SPASS: Tutorial

SPASS Home Page
Tutorial
WebSPASS
Download
Useful Links
(Hi)Story
Contact

Learn How to Use SPASS!

SPASS is an automated theorem prover for first-order logic with equality. So the input for the prover is a first-order formula in our syntax. Running SPASS on such a formula results in the final output `SPASS beiseite: Proof found.` if the formula is valid, `SPASS beiseite: Completion found.` if the formula is not valid and because validity in first-order logic is undecidable, SPASS may run forever without producing any final result.

Lets start a complete loop through the usage of SPASS. Starting point is always some problem you want to solve. Assume that out of the two sentences

(1) Sokrates is a human.
(2) All humans are mortal.

you want to conclude

(3) Sokrates is mortal.

The first step is to formalize the problem in first-order logic. The formulae are

(1) *Human(sokrates)*
(2) $\forall$ *x Human(x)* $\supset$ *Mortal(x)*
(3) *Mortal(sokrates)*

where syntactically *Human* and *Mortal* are predicates while *sokrates* is a constant. The symbol $\forall$ introduces universal quantification and $\supset$ material implication. The next step is to provide SPASS an input file that contains exactly the above formulae in our syntax.

```
begin_problem(Sokrates1).

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

An SPASS input file consists of three parts, a description part started with `list_of_descriptions.`, a part where the signature is declared starting with `list_of_symbols.`, a part where all axioms are given, starting with `list_of_formulae(axioms).` and a final part where all conjectures are presented, starting with `list_of_formulae(conjectures).`. Formulae are introduced by `formula(` *`formula`* `).` and are always written in prefix notation. For further details, see the syntax description. Then SPASS tries to prove that the conjunction of all axiom formulae implies the disjunction of all conjectures.

You can copy the above input description for SPASS and paste it in our WebSPASS interface. The result output will be (similar to)

```
--------------------------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 ISoR IORe
 Reductions: RFClR RBClR RObv RUnC RTaut RSST RSSi RFSub RBSub RCon
 Extras    : Input Saturation, Always Selection, No Splitting, Full Reduction,
 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 1.0.0
SPASS beiseite: Proof found.
Problem: sokrates1.dfg
SPASS derived 1 clauses, backtracked 0 clauses and kept 4 clauses.
SPASS allocated 438 KBytes.
SPASS spent     0:00:00.12 on the problem.
                0:00:00.02 for the input.
                0:00:00.02 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------------------------------
```

SPASS reads the input file and transforms the formulae into clause normal form, where the conjecture(s) is negated, since SPASS is based on refutation. So the clauses

```
1[0:Inp] ||  -> Human(sokrates)*.
2[0:Inp] || Mortal(sokrates)* -> .
3[0:Inp] || Human(U) -> Mortal(U)*.
```

are our input clauses where variables where `->` denotes implication, a `*` means that a literal is maximal and negative literals left from `||` are monadic literals with a variable argument, treated by the sort technology in SPASS. Next SPASS analyzes the problem and finds out that

```
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
```

Based on this information SPASS decides to use the settings

```
Inferences: IEmS ISoR IORe
Reductions: RFClR RBClR RObv RUnC RTaut RSST RSSi RFSub RBSub RCon
Extras    : Input Saturation, Always Selection, No Splitting, Full Reduction,
Precedence: Mortal > Human > sokrates
Ordering  : KBO
```

where he decides to enable the sort inference rules empty sort (`IEmS`) and sort resolution (`ISoR`) and also ordered resolution (`IORe`). Reductions are forward and backward clause reduction (`RFClR RBClR`), obvious reductions (`RObv`), unit conflict (`RUnC`), syntactic tautology deletion (`RTaut`), static soft typing (`RSST`), sort simplification (`RSSi`), forward and backward subsumption (`RFSub RBSub`) and condensation (`RCon`). SPASS chooses the Knuth-Bendix ordering (`KBO`) with precedence `Mortal > Human > sokrates`.

For such a simple example, SPASS already finds the proof by input saturation/reduction, hence it does not output any *given* clauses.

```
Usable Clauses:
1[0:Inp] ||  -> Human(sokrates)*.
2[0:Inp] || Mortal(sokrates)* -> .
3[0:Inp]Human(U) ||  -> Mortal(U)*.
SPASS V 1.0.0
SPASS beiseite: Proof found.
Problem: sokrates1.dfg
SPASS derived 1 clauses, backtracked 0 clauses and kept 4 clauses.
SPASS allocated 438 KBytes.
SPASS spent     0:00:00.12 on the problem.
                0:00:00.02 for the input.
                0:00:00.02 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.
```

When called with default settings, SPASS does not output a proof. To get a proof, enter the option `-DocProof` in the options field of the WebSPASS interface and then the output will contain in addition the following proof

```
Here is a proof with depth 1, length 5 :
1[0:Inp] ||  -> Human(sokrates)*.
2[0:Inp] || Mortal(sokrates)* -> .
3[0:Inp]Human(U) ||  -> Mortal(U)*.
4[0:Res:3.1,2.0]Human(sokrates) ||  -> .
5[0:ClR:4.0,1.0] ||  -> .
Formulae used in the proof : 1 3 2
```

where clause `4` is the result of a resolution step between the clauses `3` and `2` and the empty clause, clause `5` is obtained via clause reduction from clauses `4` and `1`. The numbers given after `Formulae used in the proof` refer to the numbering of formulae in the SPASS input file. For further options consider our options help page.

Document last changed on Monday, 04 October 99 - 16:41