4 Prolog (ijl20)

In your answers ensure each relation which you define has a comment giving a declarative reading of its behaviour. Avoid unnecessary use of *cut* or other extra-logical relations. The library relations `\=`, `is`, and `atomic(X)` may be used, the last succeeding if `X` is a number or atom e.g. `42` or `abc`. Other relations should not be assumed.

(*a*) Define a relation `eval/2` to reduce arithmetic terms, so that e.g. `eval(1+2*3,N)` succeeds with `N=7`. Atoms should reduce to themselves, so e.g. `eval(a,Ans)` succeeds with `Ans=a`. [4 marks]

(*b*) Extend `eval` to allow function calls within arithmetic terms, such that `2 * apply(inc, [2+3])` reduces to `12`. We will declare functions as Prolog facts in a `fun/2` relation, e.g. `inc` above will be specified by the fact `fun(apply(inc,[N]), N+1)`. Note the function arguments are held in a list, `[N]` in this example, to support multi-argument functions. [6 marks]

(*c*) Extend relation `eval` to support a `==/2` operator, which reduces term `A==B` to `true` if `A` and `B` reduce to the same number or atom and `false` otherwise. For example `eval(1+3==2+2,Ans)` succeeds with `Ans=true`. [3 marks]

(*d*) Extend relation `eval` to support terms of the form `if(Condition,Then,Else)`. These `if/3` terms should reduce to the reduction of either the `Then` term or the `Else` term determined by `Condition` reducing to `true` or `false`. For example `eval(if(1+2==4+5,a,b),Ans)` succeeds with `Ans=b`. Add a fact to the `fun` relation specifying the *factorial* function such that `eval(apply(fact,[5]),N)` succeeds with `N=120`. [7 marks]