CIS 343

Extra Credit Assignment

Muna Gigowski

Constraint/Logic Based Programming

Constraint programming is a programming paradigm where relations between variables are expressed via constraints. Constraints are quite different than primitives, as they do not specify what to execute, but rather the properties of a solution to be found. This structure categorizes constraint programming as a form of non-procedural programming (formally “declarative programming”). As with other non-procedural programming types, constraint programming is often injected into other languages to be used as a debugging/program validation tool. Some common applications of CL programming are theorem proving, expert systems, term rewriting, type systems, and automated planning, and natural language processing. The most common data types in CL programming languages are floating point numbers, trees, or terms, all of which can be used in evaluations (there are no assignments in CL programming) to achieve some goal. One of the most popular constraint logic based programming languages is Prolog, which is still widely used today. In the following paragraphs I will detail specifically how Prolog handles some of its logic-based constraint evaluations, and why it has risen to the top of the CL programming list.

Prolog was one of the first logic programing languages, and is based around the concept of relations, which are then queried for validity. Relations can be broken down into clauses, which are composed of terms. A term can be an atom (a general purpose name), number, variable, or compound term (several terms used together). Clauses themselves can be of two types: a fact or a rule. A fact is a clause with an empty body, such as:

**star(Vega)**

Now we could query this fact with an expression similar to:

**?- star(Vega)**

(the answer to this query would be “Yes”)

Rules on the other hand are clauses with bodies, such as:

**flower(X) :- tulip(X)**

Given this rule, we could run a query that asks “what things are flowers?” with the syntax:

**?- flower(X)**

(In response we would get X = tulip)

Finally, there are goals, which are essentially a clause followed by arguments (the arguments being what you are striving to meet/achieve). Logically, for a goal to be achieved the predicate must have appeared in at least one fact or rule in the program. Goals and queries are very similar, and you can even have a goal as a part of a query or a rule. To put it simply, goals are statements you are trying to achieve, and you can ask the program if they are achieved by running queries on rules or facts. This all may seem rather simplistic, but because of the wide range of possible relations and queries Prolog can be used in countless contexts of logical processing.

Moving outward, a Prolog program executes when it is given a query. The Prolog engine tries to resolve the query (refuting the negated query) by means of a method called SLD Resolution. If the negation of the given query can be refuted, proving that the given query was successful, the program continues on. I found it extremely interesting that to prove a query the program tries to DISPROVE the query’s negation, instead of directly disprove the given query itself. I was not able to find an answer as to why the language logically executes this way, but the roundabout method it uses to prove “truthiness” is very interesting (and a bit over my head). To end with a few cool facts:

* In Prolog, the loading of code is referred to as “consulting”
* Prolog is able to perform recursion, loops, and sorting
* Prolog is a language that can be both compiled and interpreted

After researching the basics of CL programming and the various CL languages, it became immediately apparent that logic-based programming is not my preferred type of development. Although I find the methods used within this programming paradigm to be exceptionally interesting, to me the implementation is so sterile and boring I think I would fall asleep before ever finishing my program. In any case, there is no question that the applications of CL programming are extensive, which makes them great languages to familiarize myself with.