Alma Mater Studiorum - University of Bologna

Computer Science and Engineering - DISI

Artificial Intelligence

Labeled Prolog: a computational model in 2p-KT

Master degree thesis

Supervisor

Prof. Roberta Calegari

Co-supervisor

Prof. Giovanni Ciatto

Candidate

Giuseppe Boezio

Academic Year 2021-2022- Third session

Abstract

Content of the abstract

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Thesis organization

First chapter introduces the general content about thesis and gives a short presentation of the topic, the problem and the solution we propose;

Second chapter a deepening about the theoretical foundations used during the stage and the project;

Third chapter presents the datasets used during for the training and the testing of the model;

Fourth chapter presents the experiments did during to develop the system;

Fifth chapter presents the different implementations of the system;

Sixth chapter discusses about the results and possible future developments.

During the drafting of the essay, following typography conventions are considered:

- the acronyms, abbreviations, ambiguous terms or terms not in common use are defined in the glossary, in the end of the present document;
- the first occurrences of the terms in the glossary are highlighted like this: word;
- the terms from the foreign language or jargon are highlighted like this: *italics*.

Chapter 2

Background notions

2.1 The Prolog Language

2.1.1 Brief History

Prolog stands for *PROgrammation en LOGique* and it emerged during 1970s as a way to use logic as a programming language. The early developers of this language were Robert Kowalski, Maarten van Emden and Alain Colmelauer. The programming language, Prolog, was born of a project aimed not at producing a programming language but at processing natural languages; in this case, French [**10.1145/155360.155362**]. The project gave rise to a preliminary version of Prolog at the end of 1971 and a more definitive version at the end of 1972. Prolog gained a lot of attraction from the computing society as it was the very first logic programming language. The language still holds considerable importance and popularity among the logic programming languages and comes with a range of commercial as well as free implementations.
Prolog is used for different kind of tasks such as:

- theorem proving [**coelho1986automated**]
- expert systems [**merritt2012building**]
- knowledge representation [**gelfond2002logic**]
- automated planning [**pinna2015resolving**]
- natural language processing [**lally2011natural**]

2.1.2 Concepts

Syntax and semantics of Prolog are described in ISO standard ISO/IEC 13211. Prolog is a logic programming language; this means that it is used to describe known facts and relationships about a problem and less about prescribing the sequence of steps taken by a computer to solve the problem. When a computer is programmed in Prolog, the actual way the computer carries out the computation is specified partially by the logic declarative semantics of Prolog, partly by what new facts can be inferred from the given ones, and only partly by explicit control information supplied by the programmer.

Prolog is used to solve problems which involve objects and relations among them. The main features of the programming language are:

- specifyng some *facts* about some objects and their relationships
- defining some *rules* about object and their relationships
- asking *questions* about objects and their relationships

Prolog programs are built from terms. A term is either a constant, a variable or a structure.

2.1.2.1 Constants

A constant is a sequence of characters which denotes a specific object or relationship. A constant can be an atom or a number. All constants begin with a lower case letter.

Example

a is an atom
12 is a number

2.1.2.2 Variables

A variable looks like an atom except it has a name beginning with capital letter or underline signed. A variable should be thought of as standing for some objects we are unable or unwilling to name at the time we write the program.

Example

X and *Answer* are valid names for variables

2.1.2.3 Structures

A structure is a collection of other objects called *components*. Structures help to organize the data in a program because they permit a group of related information to be treated as a single object instead of separate entities. A structure is written in Prolog by specifying its *functor* and its *components*. The coponents are enclosed in round brackets and separated by commas. The functor is written just before the opening round brackets.

Example

owns(john,book) is a structure having owns as functor and, john and book as components

2.1.2.4 Facts

A fact is a relation among objects which are all *ground*. This means that a fact is a *structure* which does not contain any variables among its components. A fact is written as a structure followed by a dot (.). The names of the objects that are enclosed within the round brackets in each fact are called *arguments*. The name of the relationship which comes just before the round brackets is called *predicate*.

Example

king(john,france). is a fact

2.1.2.5 Rules

A rule is a disjunction of predicates, where at most one is not negated, written in the following way:

$$Head :- Body.$$

where Head is a predicate, Body is a conjunction of predicates and the symbol ":-" means that the body implies the head. This kind of structure is called Horn clause. A Prolog program can be seen as a list (because order matters) of Horn clauses

called *theory*. Facts could be seen as Rules having Body equals to true. Rules are used to describe some complex relations among objects of the domain of discourse and differently from facts can contain variables.

Example

motherOf(X,Y) :- parentOf(X,Y),female(X).

The aforementioned description of Prolog language has been adapted from **[Clocksin1987Programmin**

2.1.2.6 Unification

A substitution is a function which associates a variable to a given term. The most general unifier (m.g.u.) is the substitution which allows to transform two predicates making them equals such that all other substitutions can be obtained through a composition with this one. The unification is a process whereby two structures are made equals via substitution and it is used several times during the Prolog resolution process.

2.1.2.7 Resolution

The resolution in Prolog happens in the following way:the interpreter tries to verify whether a conjunction of predicates (the goal) provided by the user can be derived from the current program or not and in the case it could, it provides a computed answer substitution (c.a.s.) which is a set of substitutions which allow to make true the user's goal.
Prolog resolution process is called SLD (Selective Linear Definite clause resolution) and works as follows:
SLD resolution implicitly defines a search tree of alternative computations, in which the initial goal clause is associated with the root of the tree. For every node in the tree and for every definite clause in the program whose positive literal unifies with the selected literal in the goal clause associated with the node, there is a child node associated with the goal clause obtained by SLD resolution. A leaf node, which has no children, is a success node if its associated goal clause is the empty clause. It is a failure node if its associated goal clause is non-empty but its selected literal unifies with no positive literal of definite clauses in the program. SLD resolution is non-deterministic in the sense that it does not determine the search strategy for exploring the search tree. Prolog searches the tree depth-first, one branch at a time, using

backtracking when it encounters a failure node. Depth-first search is very efficient in its use of computing resources, but is incomplete if the search space contains infinite branches and the search strategy searches these in preference to finite branches: the computation does not terminate. The SLD resolution search space is an or-tree, in which different branches represent alternative computations.[**Gallier1985LogicFC**]

2.1.3 2p-Kt

2p-Kt is a general, extensible, and interoperable ecosystem for logic programming and symbolic AI written in Kotlin which supports the Prolog ISO standard.
2p-kt is the evolution of another project called tuProlog [**CIATTO2021100817**].

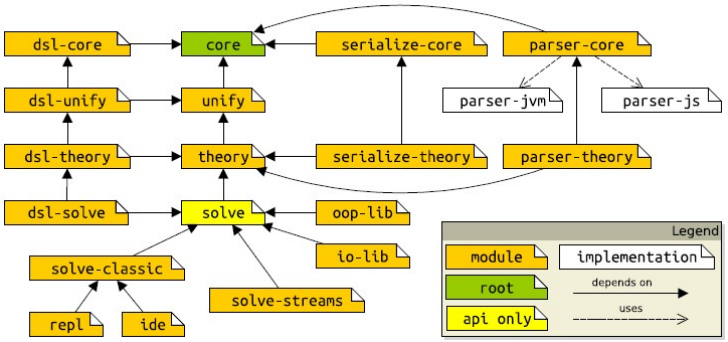

Figure 2.1: 2P-Kt project map. LP functionalities are partitioned into some loosely-coupled and incrementally-dependent modules.

To support reusability 2p-Kt is divided into several modules described as follows:

- **:core**: exposes data structures for knowledge representation via terms and clauses, other than methods supporting their manipulation
- **:unify**: used to compare and manipulate logic terms through logic unification
- **:theory**: in-memory storage of clauses into ordered (e.g. queues) or unordered (e.g. multisets) data structures, and their efficient retrieval via pattern-matching
- **:serialize-* and :parser-***: used to perform ancillary operations such as serialization and parsing
- **:solve**: aspects which are orthogonal w.r.t. any particular resolution strategy —e.g. errors management, extensibility via libraries, I/O, etc
- **:solve-***: modules which implement a specific resolution strategy

- **:repl and :ide**: provide CLI and GUI

The structure of the project can be seen in figure 2.1.
2p-Kt provides a well-grounded technological basis for implementing/experimenting/extending the many solutions proposed in the literature—e.g., abductive inference, rule induction, probabilistic reasoning and labelled LP.

2.2 Constraint Programming

2.2.1 Brief History

2.2.2 Constraint Logic Programming

2.2.3 SWI Prolog - CLP libraries

Chapter 3

CLP in 2p-Kt

3.1 Requirements

3.2 Design

3.3 Implementation

3.4 Case study

3.4.1 Design

3.4.2 Implementation

Chapter 4

Labeled Prolog

4.1 Model

4.1.1 Labeled Variables

4.1.2 Labeled Terms

4.2 Implementation

Chapter 5

CLP as Labeled Prolog

Chapter 6

Conclusions and future work

Acknowledgements

First, I would like to express my deepest gratitude to Professor Calegari and Professor Ciatto for all the support they provided me during the internship and thesis redaction processes.

Second, I would like to thank my family, my friends, my former classmates and all people who believed in me during this long study path.

Last but not least, I would like to thank myself to have been able to never give up during these two years.

Bologna, 03 February 2023 Giuseppe Boezio

Bibliopraphy