PAŃSTWOWA WYŻSZA SZKOŁA W TARNOWIE

Techniki kompilacji

Prover9

GUI oraz dokumentacja

Prowadzący:
Dr inż. Radosław Klimek

Autorzy:
Damian Cop
Damian Fiołek

Tarnów 2019

Spis treści

1. Opis

Prover9 – jest to zautomatyzowane narzędzie udowadniające dla logiki pierwszego rzędu stworzone przez Williama McCune'a.

Prover9 jest następcą Ottera, innego narzędzia udowadniającego, również stworzonego przez McCune'a. *Prover9* jest doceniany za relatywnie czytelne dowody, jak i za przydatne podpowiedzi.

Standardowa teoria logiczna: „Wszyscy ludzie są śmiertelni", „Sokrates jest człowiekiem" udowadnia, że „Sokrates jest śmiertelny".

```
formulas(assumptions).
  man(x) -> mortal(x).   % open formula with free variable x
  man(socrates).
end_of_list.
```

2. Prover9 vs Windows

Dawniej, *Prover9* był możliwy do instalacji w systemie Windows, jednak od jakiegoś czasu z oficjalnej strony usunięto archiwa pozwalające na to. Jedyną opcją uruchomienia wyżej wspomnianego provera jest użycie linuksowych archiwów, dostępnych na oficjalnej stronie. Archiwa działają na każdej dystrybucji Linuxa, wykorzystana przez nas została dystrybycja o nazwie Ubuntu.

3. Instalacja

W celu poprawnego działania programu, należy zainstalować bibliotekę ***LADR*** (dołączoną do projektu). Zawiera ona całą logikę działania i bez niej używanie programu byłoby niemożliwe.

Nadmienić również należy, że cały program wymaga zainstalowanego środowiska Java Runtime Environment w wersji 8.

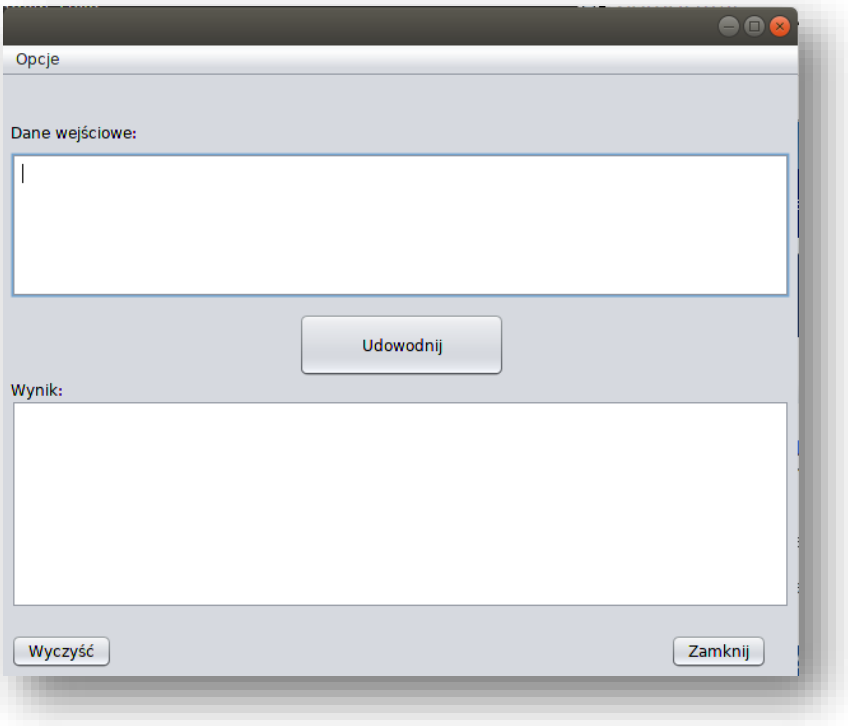
Po uprzednim zainstalowaniu wszystkich potrzebnych bibliotek, uruchamiamy środowisko ***Netbeans***, wczytujemy projekt z folderu *„prover9”* i uruchamiamy go.

4. GUI

GUI (*graphical user interface*) – jest to interfejs graficzny, ułatwiający komunikację użytkownika z *proverem*. GUI zostało wykonane w języku programowania Java, ze względu na łatwość implementacji oraz możliwość tworzenia przyjaznych użytkownikowi interfejsów. Jest to również język, który bezproblemowo działa pod środowiskiem *Linux*.

4.1 Okno główne

Okno główne aplikacji przedstawia się następująco:



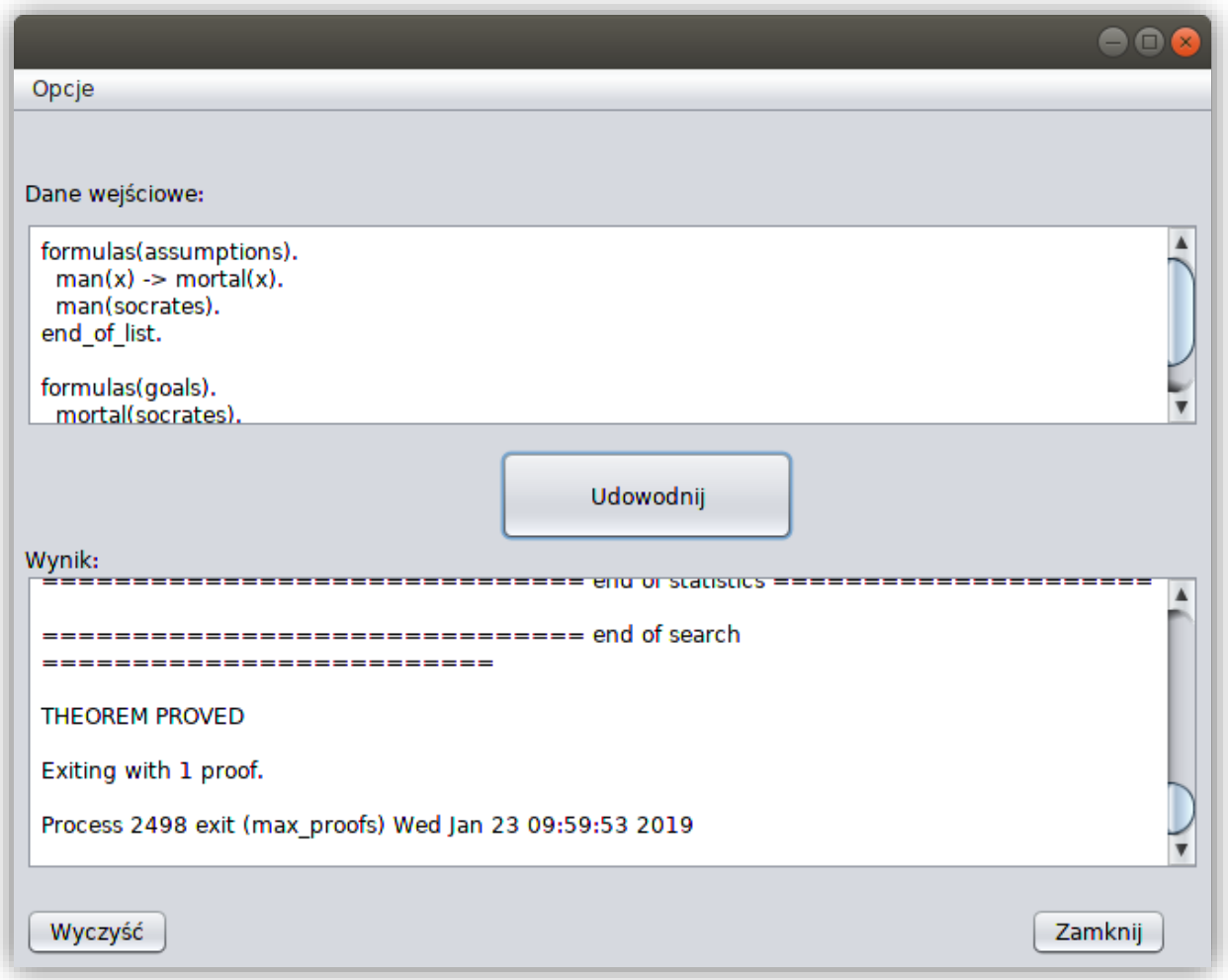
Okno główne zostało zaprojektowane w jak najprostszy sposób, aby korzystanie z provera było jak najbardziej intuicyjne. Zawarto w nim najważniejsze funkcje potrzebne, aby skorzystać z provera.

Użytkownik posiada dwie opcje do wprowadzenia teorii przygotowanej do udowodnienia. Użytkownik ma również możliwość skorzystania z teorii przykładowej, która to opcja znajduje się w wysuwanym menu na górze ekranu.

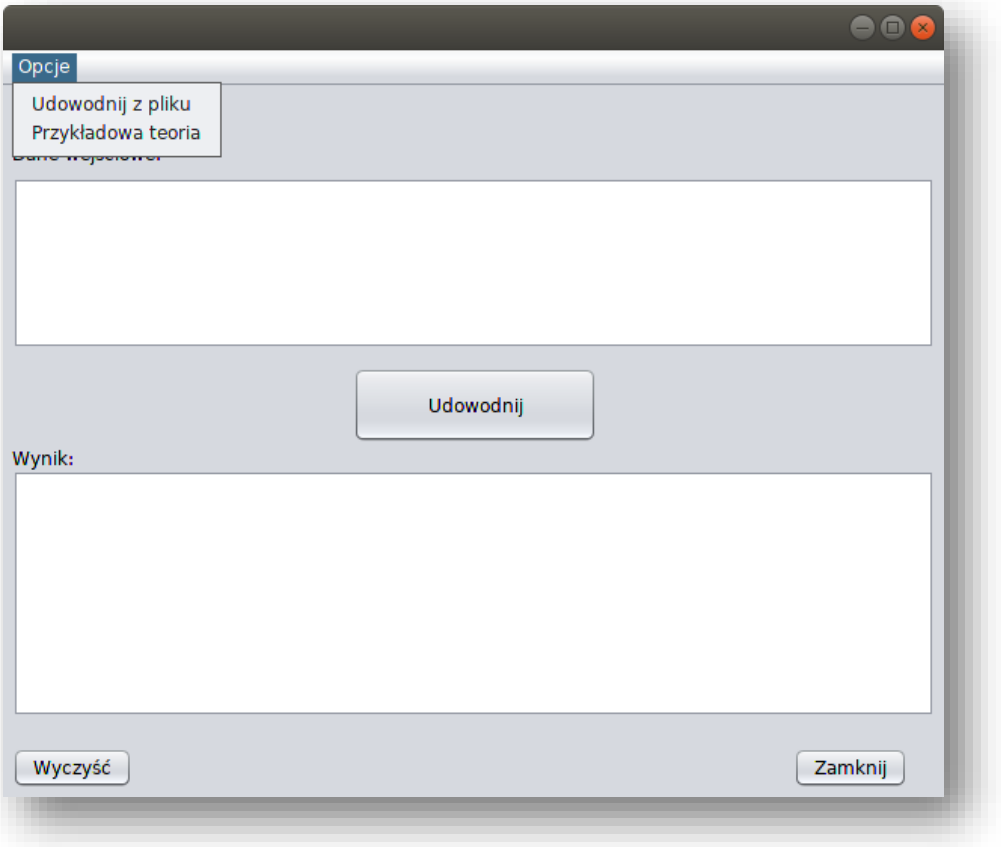
4.2 Pierwszy sposób wprowadzania

Użytkownik wprowadza dane do okienka „Dane wejściowe:”. Aby uzyskać wynik teorii, klikamy w przycisk „Udowodnij” i oczekujemy na wykonanie się próby udowodnienia teorii.

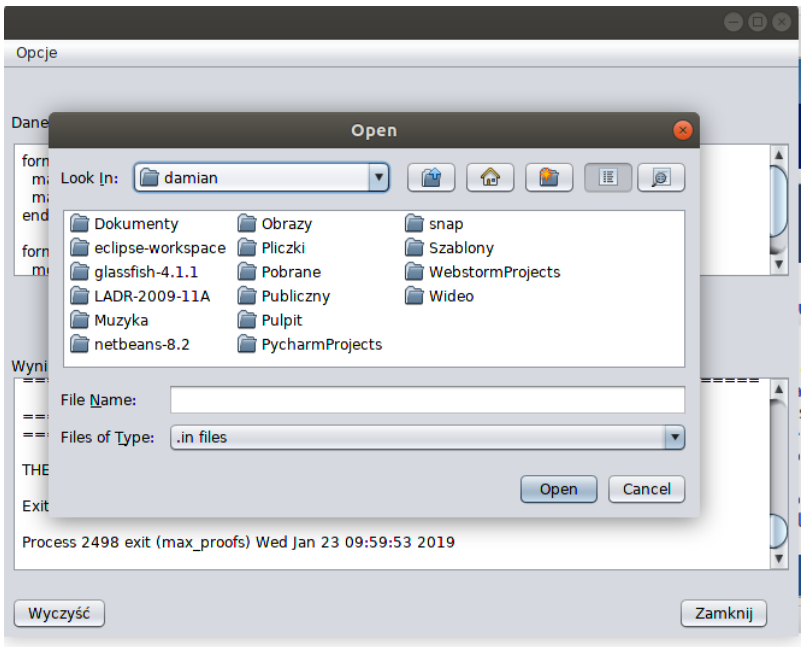
Wynik działania pojawi się w okienku „Wynik:”.



4.3 Drugi sposób wprowadzania



Innym sposobem przekazania teorii do udowodnienia jest użycie opcji „Wczytaj z pliku”. Znajduje się ona w menu „Opcje” na samym szczycie ekranu. Po kliknięciu w w/w opcję, wyświetlone zostanie okno wyboru pliku. Program obsługuje pliki w formacie „.in.”



4.4 Przykładowa teoria

„Przykładowa teoria” służy jako swoisty samouczek zarówno języka rozumianego przez Prover9, jak i prezentacja, na jakiej zasadzie działa proces rozwiązywania teorii.

```
formulas(sos).
  all x all y (subset(x,y) <-> (all z (member(z,x) -> member(z,y)))).
end_of_list.

formulas(goals).
  all x all y all z (subset(x,y) & subset(y,z) -> subset(x,z)).
end_of_list.
```

5. Przykładowe teorie

5.1 Podstawowy przykład

formulas(assumptions).

man(x) -> mortal(x).
man(socrates).

end_of_list.
formulas(goals).

mortal(socrates).

end_of_list.

Szybsze rozwiązanie w/w problemu:

```
    formulas(sos).
-man(x) | mortal(x).
man(socrates).
-mortal(socrates).
    end_of_list.
```

5.2 Czy koza jest istotą rozumną (homosapiens)?

```
    formulas(assumptions).
man(x) -> homosapiens(x).
animal(x) -> -man(x).
animal(goat).
   end_of_list.
   formulas(goals).
homosapiens(goat).
    end_of_list.
```

5.3 Pierwiastek kwadratowy z 2 jest irracjonalny

```
formulas(assumptions).
   1*x = x. % identity
   x*y = y*x. % commutativity
   x*(y*z) = (x*y)*z. % associativity
   ( x*y = x*z ) -> y = z. % cancellation (0 is not allowed, so x!=0).

    %  Now let's define divides(x,y): x divides y.
    %  Example: divides(2,6) is true b/c 2*3=6.
    %  divides(x,y) <-> (exists z x*z = y).
       divides(2,x*x) -> divides(2,x).
    % If 2 divides x*x, it divides x.
       a*a = 2*(b*b).
   % a/b = sqrt(2), so a^2 = 2 * b^2.
   (x != 1) ->  -(divides(x,a) & divides(x,b)).
   % a/b is in lowest terms
   2 != 1. % Original author almost forgot this.
end_of_list.
```

6. Prędkość działania

W ogólnym rozrachunku Prover9 rozwiązuje dane założenia logiczne w dobrym tempie, istotna jest jednak ilość formuł. Zaznaczyć również trzeba, że nie nadweręża on specjalnie procesora. Podczas testów możliwe było w pełni normalne korzystanie z komputera, nawet w przypadku rozwiązywania formuł w ilości parudziesięciu tysięcy.

Prover9 nie obciąża mocno maszyny, na której pracuje, lecz uzyskanie wyniku potrafi być czasochłonne.

6.1 Formuły proste

Formuły, które nie zawierają wiele założeń oraz celów logicznych, wykonują się od kilku do kilkunastu sekund. Przy liczbie założeń mniejszej niż 10 osiągnąć można wyniki rzędu kilku milisekund.

6.2 Formuły złożone

W przypadku formuł złożonych czas wykonania potrafi już oscylować w granicach kilku do kilkunastu minut. Nietrudnym jest osiągnąć limit czasowy, dlatego należy zakładać dostateczny margines zapasu czasu.

6.3 Prover9 vs SPASS

Z racji podobieństwa składni logicznej obsługiwanej przez oba *provery*, postanowiliśmy dokonać analizy porównawczej prędkości wykonania danych zadań. Przygotowaliśmy zadania, które zostały podzielone ze względu na ilość formuł do wykonania. Poniżej przedstawiona została tabelka porównawcza.

Ważna informacja: jedna generacja = 3 formuły logiczne!

Liczba generacji	10	100	500	1000	5000
Czas Rozwiązania Prover9	0.01s	0.09s	1.23s	4.67	170.23s
Czas Rozwiązania SPASS	0.01s	0.06s	0.68s	2.40s	56.85s
Mnożnik szybkości SPASS vs Prover9	1x	1.5x	1.8x	1.94x	2.99x
Ilość użytych klauzul	17	21	21	21	21
Problem rozwiązany	Dowód znaleziony	Dowód znaleziony	Dowód znaleziony	Dowód znaleziony	Dowód znaleziony

Jak wynika z powyższej tabeli, SPASS jest w stanie szybciej policzyć zadane mu formuły. Jest to najbardziej odczuwalne przy liczbie generacji powyżej 1000.

7. Obserwacje własne

Oprócz testów, których wyniki pokazane zostały powyżej, spróbowaliśmy również sprawdzić limit ilości oraz złożoności formuł, które Prover9 jest w stanie policzyć. Był on w stanie swobodnie pracować do ilości formuł sięgającej 15-20 tysięcy. Okazuje się jednak, że osiągnięcie rozwiązania z 30 tysięcy formuł okazało się już niemożliwe na naszym sprzęcie, gdyż limit buforu dla *provera* został osiągnięty i nie były możliwe dalsze obliczenia.

Największym atutem *provera* jest jego niski poziom obciążenia systemu, gdyż nawet przy 20 tysiącach formuł, nie wystąpiły żadne spowolnienia pracy komputera i było możliwe zostawienie procesu rozwiązywania zadania w tle, jednocześnie wykonując inne operacje.