PWSZ Tarnów

Informatyka

Techniki kompilacji

Tłumacz logiki temporalnej na format provera SPASS

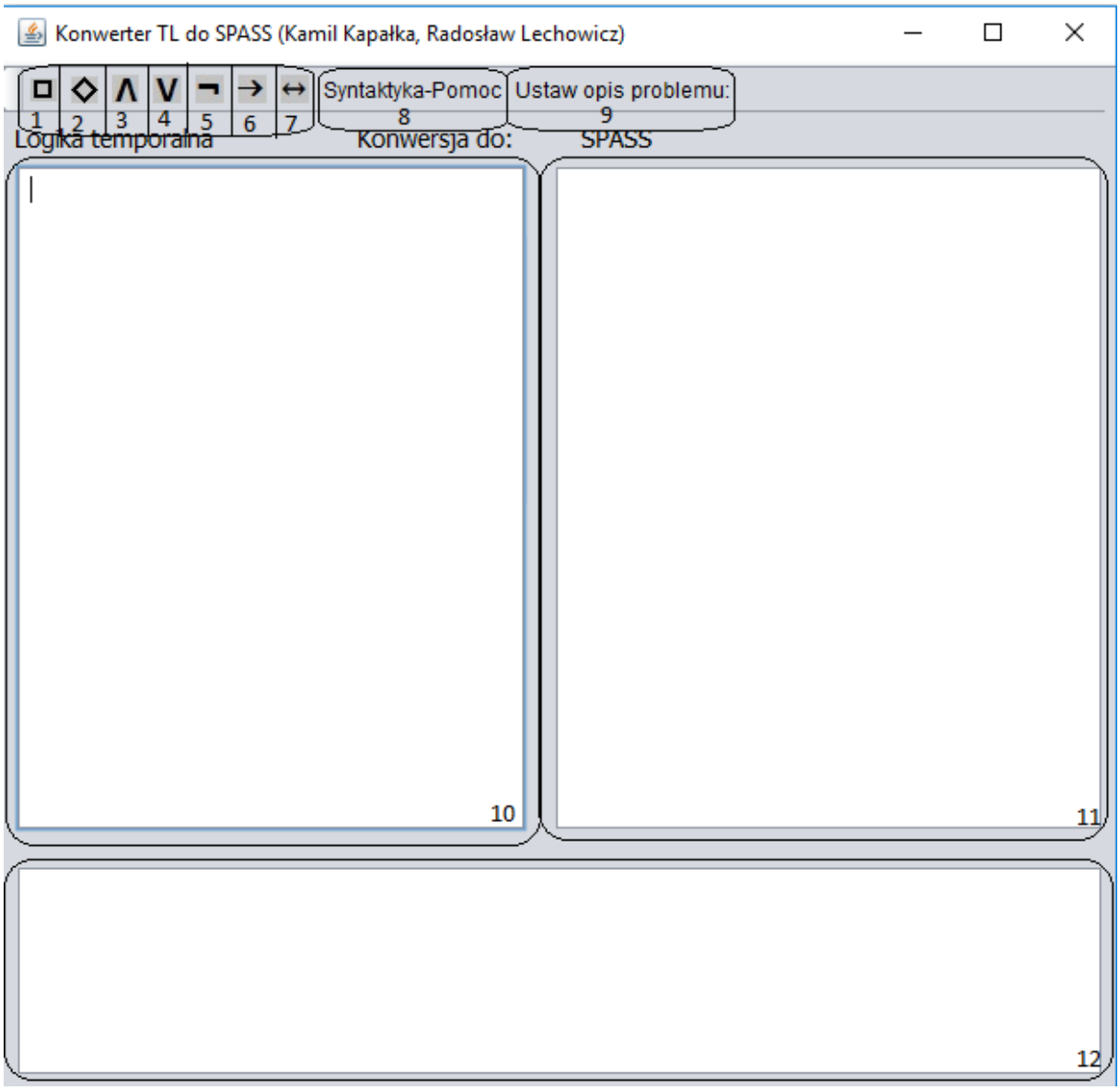
Instrukcja obsługi oraz dokumentacja

Wykonawcy:
Kamil Kapałka
Radosław Lechowicz

Instrukcja obsługi:

Program składa się z dwóch paneli:

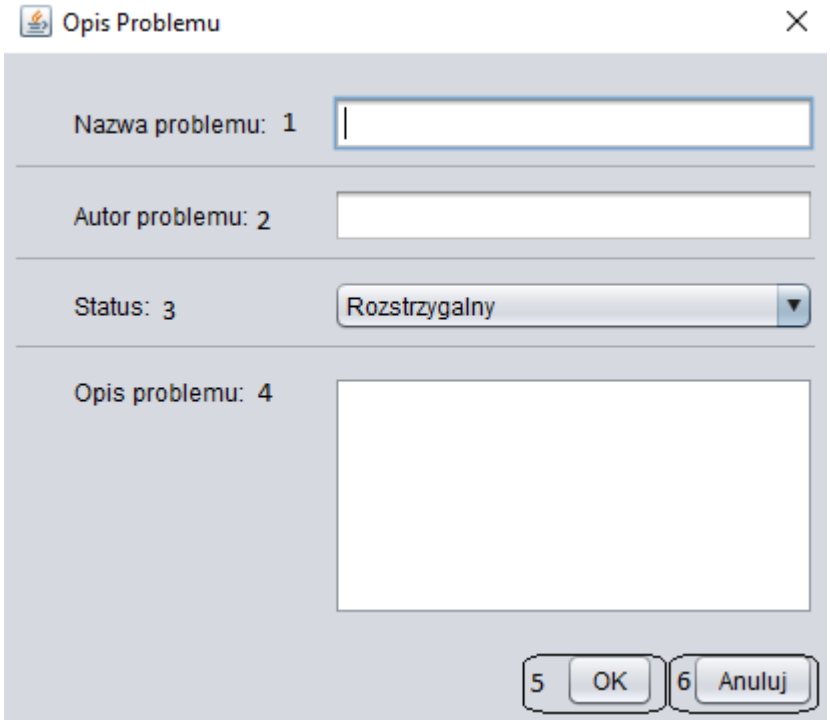
- Panel tłumacza:



  - 1-7 – guziki znaków specjalnych, wprowadzające kolejno:
    - Kwantyfikator „Zawsze”
    - Kwantyfikator „Kiedyś”
    - Znak sumy
    - Znak iloczynu
    - Znak negacji
    - Znak implikacji
    - Znak równoważności

- 8 – Pomoc odnośnie syntaktyki programu, wyświetla komunikat z drobnymi wskazówkami
- 9 – Guzik aktywujący panel opisu problemu
  - Panel opisany w dalszej części instrukcji
- 10 – Panel wprowadzania treści logiki temporalnej
  - W miarę wprowadzania tekstu do tego panelu aktualizuje się panel tekstu przetłumaczonego
- 11 – Panel tekstu przetłumaczonego
  - Zawiera treść problemu przetłumaczoną na poprawną syntaktykę provera SPASS.
- 12 – Panel konsoli, wyświetlający błędy parsera
  - Wyświetla komunikaty o błędach parsera w przypadku wprowadzenia zdań i sformułowań o niepoprawnej syntaktyce

- Panel opisu problemu:



  - 1 – Panel wprowadzania nazwy problemu
  - 2 – Panel wprowadzania autora problemu
  - 3 – Panel wyboru statusu problemu (Rozstrzygalny, Nierozstrzygalny, Nieznany)
  - 4 – Panel wprowadzania opisu problemu
  - 5 – Guzik zatwierdzenia zmian w opisie problemu
  - 6 – Guzik porzucenia zmian w opisie problemu

Jak korzystać z programu:

Aby rozpocząć tłumaczenie, wystarczy wpisać treść logiki temporalnej do panelu wprowadzania treści (nr **10)**. Program automatycznie rozpocznie tłumaczenie. Jednak aby mieć pewność, że syntaktyka jest poprawna oraz że przetłumaczony tekst można wprowadzić do provera SPASS, w panelu konsolowym (nr **12)** nie powinien wyświetlać się żaden komunikat.

Prover SPASS wymaga opisu rozwiązywanych przez siebie problemów. Wartości domyślne w polach dotyczących: nazwy, autora, statusu i opisu problemu są następujące:

- Tytuł - temp
- Autor – temp
- Status – rozstrzygalny
- Opis – temp

Jeżeli chcemy zmienić ten opis oraz wprowadzić swój własny, trzeba nacisnąć guzik „Ustaw opis problemu" (nr **9**). Wyświetli się panel, na którym będzie można te wszystkie wartości zmodyfikować.

Przykłady poprawnej syntaktyki

¬◇(?x,(_Wiedza(?x)∧□(?y,(_Czlowiek(?y)→_InteresujeSie(?y,?x)))))
„Nie będzie nigdy takiej wiedzy, którą człowiek by się zawsze interesował"

□(?x,((_Przejazd(?x)∧_Zamkniety(?x))∧◇(?y,_OtworzySie(?x,?y))))
„Zawsze, gdy przejazd jest zamknięty, to kiedyś (w momencie 'y') on się otworzy"

Syntaktyka logiki temporalnej

Syntaktyka została utworzona za pomocą generatora parserów ANTLR, na podstawie następującej gramatyki:

```
Wyrażenie:   Formuła (koniec_linii Formuła)* koniec_linii* koniec
             |koniec;

Formuła :    L_nawias Formuła Łącznik Formuła P_nawias
             | Nie Formula
             | Kw_Zawsze Lewy_nawias Zmienna separator formuła P_nawias
             | Kw_Kiedyś Lewy_nawias Zmienna separator formuła P_nawias
```

```
              | Predykat L_nawias Termin (separator Termin)* P_nawias
              | Termin równy Termin
              | Formuła '?';

Termin:       Stała
              | Zmienna
              | Funkcja L_nawias Termin (separator Termin)* P_nawias;

Łącznik :     Koniunkcja
              | Dysjunkcja
              | Implikacja
              | Równoważność;

Zmienna :     ‘?’ Słowo;
Predykat :    ‘_’ Słowo;
Stała :       ‘#’ Słowo;
Funkcja :     ‘.’ Słowo;
Słowo:        Charakter+;

L_nawias :    '(' ;
P_nawias :    ')' ;
Separator:    ',' ;

równy:        '=' ;
Nie:          '\u00ac';     (¬)
Zawsze:       '\u25A1';     (◇)
Kiedyś:       '\u25C7';     (□)
fragment Charakter: ('0' .. '9' | 'a' .. 'z' | 'A' .. 'Z');
Koniunkcja:   '\u2227';     (∧)
Dysjunkcja:   '\u2228';     (∨)
Implikacja:   '\u2192';     (→)
Równoważność:'\u2194';      (↔)
Koniec linii:('\r'|'\n')+;
Spacja:       (' '|'\t')+->skip;  (omiń)
```

Dokumentacja

Do wykonania projektu zastosowano generator parserów ANTLR w wersji 4. Projekt zawiera grafiki guzików do interfejsu graficznego oraz 10 plików z kodem:

- 6 plików wygenerowanych przez ANTLR:
  - TL.tokens
  - TLBaseListener.java
  - TLLexer.java
  - TLLexer.tokens
  - TLListener.java
  - TLParser.java
- 4 pliki autorskie:
  - ParserErrorListener.java
  - SettingsDialog.java
  - TL.g4
  - gui.java

TL.tokens

Przechowuje gramatykę parsera - listę niepodzielnych tokenów(18) oraz znaków specjalnych(16), i przypisane do nich numery identyfikacyjne

---

TLBaseListener.java

Według oficjalnej dokumentacji ten plik mógł być zastosowany do implementacji dodatkowych zachowań podczas wprowadzania złożonych tokenów (jak np. Formuła, Łącznik, Zmienna, itp.) W tym programie został zachowany plik domyślny, bez żadnych zachowań specjalnych.

TLLexer.tokens

Przechowuje gramatykę leksera - tą samą zawartość, co plik TL.tokens.

TLListener.java

Według oficjalnej dokumentacji ten plik mógł być zastosowany do utworzenia dodatkowych zachowań podczas przechodzenia przez drzewo parsowania. W tym programie nie został on jednak zmodyfikowany.

TLLexer.java

Lekser logiki temporalnej – składa się z zestawu tablic ze stałymi, m.in.: nazw zasad leksera, ich liczby porządkowej, nazw tokenów i ich wartości, lub przyrostową sieć przejść.

Ponadto zawiera funkcje, dzięki którym program może pozyskiwać wartości z tabel (tabele są prywatne, edycja z zewnątrz jest w ten sposób uniemożliwiona):

- getTokenNames() – zwraca tabelę wartości składających się z dwóch elementów – nazwy i wartości tokenu
- getVocabulary() – zwraca słownik
- getGrammarFileName() – zwraca odnośnik do pliku gramatyki
- getRuleNames() – zwraca tabelę nazw zasad
- getSerializedATN() – zwraca przyrostową sieć przejść (rozszerzona wersja rekurencyjnej sieci przejść) w formie zserializowanej
- getChannelNames() – zwraca tabelę nazw kanałów leksera (są dwa kanały: domyślny i ukryty. Używane są do filtrowania różnych tokenów)
- getModeNames() – zwraca tabelę z nazwami trybów parsowania (jest jeden tryb: domyślny)

TLParser.java

Zawiera to samo, co TLLexer.java, a ponadto jeszcze tabele stałych: liczby porządkowe i nazwy zasad konstrukcji wyrażeń z większej ilości tokenów. Dodatkowo implementuje maszynę stanową oraz klasę kontekstu (pojedynczego wywołania zasady) dla każdego wyrażenia.

ParserErrorListener

Zawiera: listę wartości tekstowych, funkcję zwracającą tą listę, oraz funkcję SyntaxError(), która dodaje do listy nowy komunikat.

Ta klasa została użyta w celu nadpisania domyślnej klasy nasłuchującej błędów, aby móc wszystkie komunikaty wyświetlić na panelu konsolowym.

TL.g4

Zawiera gramatykę logiki temporalnej.

gui.java

Zawiera zmienne:

- tekstowe: nazwa, autor, status i opis problemu
- listy wartości tekstowych: stałe, predykaty, funkcje, aksjomaty, przypuszczenia
- flagowe: czy zdanie zawiera większą niż 1 liczbę łączników
- listę tokenów parsera
- odnośnik do obiektu klasy SettingsDialog

Oraz funkcje:

- **konstruktor()** – generuje pustą listę tokenów, uruchamia program oraz co pół sekundy przeprowadza następującą procedurę: Tworzy listę tokenów na podstawie tekstu wprowadzonego do panelu logiki temporalnej, wyświetla błędy do konsoli i porównuje zmiany od ostatniego wywołania. Jeżeli lista tokenów jest inna, to nadpisuje wszystkie listy wartości tekstowych funkcją CheckStructure(nowa lista tokenów) ora przeprowadza konwersję do SPASSa. Jeżeli zaś tokeny są takie same, to tekst pozostaje niezmieniony (to jest użyteczne np. w celu przekopiowania tekstu SPASSa)
- **Funkcje guzików 1-7** – na koniec wprowadzonego tekstu wstawiająwybrany symbol specjalny
- **SyntaxHelpActionPerformed()** – wyświetla komunikat zawierający pomoc odnośnie syntaktyki
- **SettingsButtonActionPerformed()** – tworzy i wyświetla okno dialogowe SettingsDialog, służące do ustawiania własnego opisu problemu
- **CheckStructure(lista tokenów)** – Funkcja najpierw dzieli wprowadzony tekst na zdania. Jeżeli po drodze napotka stałą, dodaje jej nazwę do tabeli ze stałymi. Jeśli napotka znak końca linii, rozpoczyna nowe zdanie. Następnie w każdym zdaniu szuka funkcji oraz predykatów (oznaczonych odpowiednimi symbolami początkowymi). Jeżeli napotka, to szuka pozycji najbliższego prawego nawiasu (w celu określenia liczby argumentów) oraz wywołuje funkcję CheckPredicateFunction(lista tokenów, pozycja początkowego i końcowego tokenu, czy sprawdzany jest predykat czy funkcja). Po odnalezieniu wszystkich predykatów i funkcji przechodzi jeszcze raz przez wszystkie zdania, tym razem wywołując za każdym razem funkcję CheckAxiomConjecture(lista tokenów, token początkowy i końcowy zdania)
- **CheckPredicateFunction() –** dodaje odpowiednio do list predykatów lub funkcji nową wartość tekstową – nazwę i odpowiadającą jej liczbę argumentów. W celu określenia liczby argumentów zbiera wszystkie znaczniki stałej bądź zmiennej wewnątrz predykatu/funkcji. Następnie funkcja sprawdza, czy w liście istnieje już egzemplarz takiego predykatu/funkcji – jeżeli tak, to funkcja nie dodaje do listy żadnych nowych wartości. Jeśli nie, predykat/funkcja z liczbą argumentów jest dodana do listy.
- **CheckAxiomConjecture()** – Na początku przeprowadza podstawowe formatowanie zdania: podmienia wszystkie znaki specjalne na ich odpowiedniki w SPASSie, a jeśli napotka zmienne tuż po kwantyfikatorze (jak np. ◇`(?x,?y)`) to ujmuje je w klamry. Sprawdza czy na końcy zdania jest znak '?' – on oznacza, że zdanie jest przypuszczeniem. Zbiera pozycje wszystkich znaków '¬', 'Λ','V','→','↔', gdyż dla wyrażeń następujących po nich trzeba przeprowadzić specjalne formatowanie funkcją Modify(). Po zakończeniu formatowania dodaje zdanie odpowiednio do listy aksjomatów lub przypuszczeń.
- **Modify()** – Na początku określa zakres wszystkich znaków wyspecyfikowanych w CheckAxiomConjecture(). Jest to konieczne, gdyż w syntaktyce SPASSa funkcje odpowiadające tym znakom nie są operatorami (jak np. dodawanie – a+b), tylko funkcjami argumentowymi (np. dodaj(a,b)). Idąc token po tokenie w tył, zliczając przy tym pary lewych i prawych nawiasów. Ten lewy nawias, który wystąpił przed jakimkolwiek prawym nawiasem i wywołuje przewagę lewych nad prawymi o 1 jest wybierany jako początek zakresu znaku. Analogicznie postępuje idąc w przód – szuka prawego nawiasu który nastąpił po jakimkolwiek lewyn nawiasie i powoduje

równowagę nawiasów. Mając zakres znaku, można przetworzyć operator na funkcję argumentową.
- **Convert()** – pobiera zawartość list oraz wartości tekstowych i konwertuje je na treść sytaktyki SPASSa

---

SettingsDialog.java

Okno modyfikacji opisu problemu. Zawiera odnośnik do głównej aplikacji, trzy panele tekstowe ProblemText, AuthorText, DescriptionText oraz jeden panel wyboru StatusCombo, i dwie funkcje:

- CancelButtonActionPerformed() – porzuca wszystkie naniesione zmiany i zamyka okno.
- ConfirmButtonActionPerformed() – ustawia zmienne tekstowe w głównej aplikacji na te wybrane w oknie modyfikacji opisu problemu, potem zamyka okno.

Bibliografia:

http://www.antlr.org/ - strona główna generatora parserów ANTLR

https://www.mpi-inf.mpg.de/departments/automation-of-logic/software/spass-workbench/ - strona główna provera SPASS

https://pl.wikipedia.org/wiki/ANTLR

https://pl.wikipedia.org/wiki/Przyrostowa_sie%C4%87_przej%C5%9B%C4%87

http://staff.iiar.pwr.wroc.pl/pawel.gluchowski/wp-content/uploads/miasi/logika_w1.pdf

https://pl.wikipedia.org/wiki/Rachunek_predykat%C3%B3w_pierwszego_rz%C4%99du