Assignment 7: Type Checker in Prolog

In this assignment, you will write a type-checker for a simple functional language.

You need to write a Prolog predicate *hastype(Gamma, E, T)*, where

- *Gamma* is a list of variable-type pairs, representing type assumptions on variables
- *E* is an object language expression,
- *T* is a type.

This predicate is mutually recursively defined with another Prolog predicate

typeElaborates(Gamma, D, Gamma')

where *D* is a definition.

E ranges over (at least)

- variables, modelled as say *variable(X)*
- constants, both numerical and boolean (at least)
- arithmetic operations over numerical expressions
- boolean operations over boolean expressions
- comparison operations over numerical expressions
- equality over arbitrary expressions, where equality can be decided
- conditional expressions **if_then_else**
- qualified expressions of the form **let** *D* **in** *E* **end**
- function abstractions *\X.E* with functions as first-class citizens
- function application (*E1 E2*)
- *n*-tuples *(n >= 0)*
- expressions using projection operations.
- ....possible *extensions to constructors, and case analysis expressions*

and

D ranges over (at least)

- simple definitions *X* =**def**= *E*
- sequential definitions *D1; D2*
- parallel definitions *D1* || *D2*
- local definitions **local** *D1* **in** *D2* **end**
- ... *possible extension to recursive definitions*

and

T ranges over (at least)

- Type variables modelled as say TypeVar(A)
- Base types **tint**, **tbool**, ...
- Arrow types *T1 -> T2* |
- cartesian product types *T1 * ... * Tn (n>1)*
- ... *possible extension to union types and recursive types*...

--

You will need to define suitable constructors for expressions, definitions, types, etc.

You need to provide enough test examples to show your type checker works correctly.

Note that this checker can work as a type inference engine. However it does not work for polymorphic type inference. Show with counter-examples that this is the case.