**Prolog Cheat Sheet**

**Basic stuff**

* Constants start with lowercase letters. Numbers are also constants.
* Use ‘\_’ (anonymous variable) if you do not care what the value of the outcome variable will be. Also stops Prolog from screaming [singleton] warning at you
* [] is special, don’t forget.
* [H|T] is legal form of accessing lists:
  + So is [H1, H2, H3|T]
* When defining predicates using recursion, always put base case first.
* Meaningful comments can give you marks even if your predicate doesn’t work.
* Put \= later or last in your predicates.
* The ; (semicolon) represents the logical OR.

**Expressions**

|  |  |  |
| --- | --- | --- |
| **Meaning** | **Notation** | **Opposite** |
| T1 & T2 unify | T1 = T2 | \= |
| T1 & T2 identical | T1 == T2 | \== |
| T1 becomes the evaluation of T2 | T1 is T2 | undef. |
| T1, T2 evaluate to same number | T1 =:= T2 | =\= |

**Comparing**

|  |  |
| --- | --- |
| **Arithmetic Examples** | **Prolog Notation** |
| x < y | X < Y. |
| x ≤ y | X =< Y. |
| x = y | X =:= Y. |
| x ≠ y | X =\= Y. |
| x ≥ y | X >= Y |
| x < y | X > Y |

**Built in Predicates (allowed in exams)**

* member(X, L). Checks if element X is in list L.
* length(List, L). Returns length L of List.
* append(L1, L2, L3). L3 = L1 + L2.
* \=. Equivalent to Not.

**Aggregates**

* findall(T, Goal, List):-
  + List is the list of all instances of T for which Goal succeeds.
    - List = [T1, T2, T3, T4, …, TN]…
  + If fail returns L = []; (empty list)
  + Instance of T may appear several times to prove Goal
* setof(T, Goal, List):-
  + List is the sorted set of all instances of T such that Goal succeeds.
  + Splits into groups of each successive T
    - Eg. for Var = X, L = [].
  + On failure returns no.
  + Sorts list List in ascending order with duplicates removed.
  + If you want to return [] when fail, use \+setof in your predicate.
* Existential Quantifiers (^)
  + Eg setof(T, V^Goal, List).
  + To mean variable V is existentially quantified, will not split into groups of successive V’s and hence will all be contained in one single list.
* Use ‘-‘(hyphen/ dash) to separate variables when you want a map-like relationship. For example:

mother\_of\_the\_youngest(M):-

setof(A-Mum, C^(child\_mother(C, Mum), age(C, A)), [A-M|\_]).

Maps Age-Mum together and retrieves just M of the first element of the sorted list returned by setof.

**Working with Lists**

* When comparing heads of two lists, it is easier to break predicate down to two cases when H1 < H2 and H1 >= H2 and work separately with each.
* INCLUDE THIS >> :- use\_module(library(lists)).

**Accumulators**

* Simply a 'variable’ passed on in recursion to store temporary values
* Enables tail recursion; less computational time therefore efficiency.
  + eg. Factorial(5) requires Factorial(4), so the value could be stored rather than having to recalculate the whole bunch.