First-order logic tautology prover

Deadline: 24th June, 2022, 20:00

The objective of this assignment is to build a prover for tautologies of first-order logic based on Herbrand's theorem and the Davis-Putnam SAT solver. Modern provers are based on elaborations of this basic algorithm. On an input formula φ:

1. Convert $\neg\varphi$ to an equisatisfiable Skolem normal form $\psi \equiv \forall x_1, \ldots, x_n \cdot \xi$, with ξ quantifier-free.
2. Verify that ψ is unsatisfiable: By Herbrand's theorem, it suffices to find n-tuples of ground terms $\bar{u}_1, \ldots, \bar{u}_m$ (i.e., elements of the Herbrand universe) s.t.

   $$\xi[\bar{x} \mapsto \bar{u}_1] \wedge \cdots \wedge \xi[\bar{x} \mapsto \bar{u}_m]$$

   is unsatisfiable.
3. Use the Davis-Putnam SAT solver for propositional logic to check that the formula above is unsatisfiable.

Notice that (A) if φ is a tautology, then the algorithm above will terminate correctly verifying that φ is indeed a tautology, and if φ is not a tautology, then the algorithm above will not terminate if (B) ξ contains free variables and the Herbrand universe is infinite, and will otherwise terminate reporting that φ is not a tautology if (C.1) ξ does not contain free variables, or (C.2) the Herbrand universe is finite.

Examples. For an instance of case (A), consider the *drinker's paradox* $\varphi \equiv \exists x \cdot (D(x) \to \forall y \cdot D(y))$, which is a tautology of first-order logic. Its negation $\neg\varphi$ is logically equivalent to $\forall x \cdot (D(x) \wedge \exists y \cdot \neg D(y))$, which in prenex normal form is $\forall x \cdot \exists y \cdot (D(x) \wedge \neg D(y))$. By Skolemisation we obtain the equisatisfiable formula in Skolem normal form

$$\psi \equiv \forall x \cdot \underbrace{(D(x) \wedge \neg D(f(x)))}_{\xi},$$

where we have introduced a new Skolem functional symbol "f". Notice that at this point the Herbrand universe is empty (since there is no constant symbol), so we add to the signature an additional constant symbol c. (If there is already a constant symbol in the signature, then we do not need to add anything.) In this way the Herbrand universe contains all terms $c, f(c), f(f(c)), \ldots$. We now instantiate x in ξ with the two ground terms $u_1 \equiv c$ and $u_2 \equiv f(c)$, obtaining the ground formula

$$\underbrace{(D(c) \wedge \neg D(f(c)))}_{\xi[x \mapsto c]} \wedge \underbrace{(D(f(c)) \wedge \neg D(f(f(c))))}_{\xi[x \mapsto f(c)]},$$

which is clearly propositionally unsatisfiable. This confirms that the drinker's paradox is a tautology of first-order logic.

For an instance of case (B), consider the non-tautology $\varphi \equiv \exists x \cdot \forall y \cdot P(x, y)$. By negating and Skolemising we get $\psi \equiv \forall x \cdot \neg P(x, f(x))$, we add a fresh constant symbol c to make the Herbrand universe nonempty, however every finite expansion

$$\neg P(c, f(c)) \wedge \neg P(f(c), f^2(c)) \wedge \cdots \wedge \neg P(f^n(c), f^{n+1}(c))$$

is satisfiable (and thus ψ is satisfiable, and thus φ is not a tautology), however the program cannot determine this in a finite number of steps.

For an instance of case (C.1) consider the non-tautology $\varphi \equiv \forall x \cdot R(c, f(x))$, which after negation and Skolemisation becomes the satisfiable $\psi \equiv \neg R(c, f(d))$, which has no free variables and thus we can conclude that φ is a non-tautology. Note that the Herband universe is infinite in this case. Finally, for an instance of case (C.2) consider another non-tautology $\varphi \equiv \forall x, y \cdot \exists z \cdot R(x, y, z)$, after negation and Skolemisation we obtain $\psi \equiv \forall z \cdot \neg R(c, d, z)$ for two fresh constants c, d (comprising the entire Herbrand universe, which is thus finite in this case) and thus after verifying that the finite expansion $R(c, d, c) \wedge R(c, d, d)$ is satisfiable we can conclude that φ is a non-tautology.

Programming languages. The assignment can be solved using any modern programming language such as C, C++ (gcc), Java, Python, OCaml, Haskell... It is mandatory to provide a Makefile allowing the project to be automatically built and run in a modern computing environment. Typing "`make`" should result in an executable file called `FO-prover`, which will be run by the test suite to score the solution. In the file `FO-prover.hs` there is a simple skeleton written in Haskell correctly parsing the input. This can be used as a starting point to write the solution.

Input & output. The solution program must read the standard input `stdin`. The input consists of a single line encoding a formula of first-order logic φ according to the following Backus-Naur grammar (c.f. `Formula.hs`):

$$\begin{aligned}\varphi, \psi \ :\equiv\ & \mathsf{T} \mid \mathsf{F} \mid \mathsf{Rel}\ \text{“}\mathsf{string}\text{”}\ [t_1, \ldots, t_n] \mid \mathsf{Not}\ (\varphi) \mid \mathsf{And}\ (\varphi)\ (\psi) \mid \mathsf{Or}\ (\varphi)\ (\psi) \mid \\ & \mathsf{Implies}\ (\varphi)\ (\psi) \mid \mathsf{Iff}\ (\varphi)\ (\psi) \mid \\ & \mathsf{Exists}\ \text{“}\mathsf{string}\text{”}\ (\varphi) \mid \mathsf{Forall}\ \text{“}\mathsf{string}\text{”}\ (\varphi)\end{aligned}$$

where string is any sequence of alphanumeric characters, and the terms t_i's are in turn generated by the following grammar:

$$t \ :\equiv\ \mathsf{Var}\ \text{“}\mathsf{string}\text{”} \mid \mathsf{Fun}\ \text{“}\mathsf{string}\text{”}\ [t_1, \ldots, t_n].$$

For example, the input for drinker's paradox is:

```
Exists "x" (Implies (Rel "D" [Var "x"]) (Forall "y" (Rel "D" [Var "y"])))
```

See also the tests in `./tests` for further examples of input formulas.

`FO-prover` should output `1` (tautology) or `0` (non-tautology). Any other output will be considered invalid.

Testing, scoring, and grading. The script `./run_tests.sh` scores the provided solution against examples of type A, B, and C. The program is run for 10 seconds on each input instance (to have an idea: on a 2,3 GHz Quad-Core Intel Core i5 with 16 GB RAM) and assigns a score as follows:

type	output `0`	output `1`	timeout	# instances
A	-2	+1	0	76
B	+2 (!)	-2	+1	5
C	0	-2	-1	50

The total score is the sum of the scores for every input. The total number of points for this project is 30. If the score is $x \in \{-232, \ldots, 86\}$, then the number of awarded points is

$$\left\lceil \frac{\min(\max(x, 0), 81) \cdot 30}{81} \right\rceil .$$

Submission. The submission is done on moodle.