Logical reasoning and programming, task II

(November 13, 2017)

Problem

Your task is to write a program in Prolog that given a propositional formula φ produces an equisatisfiable formula φ' in CNF using the Tseytin transformation presented at the first lecture[1]. Hence the size of φ' is linear in the size of φ. A part of your task is to find out how to deal with equivalences.

Program

You are supposed to upload a program `cnf.pl`, in an archive, containing a binary predicate `cnf`, where the first argument is the input formula φ and the second argument is the output formula φ'. Hence using the standard notation that emphasizes the input/output behavior of predicates this can be expressed as `cnf(+InputFormula, -OutputFormula)`.

Usefull predicates

Note that you can use a built-in predicate `var` for testing whether a term is a variable or not. Hence `var(X)` succeeds, if `X` is a variable, and `var(X => Y)` fails. Also `Z = Y => Y, var(Z)` fails.

Input

As an input you get a term that describes a propositional formula. Propositional variables are expressed as variables in Prolog and the allowed connectives are negation (`~`), disjunction (`|`), conjunction (`&`), implication (`=>`), and equivalence (`<=>`). These connectives are expressed as operators in Prolog, hence add the following code[2] at the beginning of your program:

```
:-op(450,fy,~).  % Negation
:-(system_mode(true),op(502,xfy,'|'),system_mode(false)). % Disjunction
:-op(503,xfy,&). % Conjunction
:-op(504,xfx,=>). % Implication
:-op(505,xfx,<=>). % Equivalence
```

We express $(p \wedge q) \rightarrow r$ as `(P & Q) => R`, or even `P & Q => R`.

[1] Note that if you follow the approach presented at the lecture, then there is no need to worry about the following. In principle, you could produce an equisatisfiable formula φ' by checking whether φ is satisfiable or not and then produce a trivially satisfiable or unsatisfiable formula, respectively. However, you should realize that one of the goals of this transformation is to have a reasonable input for a solver. Moreover, the formula obtained by the Tseytin transformation has some additional properties that will be tested. For example, for any valuation v holds that if $v \models \varphi'$, then $v \models \varphi$. The opposite direction is a bit tricky. For any valuation v, if $v \models \varphi$, then there is a valuation v' such that $v' \models \varphi'$ and $v(p) = v'(p)$ for any propositional variable p occurring in φ. In other words, we have to change the valuation of new variables in φ' as needed.

[2] The operator used for disjunction has a meaning in Prolog and hence a special treatment is required.

Output

A formula in CNF is a conjunction of disjunctions (clauses) of literals. Our representation of a formula in CNF is a list (conjunction) of lists (disjunctions). A clause is represented by a list of literals and a conjunction of clauses is a list of their representations (lists). For example `P & (Q | ~R)` is expressed as `[[P], [Q, ~R]]`.

Note that the empty conjuction is expressed as `[]` and the empty clause as `[[]]`.

Example

Note that your output can be very different and do not take the following example as something you have to imitate. It heavily depends on the choice of rules for transformations, order, simplifications etc.

```
?- cnf(P => (Q => R), CNF).
CNF = [[_732], [~P, _750, ~_732], [P, _732], [~_750, _732],
       [~Q, R, ~_750], [Q, _750], [~R, _750]].
```