Project Proposal Prolog

Concepts of Programming Languages

Winand, Roald, Sjoerd, Alexey and Anvar

January 16, 2017

1 Background

The subject for our project will be Prolog, a general-purpose logic programming language. It is considered as one of the first and most popular logic programming language available. Originally Prolog was intended as a way to process natural language, but it has also been used for theorem proving and expert systems.

The language is suited for situations in which rule-based logical queries are used, for example databases or voice processing systems. It originated from first-order logic and is, unlike most programming languages, declarative. This means that the flow is expressed in terms of relation represented as fact and rules. By running a query over these relations, a computation can be done.

2 Problem

How can we implement Prolog in C#?

3 Methodology

The goal of our project will be embedding Prolog in C#. As this is quite a complex and iterative task, we are going to do this by implementing each particular aspect of Prolog step by step.

3.1 Term representation

Prolog distinguishes four types of terms: atoms, numbers, variables and compounds. As these terms all require a specific representation in C#, a solution will be constructed to do this in an efficient way. Lists are considered a special compound term, hence are composed of the terms in the list. For example the list [1, 2, 3] is actually represented as the compound term .(1, .(2, .(3, []))). This means that a special representation is required for lists, as they are not constant in length. Strings are actually lists of characters and thus also represented this way. At the completion of this part the following terms, for example, can be represented in an efficient way:

```
true
false
foo
foo(bar)
[1, 2, 3]
[a, b, c]
"abc"
foo(X)
foo(_)
```

3.2 Clauses

When term representation is complete, the processing of clauses, which are facts and rules stored in some file, will be implemented. Processing means that defined clauses are stored in memory

using the correct representation. It will therefore be possible to process a file containing facts and rules. As rules can consist of multiple clauses, processing of composed rules will also be possible. It should be noted that the implication form of the horn clause will be used, as this is the default in Prolog. When this part is completed the following example file can be processed:

```
foo.
foo(bar).
bar :- foo.
bar :- true.
bar :- foo, foo(bar).
```

3.3 Unification

Variables should be bound to a specific value when it is appropriate, hence unified. Up to this point variables are only considered a type of term, thus not treated as actual variables. On occasions where variables are assigned, they will be bound to the values in the local context. This means that after this part the following rules can be processed:

```
bar(X) :- X==1.
foo(X) :- bar(X).
```

3.4 Backtracking

As the minimal requirements for a simple Prolog program are present, the execution of such a program will be made possible by implementing backtracking. Because backtracking has quite some complexity in time and space, the completion of this part will be considered a milestone. A number of unit tests will be performed on the execution of different Prolog programs to guarantee that backtracking behaves as expected. After this part it should be possible to process a program like:

```
B.
D.
A :- B.
A :- C.
A :- D.
```

3.5 Mathematics

While the program is being tested for its soundness, mathematical operations will be implemented. As this is considered a non-critical functionality, it will only be done when everything is on track. After implementing this part, the following rules can be processed:

```
bar(X) :- X is 1.
foo(X) :- bar(Y), X is Y + 1.
```

3.6 Native functions

Due to computational efficiency, functions like *findall* are natively implemented in Prolog compilers. It should be possible to implement *findall* and alike at this point, as backtracking will be sound. This means that an attempt will be done to implement different native functions in this part.

3.7 Negation and cut

To complete the implementation of Prolog the usage of negation and cut will be made possible. As the implementation of especially cut might be quite complex, it will be considered as the final step of embedding Prolog in C#. After this part the embedding is considered complete and a number of unit tests will be ran over the resulting embedding. It will also be compared to different Prolog compilers, using specific programs (from literature).

4 Planning

4.1 Workload ditribution

We have made the following workload distribution:

- Sjoerd and Winand will work directly on the embedding itself.
- Anvar researches and writes about other embeddings of Prolog in other programming languages.
- Alexey will make unit tests.
- Roald works on both the implementation and research.

4.2 Weekly planning

- **Week 49**:
  Implement Term representation and Clauses. Begin writing unit tests for each aspect of the implementation that is being worked on from now. Start doing research on prolog embeddings in other languages and on the different aspects of the implementation.

- **Week 50**:
  Start working on Unification and Backtracking.

- **Week 51**:
  Finish implementing Unification and Backtracking and implement Mathematics.

- **Week 52 & 1**:
  Christmas break, make sure all point for weeks 49 to 51 have been completed and possibly try to get ahead of schedule by completing work scheduled for week 2.

- **Week 2**:
  Implement Native functions, start with implementing Negation and Cut. Start putting research together in the format of the final report.

- **Week 3**:
  Finish implementing Negation and Cut and thus the embedding part of the project. Write the final unit tests. Run unit tests and include results in the report

- **Week 4**:
  Finish the final details of the report and make a presentation. Practise presentation.