

ST0244 Programming Languages

7. Logic Programming

Andrés Sicard-Ramírez

Universidad EAFIT

Semester 2023-2

Preliminaries

Conventions

- The number and page numbers assigned to chapters, examples, exercises, figures, quotes, sections and theorems on these slides correspond to the numbers assigned in the textbook [Lee 2017].
- The source code examples are in course's repository.

Introduction

From prescriptive language to descriptive languages (p. 278)

- *'Assembly languages are very prescriptive language, meaning that you must think in terms of the particular machine and solve problems accordingly. Programmers must think in terms of the von Neumann machine stored program computer model.'*

Introduction

From prescriptive language to descriptive languages (p. 278)

- *'Assembly languages are very prescriptive language, meaning that you must think in terms of the particular machine and solve problems accordingly. Programmers must think in terms of the von Neumann machine stored program computer model.'*
- *'C++, Java and Python are high-level languages and hence allows you to think in a more descriptive way about a problem. However, the underlying computational model is still the von Neumann machine.'*

Introduction

From prescriptive language to descriptive languages (p. 278)

- *'Assembly languages are very prescriptive language, meaning that you must think in terms of the particular machine and solve problems accordingly. Programmers must think in terms of the von Neumann machine stored program computer model.'*
- *'C++, Java and Python are high-level languages and hence allows you to think in a more descriptive way about a problem. However, the underlying computational model is still the von Neumann machine.'*
- *'[Haskell and] Standard ML are high-level languages too, but allows the programmer to think in a mathematical way about a problem. These languages get away from the traditional von Neumann model in some ways.'*

Introduction

From prescriptive language to descriptive languages (p. 278)

- *'Assembly languages are very prescriptive language, meaning that you must think in terms of the particular machine and solve problems accordingly. Programmers must think in terms of the von Neumann machine stored program computer model.'*
- *'C++, Java and Python are high-level languages and hence allows you to think in a more descriptive way about a problem. However, the underlying computational model is still the von Neumann machine.'*
- *'[Haskell and] Standard ML are high-level languages too, but allows the programmer to think in a mathematical way about a problem. These languages get away from the traditional von Neumann model in some ways.'*
- *'Prolog takes the descriptive component of languages further and lets programmers write programs based solely on describing relationships.'*

Introduction

Features of logic programming's languages (pp. 277-278)

- ***Descriptive languages:*** Programs are expressed as *known facts* and *logical relationships* about a problem. Programmers assert the existence of the desired result and a *logic interpreter* then uses the computer to find the desired result by making *inferences* to prove its existence.'

Introduction

Features of logic programming's languages (pp. 277-278)

- ***Descriptive languages:*** Programs are expressed as *known facts* and *logical relationships* about a problem. Programmers assert the existence of the desired result and a *logic interpreter* then uses the computer to find the desired result by making *inferences* to prove its existence.'
- ***Non-procedural languages:*** The programmer states only what is to be accomplished and leaves it to the interpreter to determine how it is to be accomplished.'

Introduction

Features of logic programming's languages (pp. 277-278)

- **'Descriptive languages:** Programs are expressed as *known facts* and *logical relationships* about a problem. Programmers assert the existence of the desired result and a *logic interpreter* then uses the computer to find the desired result by making *inferences* to prove its existence.'
- **'Non-procedural languages:** The programmer states only what is to be accomplished and leaves it to the interpreter to determine how it is to be accomplished.'
- **'Relational languages:** Desired results are expressed as relations or predicates instead of as functions. Rather than define a function for calculating a square root, the programmer defines a relation, say $\text{sqrt}(x, y)$, that is true exactly when $y^2 = x$.'

Getting Started with Prolog

Introduction

- **Prolog** is the programming language usually associated with logic programming.
- **Prolog** was developed in 1972 by Alain Colmerauer and Phillipe Roussel.
- **Prolog** is based on **first-order logic** and **unification** (variables unify to terms). It is not based on the von Neumann architecture.
- Unification is made using depth-first search and backtracking.
- There are various versions of **Prolog** available including SWI-Prolog and GNU Prolog.
- Some references: [Clocksin and Mellish 2003], [Ulf and Maluszyński 2000] and [Apt 1996].

Getting Started with Prolog

Example

See file `lp/family.pl`.

Fundamentals

Terminology

- Variables
- Atoms (textual constants)
- Numbers (numeric constants)
- Terms (variables or constants)
- Predicates (properties or relations)
- Facts (predicates instanced)

Fundamentals

Definition

A **Prolog program** is a set of facts and predicates [Clocksin and Mellish 2003].

Fundamentals

Example

In `lp/family.pl` we have for example:

- Atoms

`bruce and esther.`

- Predicates

`female(X) and parent(X, Y).`

- Facts

`female(michelle), male(john) and parent(gary, kent).`

Fundamentals

Example

We define the binary predicate (relation) *father*.

X is a *father* of *Y* if $(:-)$ *X* is a *parent* of *Y* and $(,)$ *X* is *male*:

```
father(X,Y) :- parent(X,Y), male(X).
```

The Prolog Program

Prolog performs **unification** to search for a solution.

In general, **unification** is the process of solving a set of equations between symbolic expressions: Getting a list of **substitutions** of terms by variables.

For getting a valid substitution, Prolog uses **depth-first search** and **backtracking**.

Example

Using the `lp/family.pl` we have:

```
? - father(gary, X) .  
X = kent;  
X = stephen;  
X = anne.
```


Lists

Prolog supports lists:

- The empty list is written `[]`.
- The list with head `H` and tail `T` is written `[H|T]`.
- Sugar syntax: `[1, 2, 3]` denotes the list `[1|[2|[3|[]]]]`.
- Sugar syntax: `[1]` denotes the list `[1|[]]`.
- Since Prolog uses depth-first search and backtracking we can also use `[H|T]` for pattern matching.

Lists

Example

See file `lp/lists.pl`.

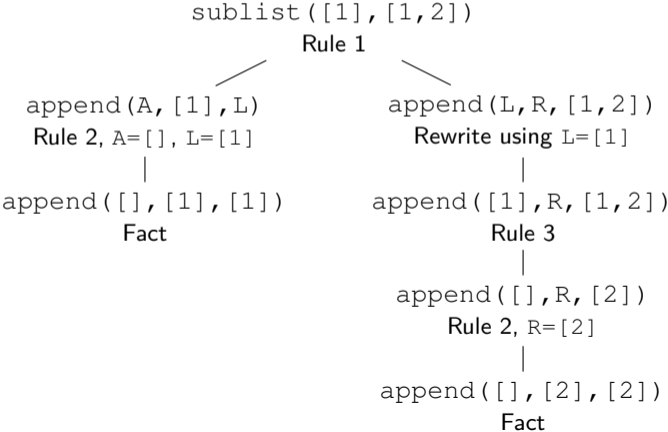
Lists

Example

Rule 1: `sublist(X,Y) :- append(_,X,L), append(L,_,Y).`

Rule 2: `append([],Y,Y).`

Rule 3: `append([H|T1],L2,[H|T3]) :- append(T1,L2,T3).`



The Accumulator Pattern





Example

See file `lp/reverse.pl`.

Built-In Predicates

- $X = Y$ succeeds if X and Y unify.
- $X \neq Y$ succeeds if X and Y do not unify.
- Relational operators on numbers on infix form ($<$, $>$, $=<$, $>=$, $:=$ and \neq).
- The `not/1` predicate checks that the argument (predicate) does not hold.
- The `atom/1` predicate checks that the argument is an atom.
- The `number/1` predicate checks that the argument is a number.

References

-  Apt, Krzystof R. (1996). From Logic Programming to Prolog. Series in Computer Sciences. Prentice-Hall (cit. on p. [10](#)).
-  Clocksin, William F. and Mellish, Christopher S. [1981] (2003). Programming in Prolog. 5th ed. Springer. DOI: [10.1007/978-3-642-55481-0](https://doi.org/10.1007/978-3-642-55481-0) (cit. on pp. [10](#), [13](#)).
-  Lee, Kent D. [2014] (2017). Foundations of Programming Languages. 2nd ed. Undergraduate Topics in Computer Science. Springer (cit. on p. [2](#)).
-  Ulf, Nilsson and Maluszyński, Jan [1990] (2000). Logic, Programming and Prolog. 2nd ed. John Wiley & Sons (cit. on p. [10](#)).