VL Logikorientierte Programmierung

Prolog Exercise

Uwe Egly, Michael Fink, Hans Tompits

Technische Universität Wien
Institut für Informationssysteme, Arbeitsbereich Wissensbasierte Systeme 184/3
Summer term 2009

1 Introduction

In this paper, you will find the Prolog exercise as well as additional information and hints for its solution.

The goal of the first exercise is the implementation of a simple theorem prover for ortho-logic (OL). From the introductory courses of your studies, you should know what a Gentzen (or sequent) system (for classical propositional logic) is. Just in case you have forgotten it, a brief description follows.

The basic objects of a sequent system are *sequents* of the form

$$\Gamma \;\vdash\; \Delta$$

where Γ, Δ are either sets, sequences or multi-sets of formulas. We will work with *sets of formulas* in the following. Additionally, we will restrict our attention to sequents with *at most two occurrences of formulas*. We call such sequents *2-sequents*. Ortho-logic can be characterized by a Gentzen system where every sequent is a 2-sequent. We call this Gentzen system GOL. Axioms for GOL are sequents of the form $f \vdash f$ for a formula f. The inference rule of GOL are as follows:

$$\frac{M \vdash a,\, N}{M \vdash a \vee b,\, N}\ R1 \qquad \frac{M \vdash b,\, N}{M \vdash a \vee b,\, N}\ R2 \qquad \frac{M,\, a \vdash N \quad M,\, b \vdash N}{M,\, a \vee b \vdash N}\ R3$$

$$\frac{M,\, a \vdash N}{M,\, a \wedge b \vdash N}\ R4 \qquad \frac{M,\, b \vdash N}{M,\, a \wedge b \vdash N}\ R5 \qquad \frac{M \vdash a,\, N \quad M \vdash b,\, N}{M \vdash a \wedge b,\, N}\ R6$$

$$\frac{M,\, a \vdash N}{M \vdash \neg a,\, N}\ R7 \qquad \frac{M \vdash a,\, N}{M,\, \neg a \vdash N}\ R8$$

$$\frac{M \vdash N}{M,\, a \vdash N}\ R9 \qquad \frac{M \vdash N}{M \vdash a,\, N}\ R10$$

Such a system is not well-suited for a proof search by computers, for which reason we will construct a simpler system in the following. This system is based on *negation normal form* (NNF). A formula is in NNF if (i) there are no other connectives than $\wedge, \vee, \neg$ in the formula, and (ii) the negation signs occur only in front of propositional formulas. Fortunately, every formula f has a NNF f' and it holds that f is provable in GOL iff f' is provable in GOL. Your first task is to write a Prolog program for constructing the NNF of a given formula (mainly by applying deMorgan's laws and the law of double negation elimination).

2 Translations of Formulas Into Negation Normal Form

Exercise 1: Implement a translation of an arbitrary formula with the connectives $\neg, \vee, \wedge$ into a negation normal form. Use the predicate

```
nnf(+InFormula,-OutFormula)
```

and denote formulas over propositional variables (in lower-case letters), the connective $\neg$ by `-`, the connective $\vee$ by `v`, and the connective $\wedge$ by &. Additionally, define the operators as follows:

```
op(425, fy,-)
op(450,yfx,&)
op(500,yfx,v)
```

The goal

```
:- nnf(-(a & (-(-(-b)) v c)),OutFormula).
```

should result in a single answer `OutFormula = -a v (b & -c)`. Obey that consecutive negations require parentheses. Choose (and document!) 10 cases to test the normal form translation. $\diamond$

3 Proof Search in GOL$^+$

The input formula has been translated into NNF. Next, we consider the system GOL$^+$, which we will use in the following. The axioms are of the form

$$\vdash a, \neg a$$

where a is (necessarily) a propositional variable (why?). We can simplify the inference rules resulting in the following rules for GOL$^+$.

$$\frac{\vdash e,\, N}{\vdash e \vee f,\, N}\ \vee \qquad \frac{\vdash f,\, N}{\vdash e \vee f,\, N}\ \vee \qquad \frac{\vdash e,\, N \qquad \vdash f,\, N}{\vdash e \wedge f,\, N}\ \wedge \qquad \frac{\vdash N}{\vdash e,\, N}\ W$$

Thereby, e, f are formulas and N is a set of formulas with cardinality ≤ 1.

Exercise 2: a) Implement a predicate

```
literalp(+InFormula)
```

which detects whether the given input formula `InFormula` in NNF is a literal (= a propositional variable or its negation). Examples of goals and their answers are:

yes	no
`literalp(a)`	`literalp(-a v b)`
`literalp(-a)`	`literalp(-a v b & c)`

b) Implement a predicate

```
no_or(+InFormula,-NoOR)
```

which computes the number of occurrences of the connective `v` in `InFormula` in NNF. Examples of goals and their answers are:

`no_or(a,N)`	$\mapsto$	`N = 0`	`no_or(-a v b,N)`	$\mapsto$	`N = 1`
`no_or(-a,N)`	$\mapsto$	`N = 0`	`no_or(a v b & (c v d),N)`	$\mapsto$	`N = 2`

$\diamond$

Before we continue with the actual automated proof search, we consider the formula $\neg p \vee (\neg q \vee (q \wedge (p \vee \neg q)))$ and one of its proof in GOL$^+$.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash q, \neg q}{\vdash q, \neg q \vee (q \wedge (p \vee \neg q))}\;\vee}{\vdash q, \neg p \vee (\neg q \vee (q \wedge (p \vee \neg q)))}\;\vee \qquad \dfrac{\dfrac{\vdash p, \neg p}{\vdash p, \neg p \vee (\neg q \vee (q \wedge (p \vee \neg q)))}\;\vee}{\vdash p \vee \neg q, \neg p \vee (\neg q \vee (q \wedge (p \vee \neg q)))}\;\vee}{\vdash q \wedge (p \vee \neg q), \neg p \vee (\neg q \vee (q \wedge (p \vee \neg q)))}\;\wedge}{\vdash \neg q \vee (q \wedge (p \vee \neg q)), \neg p \vee (\neg q \vee (q \wedge (p \vee \neg q)))}\;\vee}{\vdash \neg p \vee (\neg q \vee (q \wedge (p \vee \neg q)))}\;\vee
$$

In order to implement proof search, we use a three place predicate:

```
prove2(SeqFormula1,SeqFormula2,MaxNoOfWeakenings)
```

Thereby, `SeqFormula1` and `SeqFormula2` denote the formula occurrences in the 2-sequent and `MaxNoOfWeakenings` denotes the maximal number of the rule W in any branch of the proof tree. The use of this boundary is necessary in order to achieve termination of the search.

As you might have observed, the first two arguments of `prove2` represent the 2-sequent which has to be proved. With this representation, there is a minor difficulty: sequents with less than two occurrences of formulas cannot be represented in this way (if we fix the arity of the predicate). From theory, we know that sequents with no formula at all cannot occur in the proof. It remains to find a solution for the case when we have exactly one formula in the sequent.

The solution is simple. If we have a sequent with exactly one formula, we write this formula on *both* argument positions. Moreover, we check in the implementation of `prove2` whether the sequent contains one or two formulas. In the former case, `SeqFormula1` = `SeqFormula2`, and in the latter case, the two formulas differ.

In the following table, we classify the types of formulas which can occur in 2-sequents.

type 1	$\wedge$	literal	$\vee$	literal	$\wedge$	$\wedge$	$\vee$	$\vee$	W	$\vee$
type 2	literal	$\wedge$	literal	$\vee$	$\wedge$	$\vee$	$\wedge$	W	$\vee$	$\vee$

A table entry of the form $\wedge$ ($\vee$) means that the corresponding `SeqFormulai` is a conjunction (disjunction). A table entry literal means that the corresponding `SeqFormulai` is a literal. The table entry W indicates applicability of the rule W. Observe that W occurs only together with disjunction. This is *not* an error but the application of a result from [1] stating the completeness of proof search under this restriction.

In the following, we provide implementations for the axioms and the cases ($\wedge$, literal) and ($\vee$, W).

Axiom

```
prove2(A,-A,_NOW).
prove2(-A,A,_NOW).
```

($\wedge$, literal)

```
prove2((F1 & F2), A, NOW) :-
    literalp(A), prove2(F1,A,NOW), prove2(F2,A,NOW).
```

($\vee$, W)

```
% If both formulae are the same, the set is essentially unary
% and no weakening should be  possible.
prove2((F1 v F2), Z, NOW) :-
    NOW > 0, Z \== (F1 v F2), Z \== (F2 v F1),
    prove2((F1 v F2), (F1 v F2), NOW-1).
```

Exercise 3: Complete the proof search by implementing the remaining cases from the above table. Observe that there are *two* rules for disjunction in GOL^+. Test the predicate `prove2` by constructing at least 10 formulas in NNF and counting their occurrences of disjunctions. Document your test cases together with the result of the proof search. Explain that you have considered all relevant cases. ◇

4 The Whole Prover

We are now ready to construct the whole system from the components discussed so far. Define a rule of the following form.

```
prove(InFormula) :-
    nnf(InFormula,NNFFormula),
    no_or(NNFFormula,NoOr),
    prove2(NNFFormula,NNFFormula,NoOr).
```

Exercise 4: Test the system with the following formulas and check the resulting answers against the expected ones.

formula	expected answer
`-p v (q & (p v -q))`	`no`
`-p v p`	`yes`
`-p v (-q v (q & (q & (p v -q))))`	`yes`
`-p v (-q v (q & (p v -q)))`	`yes`
`-p v (-(-p) & (-p v -q)) v q`	`yes`

Optional Exercise 1: Proof Generation (5 points)

The goal `prove` does not return a proof in the success case but only the answer `yes`. Strictly speaking, `prove` implements a decision procedure. Extend the prover such that a proof is returned whenever the formula is provable. Print the proof with Prolog's output predicates. ◇

Optional Exercise 2: A Prover for Lattices (5 points)

Investigate how the discussed procedures can be used to implement a proof system for lattices (and implement it).

If you have questions don't hesitate to ask the teaching assistants.

Have fun!

References

[1] U. Egly and H. Tompits. On Different Proof Search Strategies for Orthologic. *Studia Logica*, 73:131–152, January 2003.