Logical reasoning and programming, task III

(December 11, 2018)

We have seen that it is easy to add new Boolean connectives, e.g. equivalence, to our systems, because all Boolean connectives are definable there. However, sometimes we want to add more complex connectives and operators. This leads to so called non-classical logics. For example, we can add a unary modal operator *necessarily*, usually denoted $\Box$ (called box), where $\Box p$ means "necessarily p". Logics containing such operators are called modal logics and were studied already by Aristotle. They occur quite frequently in computer science, e.g., in temporal and description logics.

Clearly, we should define the meaning of box, but we do not provide it. Instead of that, we shall present proof systems for the weakest normal modal propositional logic, called K, and define a provability in K using these systems. Although you are encouraged to find some more details about K, this is not necessary for completing the task.

The task consists of three parts

I implement in Prolog a tableau system for K (8 points),

II modify it (2 points),

III implement in Prolog a Hilbert style system for K (5 points).

You are supposed to submit all programs in an archive to BRUTE. Note that this task will be evaluated manually.

Formula representation

The following representation of formulae in Prolog is used

Logic	Prolog
p	`p`
$\neg p$	`n(p)`
$p \wedge q$	`a(p,q)`
$p \rightarrow q$	`i(p,q)`
$\Box p$	`box(p)`

and it naturally extends to all formulae. Propositional variables are atoms in Prolog, negation is expressed by a unary function `n`, conjunction is a binary function `a`, implication is a binary function `i`, and box is a unary function `box`.

For example, $\neg\neg((\Box p) \wedge ((\Box\neg(p \wedge \neg q)) \wedge (\neg\Box q)))$ is represented in Prolog by `n(n(a(box(p),a(box(n(a(p,n(q)))),n(box(q)))))).`

Note that in Part I and II an implication $\varphi \rightarrow \psi$ is just a shortcut for $\neg(\varphi \wedge \neg\psi)$. On the other hand, in Part III a conjunction $\varphi \wedge \psi$ is just a shortcut for $\neg(\varphi \rightarrow \neg\psi)$.

I Modaltab

Problem

Your task in this part is to implement a tableau system for K in Prolog. Note that the tableau system presented here differs slightly from the tableau system presented at the lecture.

Our tableau system for K deals with the sets of formulae. A formula φ is provable in K iff we can derive that $\{\neg\varphi\}$ is unsatisfiable using the tableau system for K. The tableau system contains rules that preserve satisfiability; if a set of formulae (above line) is satisfiable, then at least one of sets of formulae (below line) is satisfiable as well. It consists of the following rules:

$$\frac{\Gamma \cup \{\varphi, \neg\varphi\}}{\bot}\ (\bot) \qquad\qquad \frac{\Gamma \cup \{\neg\neg\varphi\}}{\Gamma \cup \{\varphi\}}\ (\neg)$$

$$\frac{\Gamma \cup \{\varphi \wedge \psi)\}}{\Gamma \cup \{\varphi, \psi\}}\ (\wedge) \qquad\qquad \frac{\Gamma \cup \{\neg(\varphi \wedge \psi)\}}{\Gamma \cup \{\neg\varphi\} \qquad \Gamma \cup \{\neg\psi\}}\ (\vee)$$

$$\frac{\Gamma \cup \Delta}{\Gamma}\ (\subset) \qquad\qquad \frac{\Box\Gamma \cup \{\neg\Box\varphi\}}{\Gamma \cup \{\neg\varphi\}}\ (\mathsf{K})$$

where Γ and Δ are sets of formulae, φ and ψ are formulae, and $\Box\Gamma = \{\,\Box\varphi \mid \varphi \in \Gamma\,\}$. The special symbol $\bot$ represents unsatisfiability and the rule $(\bot)$ says that a set of formulae containing φ and $\neg\varphi$ is trivially unsatisfiable, we also say that $(\bot)$ closes a branch. Only the rule $(\vee)$ contains branching — to prove that $\Gamma \cup \{\neg(\varphi \wedge \psi)\}$ is unsatisfiable, we have to prove that both $\Gamma \cup \{\neg\varphi\}$ and $\Gamma \cup \{\neg\psi\}$ are unsatisfiable.

A set of formulae is unsatisfiable if there is a derivation of this set using the given rules such that all branches are closed (end with $\bot$).

The following example is a proof of formula $\neg((\Box p) \wedge ((\Box\neg(p \wedge \neg q)) \wedge (\neg\Box q)))$, which is equivalent to $\Box p \to (\Box(p \to q) \to \Box q)$ in the language with implication. In other words, it says "if necessarily p and necessarily $p \to q$, then necessarily q". A basic principle which is valid in all normal modal logics.

$$\cfrac{\{\neg\neg((\Box p) \wedge ((\Box\neg(p \wedge \neg q)) \wedge (\neg\Box q)))\}}{\cfrac{\{(\Box p) \wedge ((\Box\neg(p \wedge \neg q)) \wedge (\neg\Box q))\}}{\cfrac{\{\Box p, (\Box\neg(p \wedge \neg q)) \wedge (\neg\Box q)\}}{\cfrac{\{\Box p, \Box\neg(p \wedge \neg q), \neg\Box q\}}{\cfrac{\{p, \neg(p \wedge \neg q), \neg q)\}}{\cfrac{\{p, \neg p, \neg q\}}{\bot}\,(\bot) \qquad \cfrac{\{p, \neg\neg q, \neg q\}}{\bot}\,(\bot)}\,(\vee)}\,(\mathsf{K})}\,(\wedge)}\,(\wedge)}\,(\neg)$$

The depth of such a proof, which is clearly a binary tree, is 6, because the longest path from $\bot$ to the root contains six applications of rules. Hence we say that $\neg((\Box p) \wedge ((\Box\neg(p \wedge \neg q)) \wedge (\neg\Box q)))$ has a proof of depth 6.

For further details about tableau systems for modal logics see, e.g., this text.

Program

You are supposed to upload a program `modaltab.pl`, in an archive, containing a binary predicate `prove`, where the first argument is an input formula and the second argument is a number. Hence we have `prove(-Formula, -Depth)` and it succeeds iff `Formula` has a proof of depth at most `Depth`. You can assume that implications do not occur in `Formula`.

For example,

```
?- prove(n(a(box(p), a(box(n(a(p, n(q)))), n(box(q))))), 6).
true.
```

and it also succeeds for the second argument being $7, 8, \ldots$.

Tips

It is wise to represent sets of formulae using lists. In this case you can use the predicate `select/3`, see help for further details.

There is no need to optimize your program for efficiency, your task is to write a simple program. A strightforward implementation of the presented tableau system is just fine.

Examples

```
?- prove(n(a(box(p), n(n(a(box(n(a(p, n(q)))), n(box(q))))))), 7).
true.

?- prove(n(a(box(p), n(p))), 10).
false.

?- prove(n(a(n(p), n(box(n(a(n(p), n(n(p)))))))), 7).
true.
```

II Modaltab 2

Note that the only reason to have the rule ($\subset$) in the previous part is to make it possible to apply the rule (K). Modify the program from Part I in such a way that you replace two rules ($\subset$) and (K) by a single rule; combine them together.

Note that formulae can have shorter proofs now, but we are not very concerned by that, because the second argument `Depth` in `prove` is really not necessary. The maximal depth of a proof is given by the size of `Formula`. Hence remove it—you can ignore it completely as long as you are sure that no cycle is possible; note that after an application of a rule you should obtain a smaller set.

Program

You are supposed to upload a program `modaltab2.pl`, in an archive, implementing the two above mentioned ideas. Hence we have `prove(-Formula)` and it succeeds iff `Formula` has a proof in a modified calculus. You can assume that implications do not occur in `Formula`.

III Modalhil

In this part your goal is to write a program that generates formulae provable in a (propositional) Hilbert style proof system for K in a naïve way. Namely you have the following schemata of axioms

$$
\begin{aligned}
&A \to (B \to A)\\
&(A \to (B \to C)) \to ((A \to B) \to (A \to C))\\
&(\neg B \to \neg A) \to (A \to B)\\
&\Box(A \to B) \to (\Box A \to \Box B)
\end{aligned}
$$

meaning that any instance of them has a proof of depth 1. For example, $(p \to q) \to ((q \to p) \to (p \to q))$, which we write as `i(i(p,q),i(i(q,p),i(p,q)))`, is an instance of the first schema. We also say that all these schemata have proofs of depth 1.

You can derive new schemata by using the following two rules:

(CD) If a schema A has a proof of depth k and a schema $B \to C$ has a proof of depth l, then a schema $C\sigma$, where $\sigma = mgu(A, B)$, has a proof of depth $\max(k, l) + 1$. Moreover, $C\sigma$ is a schema that is the result of applying the substitution σ on the schema C.

(Nec) If a schema A has a proof of depth k, then the schema $\Box A$ has a proof of depth $k + 1$.

For example, from $\Box(A \to B) \to (\Box A \to \Box B)$ and $(C \to (D \to E)) \to ((C \to D) \to (C \to E))$ we can derive $(\Box(A \to B) \to \Box A) \to (\Box(A \to B) \to \Box B)$ by (CD), because we use the substitution $\sigma = \{C \mapsto \Box(A \to B), D \mapsto \Box A, E \mapsto \Box B\}$ that is a mgu of $\Box(A \to B) \to (\Box A \to \Box B)$ and $C \to (D \to E)$. Moreover, the depth of the proof is 2.

We say that a formula φ has a proof of depth k, if a schema A has a proof of depth k and φ is an instance of A. For example, $\Box(A \to A)$ has a proof of depth 4 and hence $\Box((p \to q) \to (p \to q))$ has a proof of depth 4.

Program

You are supposed to upload a program `modalhil.pl`, in an archive, containing a binary predicate `derive`, where the first argument is an input formula and the second argument is a number. Hence we have `derive(-Formula, -Depth)` and it succeeds iff `Formula` has a proof of depth at most `Depth`. You can assume that conjunctions do not occur in `Formula`.

For example,

```
?- derive(i(box(p), i(box(i(p, q)), box(q))), 4).
true.
```

and it also succeeds for the second argument being $5, 6, \ldots$.

Tips

You can use the predicate `max/2`, see help. If you need unification with the occurs check, then use the predicate `unify_with_occurs_check/2`, see help.

There is no need to optimize your program for efficiency, your task is to write a simple program. A strightforward implementation of the presented Hilbert style system is just fine, although it is very inefficient compare to the approach presented during labs; a translation to first-order logic.

Examples

```
?- derive(i(box(q), i(box(p), box(p))), 4).
true.

?- derive(i(box(p), p), 4).
false.

?- derive(i(box(i(p, p)), i(q, q)), 4).
true.

?- derive(i(p, box(p)), 5).
false.
```

The last example can take a long time to compute (depending on your implementation). There is no need to wait for it (or rewrite your program), if you believe that your implementation is correct.