Simple Prolog Interpreter using C++

Aditya Sheth(IMT2018003)
Dev Patel(IMT2018021)
Prateek Kamboj(IMT2018057)

May 19, 2021

Introduction to Prolog Functionalities

1. **Unification Problem:** Given 2 expressions, unification function checks if two expression can be unified by assignment of some internal variables to a constant or complex term.
2. **Proof-search problem:** Given a knowledge base of facts $\{f_1, f_2, ..f_n\}$(each f_i may have conjunction of dependency on other facts); there will be a query fact q given. Return all the valid assignment of internal variables in q. A variable assignment of f is valid if the corresponding f unifies with atleast one fact in knowledge base.

Our project majorly provides implementation to solve these 2 problems.

Data Structures used

```
struct Variable
{
    string var_name;
    Variable();
    Variable(string tmp);
    void print();
    bool operator<(const Variable &other) const;
    bool operator==(const Variable &other) const;
};
struct Constant
{
    bool is_int;
    string const_str;
    int const_int;
    Constant(string tmp);
    Constant(int tmp);
    void print();
};
```

```
struct Complex
{
    string functor;
    vector<Term *> subterms;
    Complex(string functor_name, vector<Term *> subterms_list);
    void print();
};
struct Term
{
    Variable *variable;
    Constant *constant;
    Complex *complex;
    Term();
    Term(Variable *tmp);
    Term(Constant *tmp);
    Term(Complex *tmp);
    void print();
};
```

Unification

1. Keep track of left term and right term in recursion.
2. **Both left and right term variable:** Merge the variables in Disjoint-set union data structure maintained.
3. **One term is variable and other is non-variable:** Check if previously assigned value unifies with new value or not. Handled trivial case seperately.
4. **Both terms are non-variable:**
    1. Both terms are constant: check equality.
    2. Both terms are complex: recursively unify all the internal terms
    3. One term is complex and other is constant: Trivially returns false.
5. At end of unification, check if all variables in same disjoint-set have unifiable assignments.

Proof-search

1. For Finding variable assignment to fact Q, we iterate through all pre-given facts F in knowledge-base.
2. If query Q and head of pre-given fact F unifies, we find all the possible variable assignments for each term in tail of F using proof-search recursively. Now we try all variable assignment for query fact Q. Now, a variable assignment for Q is valid with respect to Fact F only if this variable assignment satisfies every Term in tail of F. Therefore, we recursively try to solve this problem for each F with Q.

Demo

Link to github repository

Further Improvements

1. Code parsing is done using trivial code and it doesn't check for errors in syntax. Lexer libraries can be integrated to upgrade syntax checking.
2. The proof-search algorithm assumes a Directed acyclic structure for dependencies. To allow cyclic dependencies between facts, changes should be implemented.
3. In dependency of fact, only conjuction operation is used. It would be interesting problem to solve for any arbitrary use of (or,and) operations instead of just and operations.