *line-height*. Ustawienie wysokości linii na wartości bez jednostki oznacza jej zależność od rozmiaru czcionki w elemencie. Wartość wysokości linii w pikselach wyrażana jest formułą: *line-height* $\times$ *font-size*.

4. Odstęp wewnętrzny od góry w kontenerze z błędami, kontenerze z numerami linii oraz wieloliniowym polu tekstowym ustawione są na 10 pikseli. Odpowiada za to właściwość CSS *padding-top*.

5. Tekst w komponencie edytora wszędzie ma mieć taką samą czcionkę. Odpowiada za to właściwość CSS *font-family*. W programie wybrano czcionkę *Source Code Pro* [6].

W rezultacie udało się osiągnąć efekt, że wszystkie elementy przewijają się jednocześnie i cały edytor wyglądają jak jedna całość. Czerwone linie podświetlające błąd idealnie są mapowane do swojej linii wystąpienia i podświetlają dokładnie te linie, które trzeba.

3.2.3 Informacja o błędzie

Po najechaniu myszką na czerwony prostokąt podświetlający linię z błędem jest włączany timer na 250 milisekund. Jeśli użytkownik usunie kursor myszki z czerwonej linii przed upływem tego czasu: timer się anuluje. Po przekroczeniu tego czasu jest zapisywana obecna pozycja kursora myszki. Ta informacja jest przekazywana do komponentu ze szczegółami błędu nazywającego się *ErrorInfo*. Zanim będzie wyświetlony na ekranie, najpierw będzie obliczona jego pozycja względem tej konkretnej linii i edytora.

Dodane są warunki, że komponent ze szczegółami błędu nie może przekroczyć granic edytora ani z prawej, ani z lewej strony.

Komponent ze szczegółami błędu zwykle jest pokazywany nad czerwoną linią. Dodano również warunek, że komponent będzie pokazywany pod linią z błędem dla pierwszych sześciu linii w wieloliniowym polu tekstowym dlatego, że w przeciwnym przypadku będą przekroczone granicy edytora z góry i komponent ze szczegółami może być częściowo lub całkiem niewidoczny.

3.3 Testy

3.3.1 Działanie edytora

Zakłada się, że podczas pracy z edytorem, będą najczęściej załadowywane formuły niezawierające żadnych błędów lub zawierające ich stosunkowo mało. Przypadki, w których będą załadowywane formuły zawierające dużą liczbę błędów (więcej niż 1000) są pesymistyczne i mało prawdopodobne.

Największą formułą za pomocą której była sprawdzana wydajność działania edytora była formuła zawierająca 3357 zmiennych, 20103 klauzul i 0 błędów. Z testów wynika, że bez problemów daje się pracować z formułami zawierającymi dużą liczbę klauzul i zmiennych: aplikacja działa szybko i nie widać w niej żadnych problemów z wydajnością podczas pracy.

Przeprowadzono kilka testów sprawdzających jak aplikacja działa w zależności od ilości błędów i można powiedzieć, że zależności jest *liniowa*, tzn. im więcej jest błędów, tym wolniejszy staje się proces ich przetwarzania i wyświetlania na ekranie. Skutki tego widać, kiedy błędów jest 4000 i więcej. Edytor zawsze wyświetla całość: setki i tysiące komponentów z numerem linii oraz czerwonych linii podświetlających błędy, nawet jeśli użytkownik je nie widzi. Z jednej strony to jest zaletą ze względu na to, że kiedy wszystkie błędy zostaną znalezione, użytkownik będzie je widział i mógł przemieszczać się po aplikacji płynnie i bez opóźnień. Wadą jest to, że wywoływany na początku długi proces sprawdzenie błędów może powodować zawieszanie programu przez jakiś czas.

3.3.2 Zmiana struktury danych przechowującej błędy

W ostatecznej wersji programu błędy są przechowywane w tablicy jako następujące obiekty:

```
{
    line: number;
    errorCode: 0 | 1 | 2 | 3 | 4 | 5;
    description: string;
    damaged: string;
}
```

W celu polepszenia wydajność aplikacji podczas przetwarzania formuł zawierających dużą liczbę błędów została podjęta próba zmiany struktury danych przechowującej błędy na bardziej pasującą do tych danych. Linia wystąpienia błędu jest unikatowa, z tego powodu wybrano *tablicę haszującą (ang. hash table)*.

W repozytorium z kliencką częścią aplikacji na GitHub [7] stworzono gałąź eksperymentalną, nazywającą się *experimental-map*. Podjęto decyzję zamiany tablicy na obiekt, który będzie przechowywał obiekty, w których kluczem będzie linia wystąpienia błędu, a wartością będzie obiekt zawierający szczegółowe informacje o błędzie. Ten obiekt będzie miał następujący typ danych:

```
{
    [line: string]: {
        errorCode: 0 | 1 | 2 | 3 | 4 | 5;
        description: string;
        damaged: string;
    }
}
```

Każdy obiekt jest tablicą haszującą w języku JavaScript. Zakładano, że po zamianie tablicy na tę strukturę polepszy się średnia złożoność wyszukiwania i usunięcia błędu.

Zrobiono kilka testów i niestety okazało się, że po zmianie struktury danych na tablicę haszującą działanie programu stało się wolniejsze, niż w rozwiązaniu ze zwykłą tablicą, zwłaszcza dla dużych formuł zawierających dużą liczbą błędów.

Przekształcenie kodu programu i wdrążenie nowej struktury danych nie było łatwe, to wymagało dużo zmian w różnych częściach aplikacji. W wielu miejscach kod stał się bardziej skomplikowany i mniej czytelny. Rozwiązanie z tablicą błędów jest czytelniejsze, wydajniejsze i łatwiejsze w implementacji, z tego powodu w ostatecznej wersji zostawiono go.

Podsumowanie

Zaprojektowano i zaimplementowano webowy interfejs graficzny SAT-solvera. Wszystkie założenia aplikacji zostały spełnione. Pozwala ona wygodnie pracować z formułami w postaci DIMACS CNF lub zwykłym CNF oraz znajdować i wyświetlać wartościowania spełniające formułę.

Stworzoną aplikację można by było rozwinąć w następujący sposób:

- Dodać inne rodzaje SAT-solverów, np. rozwiązujący problem MAX-SAT.
- W komponencie do pracy z formułą w postaci CNF można dodać wyszukiwanie potrzebnych klauzul do zmodyfikowania.
- Usprawnić wydajność przetwarzanie dużej ilości błędów w formule w edytorze.

Spis rysunków

Bibliografia

[1] Zhaohui Fu i Sharad Malik. „On Solving the Partial MAX-SAT Problem". W: *Theory and Applications of Satisfiability Testing - SAT 2006*. Red. Armin Biere i Carla P. Gomes. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, s. 252–265. ISBN: 978-3-540-37207-3.

[2] *O formacie DIMACS CNF*. URL: `https://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html`, dostęp 28.06.2023.

[3] *Dokumentacja biblioteki React*. URL: `https://react.dev/`, dostęp 28.06.2023.

[4] *Dokumentacja frameworku FastAPI*. URL: `https://fastapi.tiangolo.com/`, dostęp 28.06.2023.

[5] Alexey Ignatiev, Antonio Morgado i Joao Marques-Silva. „PySAT: A Python Toolkit for Prototyping with SAT Oracles". W: *SAT*. 2018, s. 428–437. DOI: 10.1007/978-3-319-94144-8_26. URL: https://doi.org/10.1007/978-3-319-94144-8_26.

[6] *Czcionka Source Code Pro*. URL: `https://fonts.google.com/specimen/Source+Code+Pro`, dostęp 28.06.2023.

[7] *Repozytorium klienckiej części aplikacji, gałąź experimental-map*. URL: `https://github.com/vvvvvvvector/SAT-solver-graphical-interface-client/tree/experimental-map`, dostęp 28.06.2023.