
Państwowa Wyższa Szkoła Zawodowa w Tarnowie

# Instytut Politechniczny

**Kierunek: Informatyka**
**Specjalizacja: Inżynieria Systemów informatycznych**
**Rok: 2017/2018**

## Dokumentacja

# PWSZ Prover9 Insert Zautomatyzowane Dowodzenie Twierdzeń I rzędu

**Wykonawcy:**
Dawid Miczek
Dawid Karwat

Tarnów 2017

# Wprowadzenie

Program ‘PWSZ Prover9 Insert’ do działania wykorzystuje zewnętrzne oprogramowanie do automatycznego dowodzenia twierdzeń o nazwie Prover9 przeznaczone na platformę Unix. Poprawne korzystanie z programu wymaga najpierw zaopatrzenie się w środowisko ‘Prover9’, aby to uczynić należy uruchomić konsolę i zainstalować oprogramowanie zgodnie z poniższą instrukcją:

**sudo apt-get install prover9**

Źródło do pełnej dokumentacji twórców programu Prover9:

**https://www.cs.unm.edu/~mccune/mace4/manual/2009-11A/**

# Pojęcia

**Klauzula** – zbiór formuł logicznych. Klauzulę nazywamy prawdziwą wtedy i tylko wtedy, gdy alternatywa jej formuł logicznych jest prawdziwa. Klauzula pusta jest zawsze fałszywa.

**logika pierwszego rzędu -** termin ten nazywa się też krótko rachunkiem kwantyfikatorów.

**Kwantyfikator** – termin przyjęty w matematyce i logice na oznaczenie zwrotów: dla każdego, istnieje i im pokrewnych, a także odpowiadającym im symbolom wiążącym zmienne w formułach.

# Czym jest Prover9?

Prover9 jest to system służący do dowodzenia twierdzeń w logice pierwszego rzędu. Został on opracowany przez Williama McCune'a, który pierwszą jego wersję opublikował w roku 2005 jako następcę systemu Otter. Prover9 został stworzony razem z systemem Mace4, który szuka skończonych modeli formuł pierwszego rzędu. Oba programy mogą działać jednocześnie dla tych samych danych wejściowych. Kiedy Mace4 próbuje znaleźć kontrprzykład dla zadanych tez, to Prover9 stara się znaleźć ich dowód. Oba programy i wiele innych narzędzi) są zbudowane w oparciu o bibliotekę LADR, aby uprościć implementację.

Język jakim posługuje się Prover9 został pokazany na przykładzie, w którym na podstawie dwóch zdań chcemy udowodnić trzecie. Jeżeli założymy, że każdy człowiek jest śmiertelny oraz, że Sokrates był człowiekiem to możemy wywnioskować, że Sokrates był również śmiertelny. W systemie Prover9 będzie to zakodowane w taki sposób:

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

formulas(goals).
  mortal(socrates).
end_of_list.
```

Kod zostanie zamieniony na klauzule, a cel zostanie zanegowany. Jeśli system wykaże sprzeczność, twierdzenie zostanie udowodnione:

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

# Dane wejściowe do Prover9

Poprawne działanie ‘Prover9’ wymaga uściślonego wejścia danych, które należy przygotować tak, aby program był w stanie je poprawnie odczytać. Dlatego koniecznie jest przeprowadzenie tłumaczenia z języka potocznego na taki który zrozumie środowisko.

*Tabela zawiera kwantyfikatory, które akceptuje Prover9

| Znaczenie | Symbol | Przykład |
|---|---|---|
| negacja | `-` | `(-p)` |
| alternatywa | `\|` | `(p \| q \| r)` |
| koniunkcja | `&` | `(p & q & r)` |
| implikacja | `->` | `(p -> q)` |
| implikacja wsteczna | `<-` | `(p <- q)` |
| równowartość | `<->` | `(p <-> q)` |
| dla każdego | `all` | `(all x all y p(x,y))` |
| istnieje | `exists` | `(exists x exists y p(x,y))` |

W celu wykonania tłumaczenia z języka potocznego na taki, który zrozumie prover9 posłużyliśmy się słownikiem danych zaprezentowanym poniżej:

| Język Potoczny | Tłumaczenie |
|---|---|
| „dla kazdego” | „all” |
| „wszystkie” | „all” |
| „istnieje” | „exists” |
| „i” | „&” |
| „lub” | „\|” |
| „nieprawda ze” | „-” |
| „jezeli x to/jest y” | „->” |
| „rownowazne” | „<->” |
| „rowna sie” | „=” |

# Prezentacja danych wejściowych

W celu szerszego zaprezentowania danych wejściowych jakie uznaje Prover9 posłużymy się przykładem twierdzenia o królestwie zwierząt.

Zakładamy, że istnieje kilka zwierząt które żywią się roślinami lub/oraz zwierzętami mniejszymi od nich samych. W naszym przykładzie poszukujemy zwierzęcia które zjada inne mniejsze zwierzę które z kolei jest roślinożerne.

Założenia w języku potocznym:

formula(zalozenia) .

jezeli Wilk(x) jest zwierze(x) .
jezeli Lis() jest zwierze(x) .
jezeli Ptak(x) jest zwierze(x) .
jezeli Gasienica(x) jest zwierze(x) .
jezeli Slimak(x) jest zwierze(x) .
jezeli Ziarno(x) jest roslina(x) .

istnieje x Wilk(x) .
istnieje x Lis(x) .
istnieje x Ptak(x) .
istnieje x Gasienica(x) .
istnieje x Slimak(x) .
istnieje x Ziarno(x) .

dla kazdego x (zwierze(x) jest ( dla kazdego y (roslina(y) jest jadalna(x,y)) ) lub ( dla kazdego z ( zwierze(z) i mniejsze(z,x) i ( istnieje u (roslina(u) i jadalna(z,u))) jest jadalna(x,z)))) .

Gasienica(x) i Ptak(y) jest mniejsze(x,y) .
Slimak(x) i Ptak(y) jest mniejsze(x,y) .
Ptak(x) i Lis(y) jest mniejsze(x,y) .
Lis(x) i Wilk(y) jest mniejsze(x,y) .
Ptak(x) i Gasienica(y) jest jadalna(x,y) .

jezeli Gasienica(x) jest ( istnieje y (roslina(y) i jadalna(x,y))) .
jezeli Slimak(x) jest ( istnieje y (roslina(y) i jadalna(x,y))) .
jezeli Wilk(x) i Lis(y) jest nieprawda ze jadalna(x,y) .
jezeli Wilk(x) i Ziarno(y) jest nieprawda ze jadalna(x,y) .
jezeli Ptak(x) i Slimak(y) jest nieprawda ze jadalna(x,y) .
koniec .

formula(cele) .
istnieje x istnieje y ( zwierze(x) i zwierze(y) i jadalna(x,y) i ( wszystkie z ( jezeli Ziarno(z) jest jadalna(y,z)))) .
koniec .

Powyższe dane zostają przefiltrowane przez program ‘PWSZ Prover9 Insert’, a następnie zamienione na takie które zaakceptuje i zrozumie Prover9:

Dane po przefiltrowaniu:

formulas(assumptions) .

Wilk(x) -> zwierze(x) .
Lis(x) -> zwierze(x) .
Ptak(x) -> zwierze(x) .
Gasienica(x) -> zwierze(x) .
Slimak(x) -> zwierze(x) .
Ziarno(x) -> roslina(x) .
exists x Wilk(x) .
exists x Lis(x) .
exists x Ptak(x) .
exists x Gasienica(x) .
exists x Slimak(x) .
exists x Ziarno(x) .
all x (zwierze(x) -> ( all y (roslina(y) -> jadalna(x,y)) ) | ( all z ( zwierze(z) & mniejsze(z,x) & ( exists u (roslina(u) & jadalna(z,u))) -> jadalna(x,z)))) .
Gasienica(x) & Ptak(y) -> mniejsze(x,y) .
Slimak(x) & Ptak(y) -> mniejsze(x,y) .
Ptak(x) & Lis(y) -> mniejsze(x,y) .
Lis(x) & Wilk(y) -> mniejsze(x,y) .
Ptak(x) & Gasienica(y) -> jadalna(x,y) .
Gasienica(x) -> ( exists y (roslina(y) & jadalna(x,y))) .
Slimak(x) -> ( exists y (roslina(y) & jadalna(x,y))) .
Wilk(x) & Lis(y) -> - jadalna(x,y) .
Wilk(x) & Ziarno(y) -> - jadalna(x,y) .
Ptak(x) & Slimak(y) -> - jadalna(x,y) .
end_of_list .
formulas(goals) .
exists x exists y ( zwierze(x) & zwierze(y) & jadalna(x,y) & ( all z ( Ziarno(z) -> jadalna(y,z)))) .
end_of_list .

Przefiltrowanie danych nastąpiło zgodnie z powyższym słownikiem danych, który „wychwytuje” odpowiednie wyrazy, a następnie je tłumaczy na odpowiedniki logiczne

# Interpretacja danych wyjściowych

Prover9 po otrzymaniu przygotowanych wcześniej danych wejściowych zaczyna wykonywać swoją pracę i poszukuje dowodu, a następnie całą statystykę z powrotem odbieramy w naszym programie. Wszystko co zostało wyliczone oraz podstawowe informacje wyświetlą się w polu tekstowym ***Output:***

**Z racji dużej rozległości otrzymanych danych poniżej został zaprezentowany jedynie wycinek odpowiadający za dowód.*

```
============================== prooftrans ============================

Prover9 (64) version 2009-11A, November 2009.
Process 25539 was started by dawid on dawid-P553UJ,
Sun Jan 21 14:19:53 2018
The command was "prover9 -f prover_input.in".
============================== end of head ===========================


============================== end of input ==========================


============================== PROOF =================================


% -------- Comments from original proof --------
% Proof 1 at 0.03 (+ 0.00) seconds.
% Length of proof is 74.
% Level of proof is 10.
% Maximum clause weight is 13.000.
% Given clauses 44.


1 Wilk(x) -> zwierze(x) # label(non_clause). [assumption].
2 Lis(x) -> zwierze(x) # label(non_clause). [assumption].
3 Ptak(x) -> zwierze(x) # label(non_clause). [assumption].
5 Slimak(x) -> zwierze(x) # label(non_clause). [assumption].
```

6 Ziarno(x) -> roslina(x) # label(non_clause). [assumption].
7 (exists x Wilk(x)) # label(non_clause). [assumption].
8 (exists x Lis(x)) # label(non_clause). [assumption].
9 (exists x Ptak(x)) # label(non_clause). [assumption].
11 (exists x Slimak(x)) # label(non_clause). [assumption].
12 (exists x Ziarno(x)) # label(non_clause). [assumption].
13 (all x (zwierze(x) -> (all y (roslina(y) -> jadalna(x,y))) | (all z (zwierze(z) & mniejsze(z,x) & (exists u (roslina(u) & jadalna(z,u))) -> jadalna(x,z))))) # label(non_clause). [assumption].
15 Slimak(x) & Ptak(y) -> mniejsze(x,y) # label(non_clause). [assumption].
16 Ptak(x) & Lis(y) -> mniejsze(x,y) # label(non_clause). [assumption].
17 Lis(x) & Wilk(y) -> mniejsze(x,y) # label(non_clause). [assumption].
20 Slimak(x) -> (exists y (roslina(y) & jadalna(x,y))) # label(non_clause). [assumption].
21 Wilk(x) & Lis(y) -> -jadalna(x,y) # label(non_clause). [assumption].
22 Wilk(x) & Ziarno(y) -> -jadalna(x,y) # label(non_clause). [assumption].
23 Ptak(x) & Slimak(y) -> -jadalna(x,y) # label(non_clause). [assumption].
24 (exists x exists y (zwierze(x) & zwierze(y) & jadalna(x,y) & (all z (Ziarno(z) -> jadalna(y,z))))) # label(non_clause) # label(goal). [goal].
25 Wilk(c1). [clausify(7)].
26 -Wilk(x) | zwierze(x). [clausify(1)].
27 -Lis(x) | -Wilk(y) | mniejsze(x,y). [clausify(17)].
28 -Wilk(x) | -Lis(y) | -jadalna(x,y). [clausify(21)].
29 -Wilk(x) | -Ziarno(y) | -jadalna(x,y). [clausify(22)].
30 Lis(c2). [clausify(8)].
31 -Lis(x) | zwierze(x). [clausify(2)].
32 -Ptak(x) | -Lis(y) | mniejsze(x,y). [clausify(16)].
33 -Lis(x) | mniejsze(x,c1). [resolve(27,b,25,a)].
34 -Lis(x) | -jadalna(c1,x). [resolve(28,a,25,a)].
35 Ptak(c3). [clausify(9)].
36 -Ptak(x) | zwierze(x). [clausify(3)].
38 -Slimak(x) | -Ptak(y) | mniejsze(x,y). [clausify(15)].
40 -Ptak(x) | -Slimak(y) | -jadalna(x,y). [clausify(23)].
41 -Ptak(x) | mniejsze(x,c2). [resolve(32,b,30,a)].
48 Slimak(c5). [clausify(11)].
49 -Slimak(x) | zwierze(x). [clausify(5)].
50 -Slimak(x) | roslina(f2(x)). [clausify(20)].
51 -Slimak(x) | jadalna(x,f2(x)). [clausify(20)].
52 -Slimak(x) | mniejsze(x,c3). [resolve(38,b,35,a)].
53 -Slimak(x) | -jadalna(c3,x). [resolve(40,a,35,a)].
54 Ziarno(c6). [clausify(12)].
55 -Ziarno(x) | roslina(x). [clausify(6)].
56 -zwierze(x) | -zwierze(y) | -jadalna(x,y) | Ziarno(f3(x,y)). [deny(24)].
57 -Ziarno(x) | -jadalna(c1,x). [resolve(29,a,25,a)].
58 mniejsze(c2,c1). [resolve(33,a,30,a)].
59 -zwierze(x) | -roslina(y) | jadalna(x,y) | -zwierze(z) | -mniejsze(z,x) | -roslina(u) | -jadalna(z,u) | jadalna(x,z). [clausify(13)].
60 mniejsze(c3,c2). [resolve(41,a,35,a)].

62 mniejsze(c5,c3). [resolve(52,a,48,a)].
63 -zwierze(x) | -zwierze(y) | -jadalna(x,y) | -jadalna(y,f3(x,y)). [deny(24)].
64 zwierze(c1). [resolve(25,a,26,a)].
65 zwierze(c2). [resolve(30,a,31,a)].
66 -jadalna(c1,c2). [resolve(34,a,30,a)].
67 zwierze(c3). [resolve(35,a,36,a)].
72 zwierze(c5). [resolve(48,a,49,a)].
73 roslina(f2(c5)). [resolve(50,a,48,a)].
74 jadalna(c5,f2(c5)). [resolve(51,a,48,a)].
75 -jadalna(c3,c5). [resolve(53,a,48,a)].
76 roslina(c6). [resolve(54,a,55,a)].
77 -zwierze(x) | -zwierze(y) | -jadalna(x,y) | roslina(f3(x,y)). [resolve(56,d,55,a)].
78 -jadalna(c1,c6). [resolve(57,a,54,a)].
80 -zwierze(c1) | -roslina(x) | jadalna(c1,x) | -zwierze(c2) | -roslina(y) | -jadalna(c2,y) | jadalna(c1,c2). [resolve(58,a,59,e)].
81 -roslina(x) | jadalna(c1,x) | -roslina(y) | -jadalna(c2,y). [copy(80),unit_del(a,64),unit_del(d,65),unit_del(g,66)].
82 -zwierze(c2) | -roslina(x) | jadalna(c2,x) | -zwierze(c3) | -roslina(y) | -jadalna(c3,y) | jadalna(c2,c3). [resolve(60,a,59,e)].
83 -roslina(x) | jadalna(c2,x) | -roslina(y) | -jadalna(c3,y) | jadalna(c2,c3). [copy(82),unit_del(a,65),unit_del(d,67)].
84 -zwierze(c3) | -roslina(x) | jadalna(c3,x) | -zwierze(c5) | -roslina(y) | -jadalna(c5,y) | jadalna(c3,c5). [resolve(62,a,59,e)].
85 -roslina(x) | jadalna(c3,x) | -roslina(y) | -jadalna(c5,y). [copy(84),unit_del(a,67),unit_del(d,72),unit_del(g,75)].
90 -roslina(x) | jadalna(c2,x) | -jadalna(c3,x) | jadalna(c2,c3). [factor(83,a,c)].
103 -jadalna(c2,c6). [ur(81,a,76,a,b,78,a,c,76,a)].
109 -roslina(x) | jadalna(c3,x). [resolve(85,d,74,a),unit_del(c,73)].
113 jadalna(c3,c6). [resolve(109,a,76,a)].
117 jadalna(c2,c3). [resolve(113,a,90,c),unit_del(a,76),unit_del(b,103)].
124 roslina(f3(c2,c3)). [resolve(117,a,77,c),unit_del(a,65),unit_del(b,67)].
125 -jadalna(c3,f3(c2,c3)). [resolve(117,a,63,c),unit_del(a,65),unit_del(b,67)].
133 $F. [resolve(124,a,109,a),unit_del(a,125)].

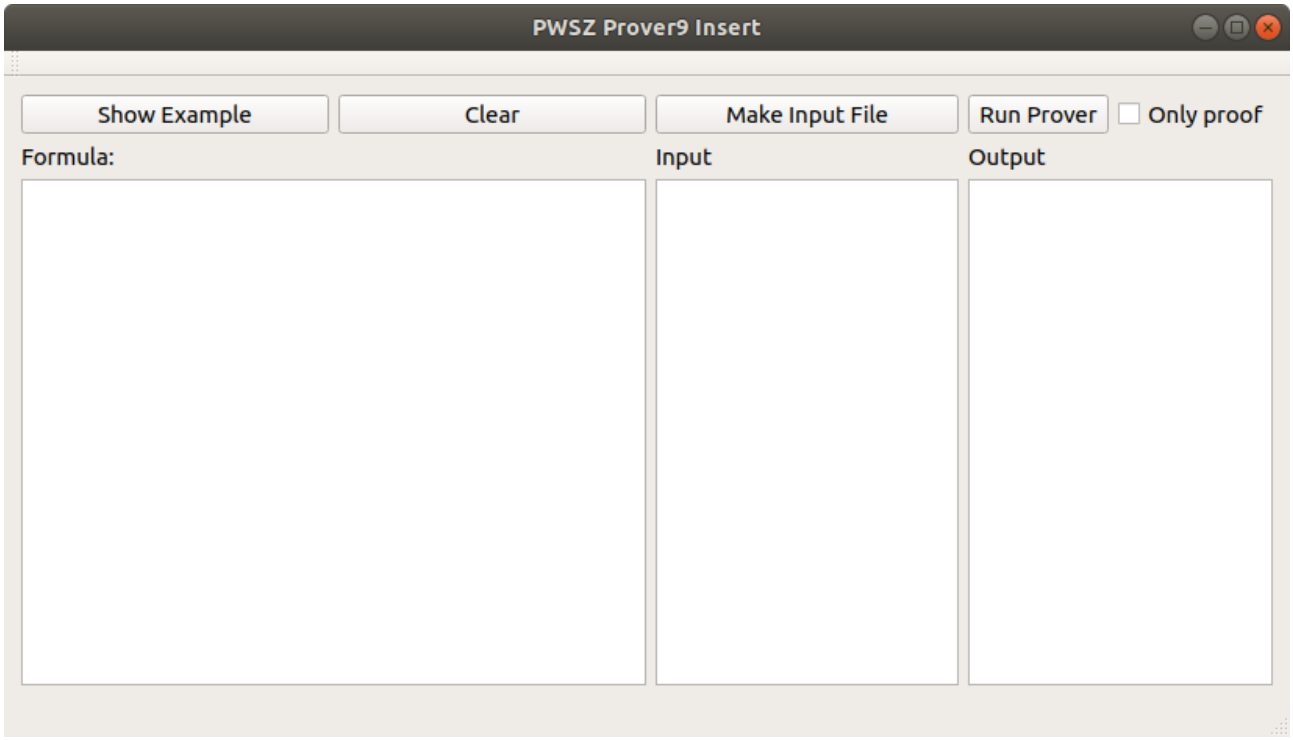
========== end of proof ==========

Prover9 próbuje odszukać dowód na podstawie negowania wcześniej wprowadzonych przez nas założeń. W wyniku sprzeczności zostaje udowodnione to co zostało wprowadzone jako cel. W powyższym przypadku odnaleźliśmy zwierzę które zjada mniejsze od siebie i żywiące się roślinami.

Interpretacja nie jest łatwa ponieważ trzeba zapoznać się z interpretacją programu Prover9, który przypisuje sobie własne klauzule i tak jak powyżej zmienne c1,c2,c3… oznaczają kolejne zwierzęta po czym można stwierdzić, że Ptak zjada gąsienice, a ta żywi się roślinnością.

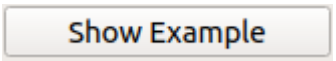
```
============================== PROCESS INITIAL CLAUSES ===============

% Clauses before input processing:

formulas(usable).
end_of_list.

formulas(sos).
-Wilk(x) | zwierze(x). [clausify(1)].
-Lis(x) | zwierze(x). [clausify(2)].
-Ptak(x) | zwierze(x). [clausify(3)].
-Gasienica(x) | zwierze(x). [clausify(4)].
-Slimak(x) | zwierze(x). [clausify(5)].
-Ziarno(x) | roslina(x). [clausify(6)].
Wilk(c1). [clausify(7)].
Lis(c2). [clausify(8)].
Ptak(c3). [clausify(9)].
Gasienica(c4). [clausify(10)].
Slimak(c5). [clausify(11)].
Ziarno(c6). [clausify(12)].
-zwierze(x) | -roslina(y) | jadalna(x,y) | -zwierze(z) | -mniejsze(z,x) | -roslina(u) |
-jadalna(z,u) | jadalna(x,z). [clausify(13)].
-Gasienica(x) | -Ptak(y) | mniejsze(x,y). [clausify(14)].
-Slimak(x) | -Ptak(y) | mniejsze(x,y). [clausify(15)].
-Ptak(x) | -Lis(y) | mniejsze(x,y). [clausify(16)].
-Lis(x) | -Wilk(y) | mniejsze(x,y). [clausify(17)].
-Ptak(x) | -Gasienica(y) | jadalna(x,y). [clausify(18)].
-Gasienica(x) | roslina(f1(x)). [clausify(19)].
-Gasienica(x) | jadalna(x,f1(x)). [clausify(19)].
-Slimak(x) | roslina(f2(x)). [clausify(20)].
-Slimak(x) | jadalna(x,f2(x)). [clausify(20)].
-Wilk(x) | -Lis(y) | -jadalna(x,y). [clausify(21)].
-Wilk(x) | -Ziarno(y) | -jadalna(x,y). [clausify(22)].
-Ptak(x) | -Slimak(y) | -jadalna(x,y). [clausify(23)].
-zwierze(x) | -zwierze(y) | -jadalna(x,y) | Ziarno(f3(x,y)). [deny(24)].
-zwierze(x) | -zwierze(y) | -jadalna(x,y) | -jadalna(y,f3(x,y)). [deny(24)].
end_of_list.

formulas(demodulators).
end_of_list.
```

# Opis interfejsu programu

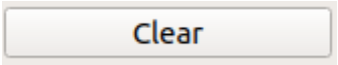



Poniżej został przedstawiony interfejs programu ‘PWSZ Prover9 Insert’ tuż po jego uruchomieniu:

## Przyciski:



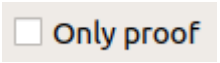

Przycisk ten powoduje wyświetlenie prostego przykładu danych wejściowych w polu tekstowym ‘Formula’




Przycisk ten odpowiada za wyczyszczenie wszystkich pól tekstowych.


Przycisk ten odpowiada za pobranie przetłumaczonych już danych wejściowych z pola ‘Input’, a następnie utworzenie pliku wejściowego z rozszerzeniem ‘.in’ i uzupełnieniem jego zawartości w/w danymi.


Przycisk ten ma za zadanie uruchomić zewnętrzny proces ‘Prover9’ z odpowiednimi parametrami, a jednym z nich jest utworzony w poprzednim kroku plik wejściowy.




Kontrolka ta odpowiada za wyświetlenie informacji w polu tekstowym ‘Output’. Jej zaznaczenie powoduje wyświetlenie samego dowodu, natomiast brak zaznaczenia powoduje wyświetlenie wszystkich informacji odebranych od Prover9.

## Pola tekstowe:

***Formula*** – pole tekstowe, które służy do wprowadzenia przez użytkownika danych przeznaczonych do sprawdzenia. Należy wprowadzać dane w języku potocznym zgodnie z słownikiem danych. Każde wprowadzone słowo zostaję od razu przefiltrowane oraz wyświetlone w polu tekstowym ‘Input’.

***Input*** – pole tekstowe, które wyświetla przetłumaczone dane wejściowe z języka potocznego na taki który zrozumie zewnętrzny program ‘Prover9’

***Output*** – pole tekstowe, które odbiera od zewnętrznego programu ‘Prover9’ dane z obliczeń przeprowadzonych w celu potwierdzenia bądź obalenia pierwotnej teorii wprowadzonej przez użytkownika.