Introduction to AI with Prolog

Arithmetic

Alexey Sery
a.seryj@g.nsu.ru

Novosibirsk State University
*THE REAL SCIENCE

НГУ ФИТ

Department of Information Technologies
Novosibirsk State University

October 11, 2020

Goals of the lesson

- To introduce Prolog's inbuilt abilities for performing arithmetic.
- To discuss how they can be applied to other data types' processing problems.

Drawbacks of the successor arithmetic

Recall our Peano arithmetic implementation.

```
num(0).
num(s(N)) :- num(N).
```

This is a pure predicate that terminates if any of the arguments is instantiated. We can read the clauses declaratively to reason about the cases that this relation describes. Other elementary relations between natural numbers can be defined analogously.

Drawbacks of the successor arithmetic

Unfortunately, this representation of natural numbers suffers from several significant disadvantages:

- First, this is not really the way we want to write and read numbers. We would like to use a more familiar notation — such as 1, 2 and 3 — to represent natural numbers.
- With some practice, we may get used to the successor notation. However, a more fundamental problem remains: This representation takes space that is directly proportional to the magnitude of the numbers we need to represent. Thus, the space requirement grows exponentially with the length of any number's decimal representation. Therefore, reasoning about larger numbers is infeasible with this representation.
- More complex relations such as multiplication and exponentiation are hard to define in such a way that they work in all directions and retain good termination properties.
- To extend this representation to integers, we also need a way to represent negative numbers.

Drawbacks of the successor arithmetic

Therefore, successor notation — albeit useful to illustrate different ways in which we could represent our data — is not how we typically reason about numbers in Prolog.

Drawbacks of the successor arithmetic

Instead, we use built-in predicates to reason about numbers in Prolog. In the case of integers, these predicates are known as **CLP(FD)** constraints, and in more recent systems also as CLP($\mathbb{Z}$) constraints. CLP(FD) stands for **Constraint Logic Programming over Finite Domains** and reminds us of the fact that in reality, we can only represent a finite subset of integers on actual machines, while $\mathbb{Z}$ denotes the integers and indicates that these constraints are designed for reasoning about all integers.

All widely used Prolog implementations provide CLP(FD) constraints. However, the exact details differ slightly between various systems. For example, in GNU Prolog, B-Prolog and other systems, CLP(FD) constraints are conveniently available right from the start. In contrast, you need to load a library to use them in SICStus Prolog and other systems.

SWI Prolog provides a number of basic arithmetic tools for manipulating integers.

Arithmetic in Prolog

SWI-Prolog defines the following numeric types:

- **Integer** If SWI-Prolog is built using the GNU multiple precision arithmetic library (GMP), integer arithmetic is unbounded, which means that the size of integers is limited by available memory only. Without GMP, SWI-Prolog integers are 64-bits, regardless of the native integer size of the platform. The inbuild predicate `integer/1` holds if an argument is an integer.
- **Rational number** Rational numbers ($\mathbb{Q}$) are quotients of two integers ($\frac{N}{M}$). Rational arithmetic is only provided if GMP is used. Rational numbers satisfy the type tests `rational/1`, `number/1` and `atomic/1` and may satisfy the type test `integer/1`, i.e., integers are considered rational numbers. Rational numbers are always kept in canonical representation, which means M is positive and N and M have no common divisors.
- **Float** Floating point numbers are represented using the C type double. On most of today's platforms these are 64-bit IEEE floating point numbers. Satisfy the type tests `float/1`, `number/1` and `atomic/1`.

Arithmetic in Prolog

Arithmetic expression	Same in Prolog
$10 + 5 = 15$	`15 is 10 + 5.`
$10 \cdot 5 = 50$	`50 is 10 * 5.`
$10 - 5 = 5$	`5 is 10 - 5.`
$5 - 10 = -5$	`-5 is 5 - 10.`
$10 \ / \ 5 = 2$	`2 is 10 / 5.`
$10 \ / \ 4 = 2.5$	`2.5 is 10 / 4.`
$10 \ / \ 4 = 4 \cdot 2 + 2$	`2 is mod(10,4).`

Arithmetic in Prolog

Arithmetic expression	Same in Prolog
$x < y$	`X < Y.`
$x \leqslant y$	`X =< Y.`
$x = y$	`X =:= Y.`
$x \neq y$	`X =\= Y.`
$x \geqslant y$	`X >= Y.`
$x > y$	`X > Y.`

Arithmetic in Prolog

SWI-Prolog provides many extensions to the set of floating point functions defined by the ISO standard.

Example

`sin, cos, tan, log, log10, exp, **, sqrt, ceil, floor, round, abs, max, min, », «`

Arithmetic in Prolog

Normally all Prolog does is just unification of variables to structures. Arithmetic is something extra that has been bolted on to the basic Prolog engine because it is useful.

Arithmetic functions are terms which are evaluated by the arithmetic predicates. Apart from the fact that the functors go between their arguments instead of in front of them these are ordinary Prolog terms, and unless we do something special, Prolog will not actually do any arithmetic.

To force Prolog to actually evaluate arithmetic expressions we have to use inbuild binary predicate `is/2`.

Since arithmetic functions is an extra stuff in Prolog then it is not surprising that there are some restrictions on this extra ability.

- The arithmetic expressions to be evaluated must be on the right hand side of `is`.
- Moreover, every variable on the right hand side must already have been instantiated to a number. If the variable is uninstantiated, or if it is instantiated to something other than a number, we will get some sort of `instantiation_error` message.

Arithmetic in Prolog

Here's an example.

```
double(N, D) :- D is N * 2.
```

This predicate simply doubles its first argument and returns the answer in its second argument.

- `double(5, A)` returns `A = 10`.
- But posing the query `double(Half, 10)` does not return anything. Instead we get the `instantiation_error` message. When we pose the query, we are asking Prolog to evaluate expression `10 is Half * 2`, which is impossible due to uninstantiation of `Half`.

Example

`15 is 10 + 5.` $\Leftrightarrow$ `is(15,+(10,5)).`

`X is (2*5 + 10) / 4.` $\Leftrightarrow$ `is(X,/(+(*(2,5),10),4)).`