Automated Reasoning

Practical Assignment 1

Daniel de Leng
(3220540)

The delivered product is a simple theorem-prover. Some flaws exist; please read the README file for more information. The program, KEProver*, was written in Java version 1.6.*

Program structure

The program uses an abstract class Logic, which the classes Formula and Literal extend. Abstract variables are the negation boolean and the isLiteral boolean. Negation is maintained in the Logic-based objects themselves rather than via a logical connective inside a Formula. To distinguish between the two different Logic instantiations, the isLiteral boolean can be used. Finally, the abstract toString function is of uttermost importance in a command-line program as this one.

The Literal objects are the most simple Logic objects available, each additionally having a label field to keep them apart. Formula objects are generally more advanced, holding a connective and two Logic objects so that both sub-formulas and literals can be used – this is also the reason why the isLiteral boolean is often used.

Parser and Prover classes are abstract and hold static functions for their purposes. Nothing prevents those functions from being repositioned to a different class if would be so desired. KEProver holds the main method and is only used for requesting user input, before delegating the information to the Parser and Prover classes.

All in all the overall structure itself is quite simple, but allows for later upgrades to e.g. incorporate FOL.

Parser

The parser was personally written and uses a kind of finite-state machine to determine which kind of character is expected next as it moves over the input string character-by-character. Because negations are 1-ary connectives only used for truth purposes, I decided to incorporate the negation boolean instead. As such, only 2-ary connectives are expected. The parser has to and does automatically resolve occurrences of multiple successive negations because of this design choice.

The parser expects Formulas by default, because it can only be certain of the Logic type from seeing an infix connective. As such, it will 'fix' Formulas by transforming them back to Literals when required to do so.

User feedback has been added in case of input errors, also showing the character number in the error message for more easy troubleshooting.

Prover

The prover will always first check if the given left-hand side (LHS) and right-hand side (RHS) clash to conclude $F. It will then start work on the LHS, first tracking negations before using alpha and beta rules, after which it will move to the RHS to do the same. Every time a modification is made, the prover will search both LHS and RHS. If no clean-up can be applied, DC will be used. Backtracking is used and a tabu list is maintained during the search.