SPASS THEOREM PROVER

Marcel Lekston
Filip Prasałek

Dokumentacja end-to-end

1. Czym jest prover SPASS i jak działa ?

1.1. Wstęp

SPASS Theorem Prover jest narzędziem do automatycznego dowodzenia twierdzeń, należących do rachunku predykatów pierwszego rzędu. Proces ten polega na rozstrzyganiu czy twierdzenie jest *dowodliwe.* Każdy krok dowodu realizowanego przez prover SPASS musi jasno wynikać albo z poprzedniego kroku, albo z przyjętego przez użytkownika **aksjomatu.**

Narzędzie to zostało stworzone i udostępnione przez naukowców z Max-Planck Institute for Informatics w Niemczech. Obecnie, wersja 3.9 provera jest dostępna do pobrania pod poniższym adresem:

https://www.mpi-inf.mpg.de/departments/automation-of-logic/software/spass-workbench/classic-spass-theorem-prover/download/

1.2. Wejście

Prover akceptuje input w postaci odpowiednio przekształconych formuł logiki pierwszego rzędu. Każdy problem zaczyna się od podania kilku **informacji** na jego temat - nazwy, autora, statusu oraz opisu.

```
begin_problem(Driver).

list_of_descriptions.
  name({*Driver and car*}).
  author({*FP & ML*}).
  status(unsatisfiable).
  description({* Car will never move unless driver is inside *}).
end_of_list.
```

Następnie, w liście symboli, wpisujemy wszystkie znane nam niestandardowe **predykaty i funkcje.** Podajemy tutaj wszystkie kwantyfikatory oraz operatory, które nie są predefiniowane i z których będziemy korzystać w dalszej części. Zadeklarowane symbole nie mogą się powtarzać. Przykładowo, możemy tutaj sformułować predykaty umożliwiające logiczny opis problemu:

```
list_of_symbols.
  functions[(t1,0),(t2,0)].
  predicates[(Autostoi,1),(Kierowcawaucie,1),(Greaterequal,2)].
end_of_list
```

Korzystając z wyżej wymienionych aksjomatów, oraz standardowych kwantyfikatorów i operatorów, jesteśmy w stanie podać listę znanych nam **aksjomatów**, co do których słuszności mamy pewność

```
list_of_formulae(axioms).
  formula(Greaterequal(t2,t1)).
  formula(not(Kierowcawaucie(t1))).

formula(forall([d,t],implies(and(Greaterequal(d,t),not(Kierowcawaucie(t))),Autostoi(d))
)).
end_of_list.
```

Wreszcie, analogicznie jak wcześniej, w postaci formuł, podawane są wszystkie **przypuszczenia**, które chcemy udowodnić:

```
list_of_formulae(conjectures).
  formula(Autostoi(t2)).
end_of_list.
```

Na końcu, możemy dodać do programu odpowiednie **flagi**, regulujące różne parametry działania programu, jak i samego outputu, który oczekujemy. Możliwości jest bardzo wiele, a kilka przykładowych to:

- **-Memory=*n***
  Reguluje zasób pamięci dostępny dla SPASSa
- **-Loops**
  Określa maksymalną liczbę iteracji
- **-RSST**
  Włącza/wyłącza zasadę Obvious Reduction
- **-DocSST**
  Włącza/wyłącza dokumentowanie typowania w outpucie

Opcje te dopisujemy za pomocą następującej klauzuli:

```
list_of_settings(SPASS).
  set_flag(Loops, 1).
  set_flag(RSST, 1).
end_of_list.

end_problem.
```

1.3. Działanie provera SPASS

Po przekazaniu wejścia wedle zasad opisanych w powyższym punkcie następuje proces wnioskowania. Celem tego procesu jest wykazanie przez SPASS że **koniunkcja wszystkich aksjomatów** dostarczonych przez użytkownika **implikuje dysjunkcję wszystkich przypuszczeń.**

SPASS parsuje input użytkownika i sprowadza go do **koniunkcyjnej postaci normalnej**, czyli takiej, która jest równoważna formule wejściowej, lecz zapisana jest tylko i wyłącznie w postaci koniunkcji klauzul (usuwane są wszystkie implikacje, równoważności etc.). Koniunkcje te zostają wtedy zanegowane, ponieważ SPASS dowodzi prawdziwości poprzez negacje.

Następnie następuje analiza przekształconego problemu, której rezultatem jest dobranie odpowiednich opcji, które pozwolą na odpowiednią analizę problemu. SPASS na tym etapie sprawdza charakter dostarczonych aksjomatów i przypuszczeń (np. czy do funkcji można podstawić nieskończenie wiele wartości, czy istnieją w formule klauzule Horna), a następnie na ich podstawie może zadecydować że posortuje klauzule w dany sposób, użyje danych wewnętrznych interfejsów do redukcji, czy uproszczania formuł itp.

Użytkownik jest informowany o wyniku powyższej analizy na wyjściu programu, co przejawia się poprzez następujący przykładowy output:

1.

```
This is a monadic Non-Horn problem without equality.
This is a problem that has, if any, a finite domain model.
There are no function symbols.
This is a problem that contains sort information.
```

2.

```
Inferences: IORe=1 IOFc=1
Reductions: RObv=1 RTaut=1 RFSub=1 RBSub=1
Extras: No Input Saturation, No Selection, No Splitting, Full
Reduction,  Ratio: 5, FuncWeight: 1, VarWeight: 1
Precedence: div > id > c^1 > temp_succ > temp_shield >
temp_zero > Autostoi > Kierowcawaucie
Ordering  : KBO
```

Gdzie pierwsza część (1.) to wynik analizy charakterystyki problemu przez SPASS, a druga (2.) to lista opcji wybranych przez prover. Przykładowo w powyższym przykładzie w sekcji Extras w (2.) mamy informacje że SPASS będzie szukał dowodu dla danego problemu poprzez jego pełną redukcję (Full Reduction).

To pokazuje olbrzymią zaletę provera SPASS - **dobiera on strategię dowodzenie w zależności od charakterystyki dostarczonego problemu.**

1.4. Interpretacja wyjścia

Dowodzenie problemu może skończyć się na 3 różne sposoby:

- Dowód został znaleziony, wtedy na wyjściu pojawi się komunikat: `SPASS beiseite: Proof found`
- Jeżeli formuła zawiera błędy semantyczne, lub nie da się dowieść poprawności danego przypuszczenia, na wyjściu pojawi się komunikat `SPASS beiseite: Completion found.` lub program ulegnie zawieszeniu bez wypisania ostatecznego rezultatu
- W razie błędu syntaktycznego - informacje w której linijce takowy został popełniony

Oprócz informacji o tym czy udało się dowieść twierdzenia, możemy poprzez przekazanie odpowiedniej flagi do provera (*-DocProof*) uzyskać pełen dowód wyprodukowany przez SPASS. Dla rozważanego wyżej przykładu wygląda on w sposób następujący:

```
Here is a proof with depth 2, length 7 :
1[0:Inp] || Autostoi(t2)*+ -> .
2[0:Inp] ||  -> Greaterequal(t2,t1)*.
3[0:Inp] || Kierowcawaucie(t1)*+ -> .
4[0:Inp] || Greaterequal(U,V)* -> Kierowcawaucie(V)
Autostoi(U).
5[0:Res:4.1,1.0] || Greaterequal(t2,U)*+ ->
Kierowcawaucie(U).
6[0:Res:2.0,5.0] ||  -> Kierowcawaucie(t1)*.
7[0:MRR:6.0,3.0] ||  -> .
Formulae used in the proof : conjecture0 axiom0 axiom1
axiom2
```

Powyższy dowód można zinterpretować w następujący sposób:

- Pierwsze 4 punkty są to dostarczone przez nas aksjomaty (oznaczone [0:Inp]), które zostały użyte w znalezieniu dowodu
- Punkty 5 i 6 to pośrednie kroki wnioskowania provera które prowadzą do znalezienia dowodu w punkcie 7 (koniec redukcji)

Dodatkowo ostatnia linijka informuje nas o tym które formuły (numerowane zgodnie z kolejnością wprowadzania w liście formuł) **po kolei** posłużyły do znalezienia dowodu przez provera SPASS.

2. Działanie naszego programu

2.1. Składnia

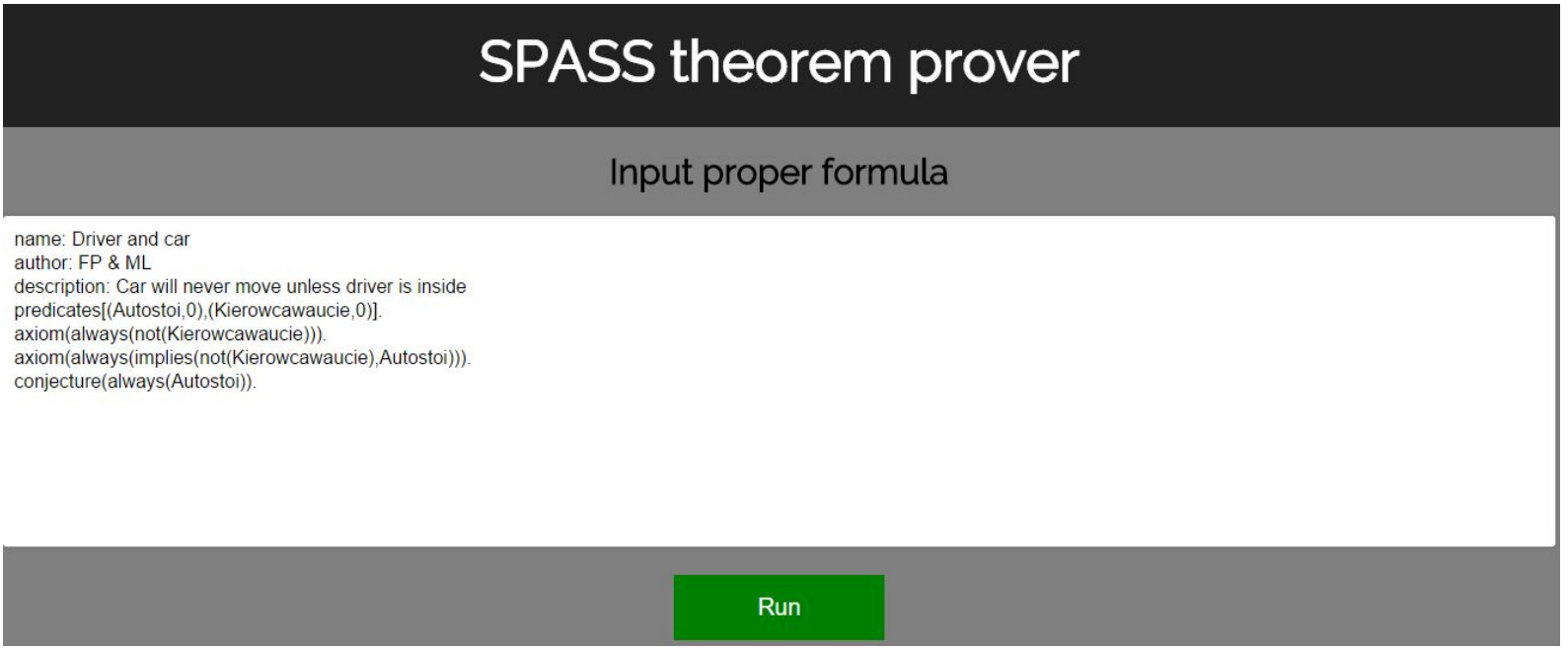
W celu uproszczenia wprowadzania formuł do programu, wprowadziliśmy następujące słowa kluczowe, które zastępują te używane w natywnej wersji provera SPASS:

- **name -** nazwa omawianego problemu
- **author -** autor problemu
- **description -** opis problemu
- **predicates -** zdefiniowane predykaty
- **functions -** zdefiniowane funkcje
- **axiom -** aksjomat
- **conjecture -** przypuszczenie

2.2. Przykładowy input

Stosując powyższe reguły, w naszym programie możemy wprowadzić formuły w następującej postaci:

```
======================== TEMPORAL DRIVER ============================
name: Driver and car
author: FP & ML
description: Car will never move unless driver is inside
predicates[(Autostoi,0),(Kierowcawaucie,0)].
axiom(not(Kierowcawaucie)).
axiom(always(implies(not(Kierowcawaucie),Autostoi))).
conjecture(Autostoi).
```



Po naciśnięciu przycisku run, zaproponowany przez nas format jest konwertowany do formatu akceptowalnego przez SPASS, a następnie poddawany dowodzeniu.

2.3. Konwersja logiki temporalnej do pierwszego rzędu

Nasz program umożliwia nie tylko udowadnianie twierdzeń *Logiki Pierwszego Rzędu* (FOL), ale również podstawowych twierdzeń *Liniowej Logiki Temporalnej* (LTL).

Logiką temporalną możemy nazwać logika umożliwiająca rozważanie zależności czasowych bez wprowadzania czasu explicite. Czas można wprowadzić do zwykłego rachunku predykatów pierwszego rzędu, co czynimy w naszym programie, ze względu na fakt, że prover SPASS akceptuje tylko FOL.

Istnieje oczywiście możliwość odwzorowania operatorów LTL przy pomocy zwykłych operatorów, charakterystycznych dla FOL. My skupiliśmy się na dwóch podstawowych operatorach LTL, czyli *sometime* oraz *always*. Do konwersji LTL do FOL stosujemy następujące reguły:

$$p \quad \rightsquigarrow \quad p(t)$$

$$\Diamond p \quad \rightsquigarrow \quad \exists t'.\ (t' \geq t) \wedge p(t')$$

$$\Box p \quad \rightsquigarrow \quad \forall t'.\ (t' \geq t) \Rightarrow p(t')$$

gdzie

rowcy w samochodzie". Wiedząc, że warunkiem koniecznym aby samochód znajdował się w ruchu, jest to, że człowiek musi się w nim znajdować. Na podstawie tego możemy wywnioskować, że jeżeli człowiek nie jest w środku, to samochód **nigdy** nie ruszy. Teza, sformułowana w postaci logiki temporalnej, wygląda następująco:

$\Diamond \rightsquigarrow F$ sometime in the Future

$\Box \rightsquigarrow G$ Globally in the future

Prover SPASS w wersji 3.9 nie akceptuje operatora większości dlatego aby odwzorować ów operator w języku zrozumiałym dla provera, przy konwersji LTL do FOL definiujemy własny predykat ***Greaterthan*** przyjmujący dwa argumenty (dwa miejsca w czasie), dzięki czemu możemy przekonwertować wejście w ten sposób, o czym więcej w następnym punkcie.

2.4. Przykładowa konwersja

Analogicznie do poprzedniego przykładu, rozpatrujemy problem "kierowcy w samochodzie"

$$\neg r \wedge G_f$$

gdzie:

- r - kierowca w aucie
- f - auto stoi

Stosując wyżej wymienione reguły, otrzymujemy następującą formułę:

$$\neg r(t) \wedge (\forall_d (d \geq t)) \implies f(d)$$

Wpisując do naszej aplikacji następujące wejście, będące odwzorowaniem powyższego problemu w zaproponowanym przez nas w punkcie 2.1 formacie:

```
name: Driver and car
author: FP & ML
description: Car will never move unless driver is inside
predicates[(Autostoi,0),(Kierowcawaucie,0)].
axiom(not(Kierowcawaucie)).
axiom(always(implies(not(Kierowcawaucie),Autostoi))).
conjecture(Autostoi).
```

Po uruchomieniu jest przekształcane do poniższego formatu, który jest zrozumiały dla provera SPASS:

```
begin_problem(Driver).

list_of_descriptions. name({*Driver and car*}).
  author({*FP & ML*}).
  status(unsatisfiable).
        description({* Car will never move unless driver is
inside *}).
end_of_list.
```

```
list_of_symbols.
  functions[(t1,0),(t2,0)].
  predicates[(Autostoi,1),(Kierowcawaucie,1),(Greaterequal,2)].
end_of_list.

list_of_formulae(axioms).
  formula(Greaterequal(t2,t1)).
  formula(not(Kierowcawaucie(t2))).
formula(forall([d,t],implies(Greaterequal(d,t),implies(not(Kierow
cawaucie(d)),Autostoi(d))))).
end_of_list.

list_of_formulae(conjectures).
  formula(Autostoi(t2)).
end_of_list.

end_problem.
```

Jak widać w *list_of_symbols* pojawił się wcześniej wspomniany predykat *Greaterequal/2* , który następnie aby faktycznie odwzorowywał dokładnie działanie arytmetycznego operatora większości musi być pierwotnie użyty jako aksjomat wyznaczając dwa punkty w czasie, które następnie będą używane w modelowaniu problemu.

Wywołanie to skutkuje znalezieniem dowodu przez prover i wyprodukowaniem następującego wyjścia, które można zinterpretować zgodnie z zasadami opisanymi w punkcie 1.4 Interpretacja wyjścia:

```
-------------------------SPASS-START-----------------------------
Input Problem:
1[0:Inp] || Autostoi(t2)* -> .
2[0:Inp] ||  -> Greaterequal(t2,t1)*.
3[0:Inp] || Kierowcawaucie(t1)* -> .
4[0:Inp] || Greaterequal(U,V)* -> Autostoi(U) Kierowcawaucie(V).
 This is a first-order Non-Horn problem without equality.
 This is a problem that has, if any, a finite domain model.
 There are no function symbols.
 The conjecture is ground.
 Axiom clauses: 3 Conjecture clauses: 1
 Inferences: IORe=1 IOFc=1
 Reductions: RFMRR=1 RBMRR=1 RObv=1 RUnC=1 RTaut=1 RFSub=1 RBSub=1 RCon=1
 Extras : Input Saturation, Always Selection, Full Splitting, Full Reduction,  Ratio:
5, FuncWeight: 1, VarWeight: 1
 Precedence: Autostoi > Kierowcawaucie > Greaterequal > t1 > t2
 Ordering  : KBO
Processed Problem:

Worked Off Clauses:

Usable Clauses:
```

```
2[0:Inp] ||  -> Greaterequal(t2,t1)*.
1[0:Inp] || Autostoi(t2)* -> .
3[0:Inp] || Kierowcawaucie(t1)* -> .
5[0:Res:4.1,1.0] || Greaterequal(t2,U)* -> Kierowcawaucie(U).
4[0:Inp] || Greaterequal(U,V)* -> Kierowcawaucie(V) Autostoi(U).
       Given clause: 2[0:Inp] ||  -> Greaterequal(t2,t1)*.
       Given clause: 1[0:Inp] || Autostoi(t2)*+ -> .
       Given clause: 3[0:Inp] || Kierowcawaucie(t1)*+ -> .
       Given clause: 5[0:Res:4.1,1.0] || Greaterequal(t2,U)*+ -> Kierowcawaucie(U).
SPASS V 3.5
SPASS beiseite: Proof found.
Problem: car.dfg
SPASS derived 2 clauses, backtracked 0 clauses, performed 0 splits and kept 6 clauses.
SPASS allocated 45939 KBytes.
SPASS spent    0:00:00.02 on the problem.
               0:00:00.01 for the input.
               0:00:00.01 for the FLOTTER CNF translation.
               0:00:00.00 for inferences.
               0:00:00.00 for the backtracking.
               0:00:00.00 for the reduction.


Here is a proof with depth 2, length 7 :
1[0:Inp] || Autostoi(t2)*+ -> .
2[0:Inp] ||  -> Greaterequal(t2,t1)*.
3[0:Inp] || Kierowcawaucie(t1)*+ -> .
4[0:Inp] || Greaterequal(U,V)* -> Kierowcawaucie(V) Autostoi(U).
5[0:Res:4.1,1.0] || Greaterequal(t2,U)*+ -> Kierowcawaucie(U).
6[0:Res:2.0,5.0] ||  -> Kierowcawaucie(t1)*.
7[0:MRR:6.0,3.0] ||  -> .
Formulae used in the proof : conjecture0 axiom0 axiom1 axiom2
--------------------------SPASS-STOP-----------------------------
```

2.5. Wywołanie provera

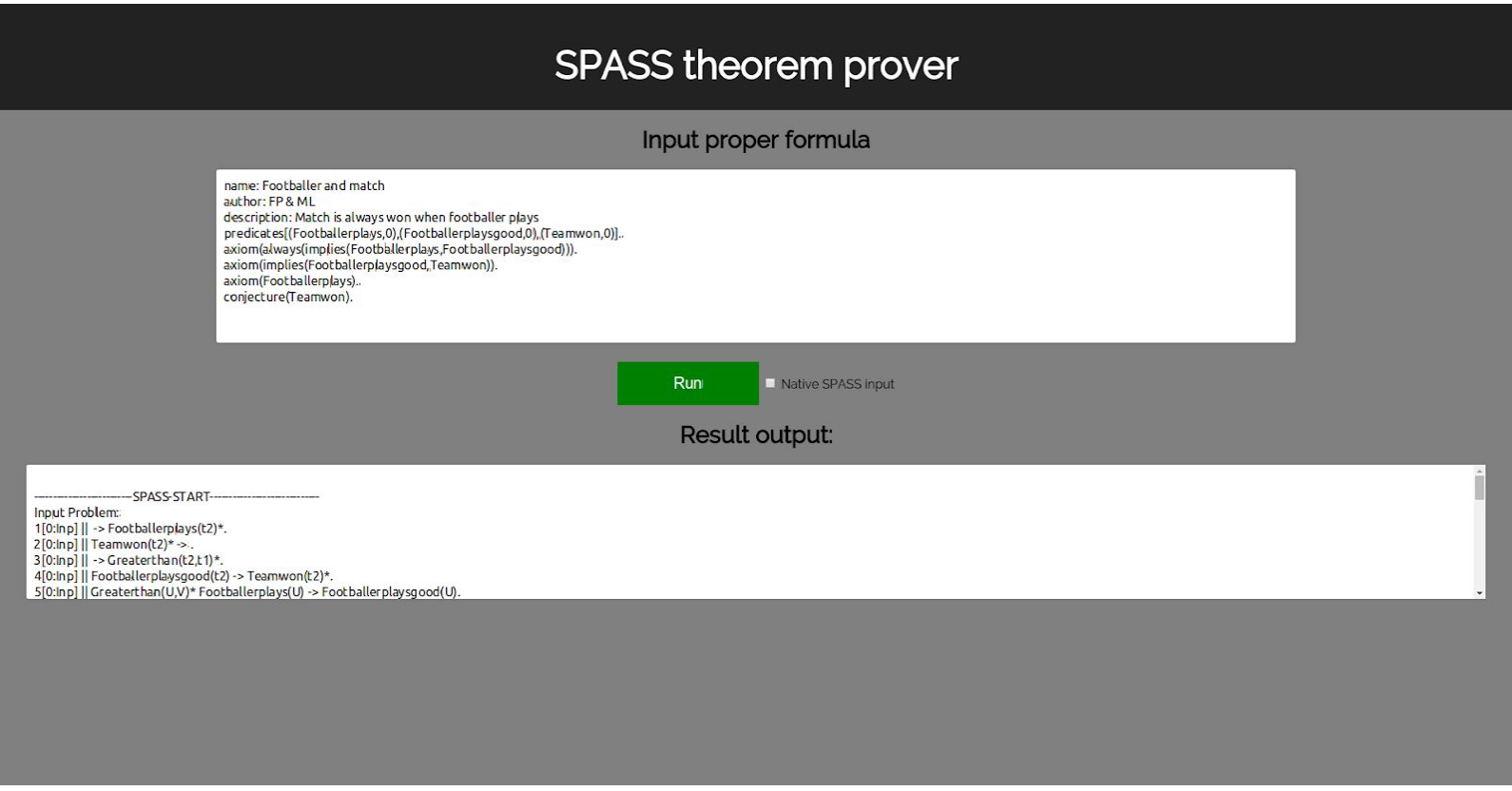
Po naciśnięciu przycisku Run w naszym interfejsie graficznym, uruchamiany w tle jest prover SPASS z flagą -Stdin, a na jego standardowe wejście przekazujemy sparsowany input. Jest to możliwe dzięki funkcji *exec()*, pochodzącej z modułu *child_process* z Node.js umożliwiającego tworzenie nowych procesów.

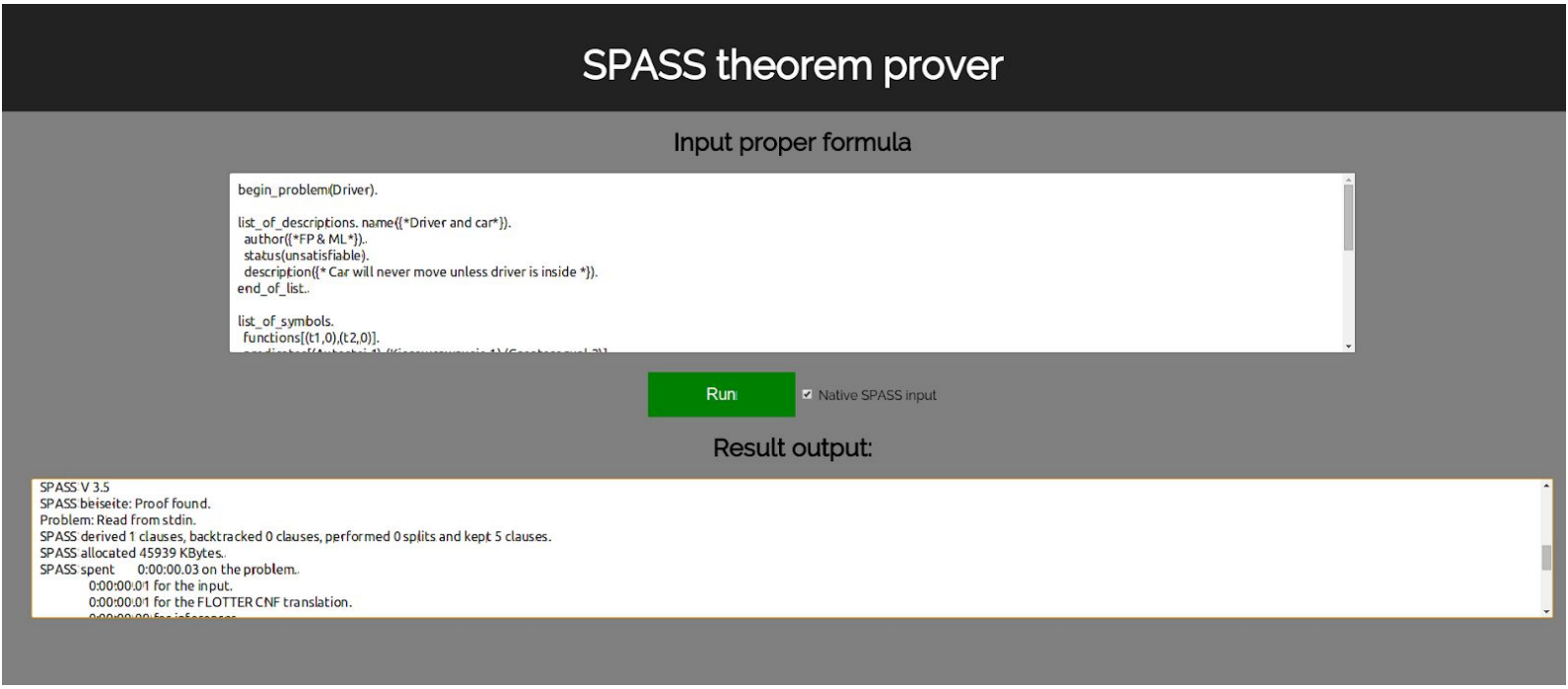
Następnie output provera jest przez odpowiednio sformatowany, a następnie wyświetlany w odpowiednim miejscu w interfejsie graficznym.

2.6. Natywny input

Nasz program umożliwia wprowadzenie formuły w natywnym formacie (oryginalnym formacie inputu dla provera SPASS) zamiast w formacie zaproponowanym przez nas. Do tego celu wystarczy zaznaczyć checkbox *Native SPASS input.*

2.7. Screeny


SPASS theorem prover
Input proper formula
name: Footballer and match
author: FP & ML
description: Match is always won when footballer plays
predicates[(Footballerplays,0),(Footballerplaysgood,0),(Teamwon,0)]..
axiom(always(implies(Footballerplays,Footballerplaysgood))).
axiom(implies(Footballerplaysgood,Teamwon)).
axiom(Footballerplays)..
conjecture(Teamwon).
Run
Native SPASS input
Result output:
----------SPASS-START----------
Input Problem:
1[0:Inp] || -> Footballerplays(t2)*.
2[0:Inp] || Teamwon(t2)* -> .
3[0:Inp] || -> Greaterthan(t2,t1)*.
4[0:Inp] || Footballerplaysgood(t2) -> Teamwon(t2)*.
5[0:Inp] || Greaterthan(U,V)* Footballerplays(U) -> Footballerplaysgood(U).


SPASS theorem prover
Input proper formula
begin_problem(Driver).
list_of_descriptions. name({*Driver and car*}).
author({*FP & ML*}).
status(unsatisfiable).
description({* Car will never move unless driver is inside *}).
end_of_list..
list_of_symbols.
functions[(t1,0),(t2,0)].
Run
Native SPASS input
Result output:
SPASS V 3.5
SPASS beiseite: Proof found.
Problem: Read from stdin.
SPASS derived 1 clauses, backtracked 0 clauses, performed 0 splits and kept 5 clauses.
SPASS allocated 45939 KBytes.
SPASS spent 0:00:00.03 on the problem.
0:00:00.01 for the input.
0:00:00.01 for the FLOTTER CNF translation.