Języki formalne i kompilatory II - dokumentacja

Wykonawcy:
- Jakub Cebula 33151
- Kacper Dąbrowa 33154
- Patryk Nytko 33189

Działanie programu:
Program wczytuje plik, tworzy drzewo parsingu z podanej gramatyki, używa wizytora do przechodzenia pomiędzy terminalami oraz wypisuje oczekiwany kod.

1. Instalacja oraz instrukcja

Prover9 - Prover9 to zautomatyzowane narzędzie do twierdzenia o logice pierwszego rzędu i logice równań opracowane przez Williama McCune'a.

Program dostępny jest na stronie
https://www.cs.unm.edu/~mccune/prover9/gui/v05.html

Bezpośredni link do pobrania
http://www.cs.unm.edu/%7Emccune/prover9/gui/Prover9-Mace4-v05-setup.exe

SPASS - jest zautomatyzowanym sprawdzianem twierdzeń dla logiki pierwszego rzędu z równością opracowanym w Instytucie Informatyki Maxa Plancka i wykorzystującym rachunek superpozycji. Nazwa pierwotnie oznaczała superpozycję Synergetic Prover Augmenting with Sorts.

Spass dostępny jest do pobrania na platformy Linux oraz Windows XP pod linkiem
https://www.mpi-inf.mpg.de/departments/automation-of-logic/software/spass-workbench/classic-spass-theorem-prover/download

Lub do skorzystania z wersji online na stronie
https://webspass.spass-prover.org/

2. Program z wykorzystaniem wtyczki ANTLR

Początkowym etapem tworzenia projektu było stworzenie gramatyki. Poniżej zamieszczamy fragment gramatyki Spass.

```
problem : 'begin_problem' '(' identifier ')' '.' description logical_part settings* 'end_problem' '.' EOF ;
description : 'list_of_descriptions' '.'
 'name' '(' ( Open text_ Close )? ')' '.'
 'author' '(' ( Open text_ Close )? ')' '.'
 ( 'version' '(' ( Open text_ Close )? ')' '.' )?
 ( 'logic' '(' ( Open text_ Close )? ')' '.' )?
 'status' '(' log_state ')' '.'
 'description' '(' ( Open text_ Close )? ')' '.'
 ( 'date' '(' ( Open text_ Close )? ')' '.' )?
 'end_of_list' '.'
 ;
log_state : 'satisfiable' | 'unsatisfiable' | 'unknown' ;
logical_part : symbol_list? declaration_list? formula_list* clause_list* proof_list* ;
symbol_list : 'list_of_symbols' '.'
 ( 'functions' '[' ( fun_sym | '(' fun_sym ',' arity ')' ) ( ',' ( fun_sym | '(' fun_sym ',' arity ')' ) )* ']' '.' )?
 ( 'predicates' '[' ( pred_sym | '(' pred_sym ',' arity ')' ) ( ',' ( pred_sym | '(' pred_sym ',' arity ')' ) )* ']' '.' )?
 ( 'sorts' '[' sort_sym ( ',' sort_sym )* ']' '.' )?
 'end_of_list' '.'
 ;
declaration_list : 'list_of_declarations' '.'
 declaration*
 'end_of_list' '.'
 ;
declaration : subsort_decl | term_decl | pred_decl | gen_decl ;
gen_decl : 'sort' sort_sym 'freely'? 'generated_by' func_list '.' ;
func_list : '[' fun_sym ( ',' fun_sym )* ']' ;
subsort_decl : 'subsort' '(' sort_sym ',' sort_sym ')' '.' ;
term_decl : 'forall' '(' term_list ',' term ')' '.' | term '.' ;
pred_decl : 'predicate' '(' pred_sym ( ',' sort_sym )+ ')' '.' ;
sort_sym : identifier ;
pred_sym : identifier ;
fun_sym : identifier ;
formula_list : 'list_of_formulae' '(' origin_type ')' '.'
 ( 'formula' '(' term? ( ',' label )? ')' '.' )*
 'end_of_list' '.'
```

Poniżej zamieszczamy gramatykę Prover-9.

```
formula: problem problem EOF;
problem: 'formulas' '(' string ')' '.' logical_part 'end_of_list' '.' ;

logical_part: (quantifier term '(' cnf_clause_body binary_operation cnf_clause_body ')' '.')? cnf_clause_body? '.';

term : quant_sym '(' term_list ',' term ')' | symbol | symbol '(' term ( ',' term )* ')' ;

term_list : '[' term ( ',' term )* ']' ;

quant_sym : 'forall' | 'exists' | identifier ;

symbol : 'equal' | 'true' | 'false' | 'or' | 'and' | 'not' | 'implies' | 'implied' | 'equiv' | identifier | string ;

cnf_clause_body : 'or' '(' term ( ',' term )* ')' | term '(' term ')';

binary_operation : '|' |
                   '&' |
                   '->' |
                   '<-' |
                   '<->';

quantifier :  'all' |
              'exists';

BINARY_TERM_OPERATION : '==' |
                        '<' |
                        '<=' |
                        '>' |
                        '>=' |
                        '+' |
                        '*' |
                        '@' |
                        '/' |
                        '\\' |
                        '^' |
                        'v' ;


identifier : Identifier ;
```

3. Kod wejściowy:

Prover9:

```
formulas(assumptions).
all x (man(x) -> mortal(x)).
man(socrates).
end_of_list.
formulas(goals).
mortal(socrates).
end_of_list.
```

SPASS:

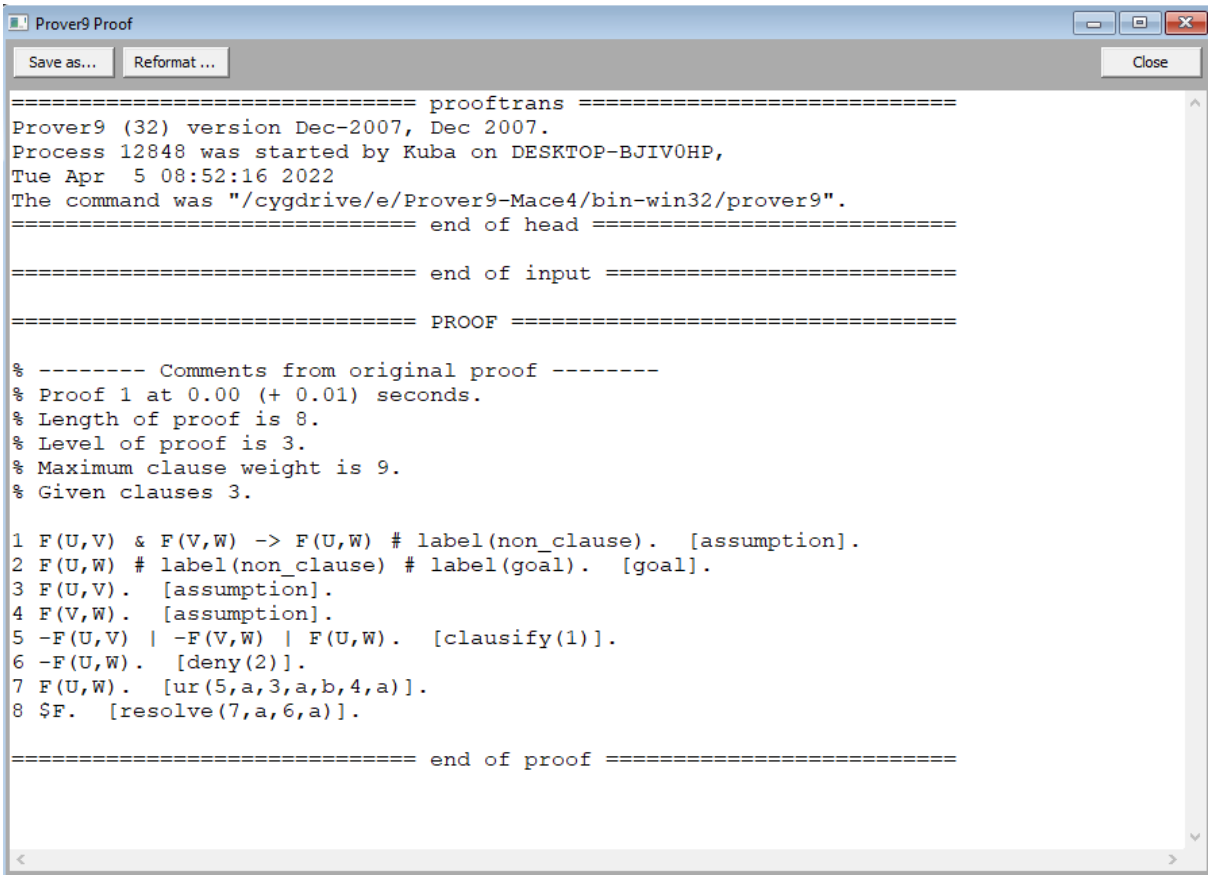
```
begin_problem(Pelletier57).
list_of_descriptions.
name({* Pelletier's Problem No. 57 *}).
author({* Christoph Weidenbach *}).
status(unsatisfiable).
description({* Problem taken in revised form from the
"Pelletier Collection",
Journal of Automated Reasoning, Vol. 2, No. 2, pages
191-216 *}).
end_of_list.
list_of_symbols.
functions[(f,2), (a,0), (b,0), (c,0)].
predicates[(F,2)].
end_of_list.
list_of_formulae(axioms).
formula(F(f(a,b),f(b,c))).
formula(F(f(b,c),f(a,c))).
formula(forall([U,V,W],implies(and(F(U,V),F(V,W)),F(U,
W)))).
end_of_list.
list_of_formulae(conjectures).
formula(F(f(a,b),f(a,c))).
end_of_list.
end_problem.
```

4. Kod wyjściowy:

Spass -> Prover9:

```
SPASS TO PROVER9

%name Pelletier's Problem No. 57
%author Christoph Weidenbach
%status unsatisfiable
%description Problem taken in revised form from the "Pelletier Collection",
Journal of Automated Reasoning, Vol. 2, No. 2, pages 191-216

formulas(assumptions).
F(U,V).
F(V,W).
&(F(U,V),F(V,W)) -> F(U,W).
end_of_list.
formulas(goals).
F(U,W).

end_of_list.
```

Wynik wywołania powyższego kodu w programie Prover9/Mace4:



Prover9 -> SPASS:

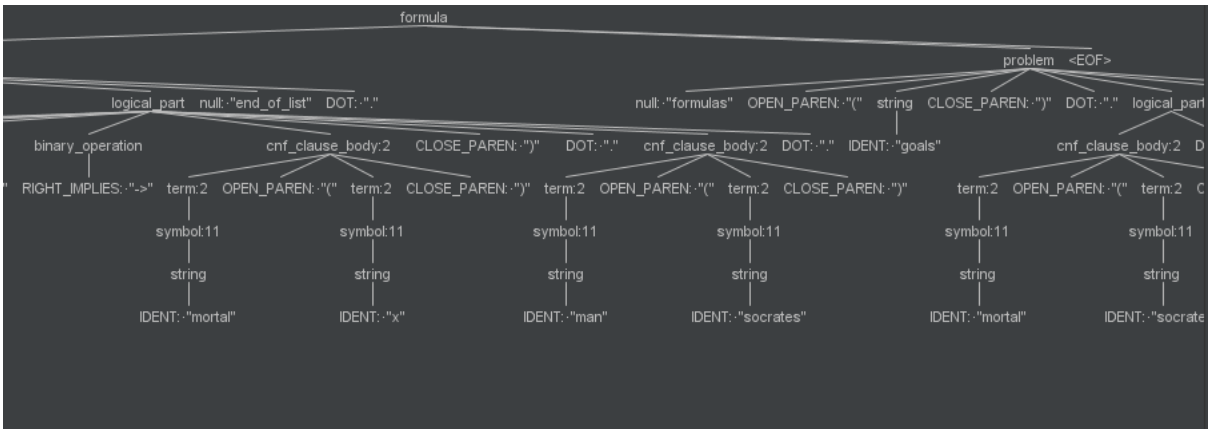
```
PROVER9 TO SPASS

begin_problem(Socrates1).
list_of_descriptions.
name({*Sokrates*}).
author({*Christoph Weidenbach*}).
status(unsatisfiable).
description({* Sokrates is mortal and since all humans are mortal, he is mortal too. *}).
end_of_list.

list_of_symbols.
 functions[(sokrates,0)].
 predicates[(Human,1),(Mortal,1)].
end_of_list.

list_of_formulae(axioms).
formula(Human(sokrates),1).
formula(forall([x],implies(Human(x),Mortal(x))),2).
end_of_list.
list_of_formulae(conjectures).
formula(Mortal(sokrates),3).
end_of_list.
end_problem.
```

Wynik wywołania powyższego kodu na stronie https://webspass.spass-prover.org/

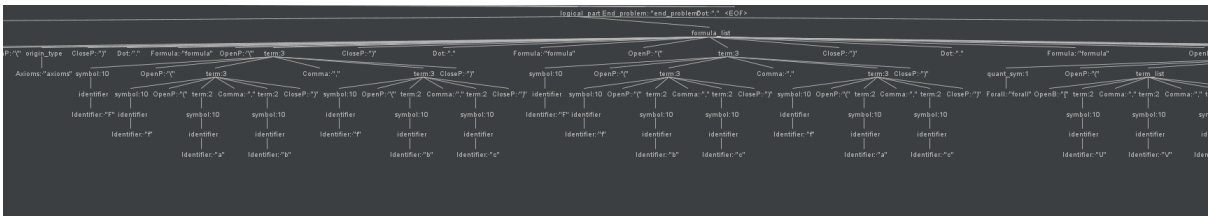
```
-------------------------SPASS-START-----------------------------
Input Problem:
1[0:Inp] ||  -> Human(sokrates)*.
2[0:Inp] || Mortal(sokrates)* -> .
3[0:Inp] || Human(u)* -> Mortal(u).
 This is a monadic Horn problem without equality.
 This is a problem that has, if any, a finite domain model.
 There are no function symbols.
 This is a problem that contains sort information.
 The conjecture is ground.
 The following monadic predicates have finite extensions: Human.
 Axiom clauses: 2 Conjecture clauses: 1
 Inferences: IEmS=1 ISoR=1 IORe=1
 Reductions: RFMRR=1 RBMRR=1 RObv=1 RUnC=1 RTaut=1 RSST=1 RSSi=1 RFSub=1 RBSub=1 RCon=1
 Extras    : Input Saturation, Always Selection, No Splitting, Full Reduction,  Ratio: 5, FuncWeight: 1, VarWeight: 1
 Precedence: Mortal > Human > sokrates
 Ordering  : KBO
Processed Problem:

Worked Off Clauses:

Usable Clauses:
1[0:Inp] ||  -> Human(sokrates)*.
2[0:Inp] || Mortal(sokrates)* -> .
3[0:Inp] Human(u) ||  -> Mortal(u)*.
SPASS V 3.9
SPASS beiseite: Proof found.
Problem: /tmp/webspass-webform_2022-04-05_11:09:03_7255l.txt
SPASS derived 1 clauses, backtracked 0 clauses, performed 0 splits and kept 4 clauses.
SPASS allocated 85013 KBytes.
SPASS spent     0:00:00.02 on the problem.
                0:00:00.01 for the input.
                0:00:00.00 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.

-------------------------SPASS-STOP-----------------------------
```

5. Fragment wygenerowanego drzewa z kodu wejściowego:

Prover9:



SPASS:


logical_part:End_problem:"end_problem"Dot:"." <EOF>
formula_list
Axioms:"axioms" symbol:10
quant_sym:1
Forall:"forall"