1988

Artificial Intelligence through Prolog by Neil C. Rowe

Rowe, Neil C.

Prentice-Hall

http://hdl.handle.net/10945/36984

Appendix D: Summary of the Prolog dialect used in this book

The Prolog examples in this book use a subset of the "core Prolog" or "standard Prolog" originally developed in several implementations at Edinburgh University and covered in W.F. Clocksin and C.S. Mellish, *Programming in Prolog*, second edition, Springer-Verlag, 1984. Substantially similar dialects are provided for many computers, and include Quintus Prolog, Logicware MPROLOG, DEC-10 Prolog, and C-Prolog. (Programs in this book were tested in Edinburgh C-Prolog on a DEC VAX-780.) Borland International Turbo Prolog is close to this standard, but doesn't support treating rules as data (Section D.8) and has some other differences. Micro-Prolog has major differences; see Appendix E. For compatibility with these dialects and easy comprehension by the reader, the Prolog subset of this book omits many common and useful features, including other arithmetic operations, input and output, structures (as opposed to embedded lists), functors, modules, and many debugging tools; see the Prolog books in Appendix F for information about these.

This Appendix is summarized in Figure D-1.

D.1 Managing facts and rules

We assume Prolog programs (consisting of rules and facts) are written outside of the Prolog interpreter. You use your favorite editor to create a text file containing them. Then you start up the Prolog interpreter (how to do this is implementation-dependent) and load in the files you want by querying **consult**, whose one argument is the name of the editor-created file you want to load. If you have several files to load, do **consult** several times.

To list all the facts and rules in the Prolog database, query **listing** with no arguments. To list only the facts and rules with a particular predicate name, query with that name.

To add a fact or rule to the Prolog database from within the interpreter, query **asserta** or **assertz**. Built-in predicate **asserta** adds the new fact at the front of the database, and **assertz** adds it to the end. To remove a fact from the Prolog database, query **retract** with that fact an argument. To remove all the facts and rules with a particular predicate name, query **abolish**, whose first argument is the predicate name and whose second is the number of arguments the first-argument predicate has. All four of these predicates fail on backtracking, so the only way to undo an **asserta** or **assertz** is a **retract**.

D.2 The format of facts, rules and queries

Facts, rules, and queries are all built of the same thing: predicate expressions. An expression consists of a predicate name (starting with a lower-case letter) optionally followed by one or more arguments separated by commas and enclosed in parentheses. Arguments that are constants must be numbers or start with a lower-case letter, and arguments that are variables must start with a capital letter. Facts are just predicate expressions followed by a period ("**.**").

Queries are things you type after the Prolog interpreter types "**?-** " on your terminal; query mode is the interpreter's normal mode. A query is a single predicate expression, or several expressions linked with commas or semicolons, concluded by a period and carriage return. Commas mean "and" and semicolons

mean "or". You can also specify that the opposite of something be true with the **not** predicate of one argument; its argument is the predicate expression you want to fail | REFERENCE 1|. .FS | REFERENCE 1| The **not** is denoted with prefix "/+" in Quintus Prolog, and it won't work with inside unbound variables in Turbo Prolog. .FE

If you type a semicolon after the interpreter types an answer to a query, the interpreter will go back to work and try to find another answer to the query. You can keep typing semicolons as long as there are more answers. If the interpreter can't find an answer, it will type **no**.

Rules consist of a predicate expression followed by a space, the symbol "**:-**", another space, a query, and a period. The stuff before the **:-** is the *left side* and the stuff after is the *right side*. Rules can be read as "if the right side is true then the left side is true".

D.3. Program layout

Spaces and carriage returns can be inserted at many places in Prolog programs without changing the meaning. However, it's conventional not to put them in normal predicate expressions (except within strings). A space is generally put after every "and"-comma and "or"-semicolon in a query or rule, as well as before and after the rule symbol **:-** and the arithmetic-assignment **is**.

Another convention of Prolog programming is to leave blank lines in programs between rules and facts beginning with different predicate names; this makes it easier to see which lines go together.

Comments are preceded with a "**/***" and followed by a "***/**".

D.4. Lists

Arguments to predicate expressions can be lists. Lists are indicated by the brackets "**[**" and "**]**"; commas (not to be confused with the commas between predicate arguments) separate the list items. The list of no items "[]" is called the empty list. Lists of any length can be represented by variables, or they can be referred to by the notation **[<front_items>|<sublist>]**. Here **<front_items>** represents an item, or several items separated by commas, at the front (left end) of a list, and **<sublist>** is a variable representing the arbitrarily-long rest of the list. Items within a list can be variables too.

D.5. Numbers

Arguments to predicate expressions can be numbers. (Some dialects accept only integers, and some others only accept decimals with digits before and after the decimal point.) Prolog handles numbers with infix notation, to make things easier to read. (Predicate expressions are prefix notation; that is, the predicate name comes first.) The infix comparison predicates are **=** (equals), **>** (greater than), **<** (less than), **=<** (less than or equal to), and **>=** (greater than or equal to). Any variables used in these comparisons must be bound to values.

Arithmetic assignment is handled by the **is** infix symbol | REFERENCE 2|. .FS | REFERENCE 2| Not available in Turbo Prolog: **=** is dual-purpose. .FE To the left of the **is** and separated by a space must be the name of a variable; to the right and separated by a space must be an arithmetic expression of constants and/or bound variables. The arithmetic expression is evaluated, and its value bound to the left-side variable. The

arithmetic expression must be in infix form, using the standard infix arithmetic symbols **+**,**-**, ***, and **/**; parentheses can group terms.

D.6. Output and input

To write something on the terminal screen use the **write** predicate of one argument. The argument can be a word, list, number, string, bound variable (for which it prints the current binding), or even unbound variable (for which it prints the current internal code of the variable name). To get a carriage return on the terminal, query the **nl** (that stands for "new line") predicate of no arguments. The predicate **read** of one argument, a variable, reads in what the user types on the terminal keyboard and binds the variable to it. What the user types must end with a period, just like everything else in Prolog.

D.7. Strings

Character strings are denoted by single-quotation marks or apostrophes ("'") around some characters and punctuation marks. Strings can contain capital letters, commas, periods, and other punctuation marks in any order, so they're less restricted than predicate names, variable names, and constants.

D.8. Treating rules and facts as data | REFERENCE 3|

.FS | REFERENCE 3| Not supported in Turbo Prolog. .FE The predicate **clause** matches its first argument to the left side and its second argument to the right side of a rule or fact in the database (if a fact, its second argument is set to the empty query which always succeeds, "true") | REFERENCE 4|. .FS | REFERENCE 4| Some Prolog dialects won't accept a **clause** expression unless the left-side predicate name is filled in. But you can get the same effect by iterating over the list of all predicate names, trying each one for the left side. .FE The infix "univ" operation represented by the symbol "**=..**" converts queries into lists and vice versa; the left side of this symbol is the query, and the right side the list. If the left side is bound it is converted into its equivalent list, and if the right side is bound it is converted into its equivalent query. Related to this, the **call** predicate of one argument takes a query and executes it as if it were typed on the terminal.

D.9. Miscellaneous predicates

The **fail** predicate of no arguments provides a predicate expression that always fails, forcing backtracking. The "true" predicate always succeeds. The cut predicate (symbolized by "!") of no arguments forces throwing away of certain backtracking information; see Section 10.7 for a detailed discussion. The **var** predicate of one argument succeeds if its argument is unbound. The **number** predicate of one argument succeeds if its argument is a number. The **halt** predicate of no arguments stops the Prolog interpreter and returns you to the operating system; it's the recommended way to finish a session.

D.10. Definable predicates

Besides the preceding built-in predicates, some predicates defined in this book in terms of others are built-in in many Prolog dialects. The **forall** predicate of two arguments (Section 7.12) checks whether universal quantification holds for some predicate expression; specifically, it checks whether every set of variable

bindings that satisfies its first-argument predicate also satisfies its second-argument predicate expression. The **doall** predicate of one argument (Section 7.12) repeatedly calls its argument predicate expression until that argument fails. The **repeat** predicate (Section 10.8) of no arguments always succeeds, and always succeeds afresh on backtracking. The **bagof** predicate | REFERENCE 5| .FS | REFERENCE 5| Called **findall** in Clocksin and Mellish. .FE of three arguments (Section 10.6) collects into a list all values to which a variable can be bound to satisfy a predicate expression. The first argument is the variable, the second is the expression (somewhere inside which must be the variable), and the third argument is the list of all possible values, the result of the computation.

D.11. Debugging

Since the activities of Prolog interpreters can be complicated, good debugging facilities are essential. These facilities differ considerably between dialects. But **trace**, **spy**, and **leash** features are common. These are built-in Prolog predicates that are queried to cause certain side effects. Predicate **trace** is a predicate of no arguments that makes the Prolog interpreter print a running account of everything it is doing; **notrace** undoes it. Predicate **spy** is a predicate of one argument, a predicate name, that makes the Prolog interpreter print everything happening with that predicate; **nospy** with that same argument undoes it. To spy on several predicates, do several **spy** commands. The **trace** and **spy** printouts report calls, successes, failures, and backtracks involving predicate expressions with a given predicate name. The **leash(on)** predicate expression tells the interpreter to stop and pause after every printout line generated by a **trace** or **spy**; **leash(off)** undoes it.

Go to book index