Artificial Intelligence in Theorem Proving

Homework 2

Please send your solutions to `bartosz@mimuw.edu.pl` by **02.06.2019**.

First-order resolution A (first-order) *resolvent* of two clauses A and B is

$$((A_0 \setminus A_1) \cup (B_0 \setminus B_1))\sigma,$$

where A_0 and B_0 are renamed versions of A and B with no variables in common, and A_1 and B_1 are arbitrary nonempty subsets of A_0 and B_0, respectively, with σ being an mgu of $A_1 \cup -B_1$. ($-C$ is a shorthand for $\{-p\colon p \in C\}$.)

Unification algorithm A set of pairs of terms $\{(x_1, t_1), \ldots, (x_n, t_n)\}$ is said to be in *solved form* if $x_1, \ldots, x_n$ are distinct variables that do not appear in terms $t_1, \ldots, t_n$.

Given a finite set of pairs of terms $S = \{s_1, t_1), \ldots, (s_n, t_n)\}$, the given algorithm either finds a set of pairs of terms in solved form that defines an mgu σ such that $s_i\sigma = t_i\sigma$, for $1 \leq i \leq n$, or it fails. If it fails, there is no unifier for the set.

- $S \cup \{(u, u)\} \rightsquigarrow S$,
- $S \cup \{(f(u_1, \ldots, u_k), f(v_1, \ldots, v_k))\} \rightsquigarrow S \cup \{(u_1, v_1), \ldots, (u_k, v_k)\}$,
- $S \cup \{(f(u_1, \ldots, u_k), g(v_1, \ldots, v_l))\} \rightsquigarrow$ fail if $f \neq g$ or $k \neq l$,
- $S \cup \{(f(u_1, \ldots, u_k), x)\} \rightsquigarrow S \cup \{(x, f(u_1, \ldots, u_k))\}$,
- $S \cup \{(x, u)\} \rightsquigarrow S[x \mapsto u] \cup \{(x, u)\}$ if $x \notin u$,
- $S \cup \{(x, u)\} \rightsquigarrow$ fail if $x \in u$,

where u, u_i, v_i are terms, x is variable, and $S[x \mapsto u]$ is S with each occurence of x substituted with a term u.

Exercise 1. (*2 points*) What resolvents can be obtained from the following two clauses?

1. {¬P(x), ¬P(E(x,y)), P(y)}
2. {P(E(E(E(E(x, E(y,z)), E(y, x)), E(z, u)), u))}

Exercise 2. (***12 points***) The task here is to train a machine learning model for premise selection and, with use of it and `E` prover, automatically prove new conjectures – the more the better.

The data for this exercise originate from Mizar Mathematical Library (MML) translated to TPTP formalism.

First, install `E` prover and set the value of the `EPROVER` environmental variable to `/path/to/E/PROVER/eprover`. Then download and unpack this archive. In a folder `data` there are files:

- `dependencies_train`, which contains a list of theorems and its dependencies (premises) extracted from proofs found with use of `E` prover. Mind that one theorem can have several different proofs and thus several different sets of dependencies (the file contains 15907 sets of dependencies for 1135 theorems).
- `features`, which contains (syntactic, sparse) features of all premises / theorems / conjectures, extracted by some reasonable method.

Using data from this two files you should train a machine learning model which is supposed to rank / select relevant premises out of all accessible for a given conjecture with unknown dependencies. Premises accessible for a conjecture are all premises appearing before the conjecture in the chronological order inherited from MML – this order is saved in the file `data/chronology`.

Having trained the machine learning model use the advice from it to prove with `E` prover as many conjectures from `data/conjectures_test` as possible. For this you will need statements of conjectures and premises – which are saved in `data/statements`. In `README.md` you will find a demonstration how to do it. You should run `E` prover as in `run_E_prover.py` script – with `--auto`, `--cpu-limit=1`, `--free-numbers` and `--proof-object` flags. `--auto` and `--cpu-limit=1` will limit the power of `E` prover, which is intentional, to amplify the impact of premise selection on the results.

There are roughly two schemas for applying machine learning for premise selection you can use:

- **multilabel setting**, where premises are treated as labels on theorems embedded in the feature space; having a new conjecture the ML model is supposed to propose labels for it according to the similarity of the conjecture to the theorems in the training set; the best labels are put to the prover as axioms for the conjecture; kNN or Naive Bayes models may be suitable in this setting;
- **binary setting**, where a machine learning model is trained on pairs

  $$[\text{features}(\texttt{theorem}), \text{features}(\texttt{premise})]$$

  labeled by 0 or 1, according to usefulness of the `premise` for proving the `theorem`. Having trained a ML model and given a new `conjecture`, we evaluate pairs

  $$[\text{features}(\texttt{conjecture}), \text{features}(\texttt{premise})]$$

  for all the `premise`s accessible for the `conjecture` and put to the prover these with the highest scores. XGBoost, Random Forest or linear models may be suitable in this setting.

You can use an arbitrary machine learning method and setting, but you should compare at least two different ML models in this task. You can use available implementations of ML models.

Hint: A sugested setting for using advice from a trained machine learning model is the following: for each `conjecture` construct a ranking of premises according to the trained model. Then, try to prove the `conjecture` with $2, 4, 8, \ldots, 2^n$ top premises from the ranking.

The solution should consist of an implementation, proofs of conjectures from `E` prover and a concise report.