CS322 Fall 1999
Module 8 (Metainterpreters)
Assignment 8

Solution.

In this assignment you will implement, in CILog, a new programming language **ArLog** that allows for adjustable parameters in arithmetic expressions and user-defined arithmetic functions.

An ArLog program is a set of clauses of the form:

```
H <= B.
```

where *H* is an atom and *B* is a body.

A body is either of the form:

- `true`
- `A & B` where *A* and *B* are bodies
- `A` where *A* is an atom defined by rules
- `X is Exp` where *X* is a number and *Exp* is a parametrized arithmetic expression. This is true if *X* is the value of the expression *Exp*.
- `assign(P,Exp)` where *P* is a parameter and *Exp* is a parametrized arithmetic expression. This assigns the value of *Exp* to the parameter *P*.
- `E1 > E2` where *E*1 and *E*2 are parametrized arithmetic expressions. (This, "assign" and "is" are the only built-in relations).

Parametrized expressions are of the form:

- `A+B`, `A*B`, `A-B`, `A/B` where *A* and *B* are parametrized expressions
- `N` where *N* is a number
- `P` where *P* is a parameter (a CILog constant).
- a user-defined function

Parametrized expressions are always evaluated in an environment, where an environment is a list of terms of the form `val(P,V)` where *P* is a parameter and *V* is a number.

We always prove goals within an environment, but the environment can be updated with an *assign* goal. In a conjunction, the rightmost conjunct is evaluated in the environment that is the result of the evaluation of the leftmost conjunct.

As well as normal clauses defining atoms, the user can define functions using:

```
F=Exp <= B.
```

Where F is a user-defined function and *Exp* is an expression. If B is true, the value of *Exp* is the value that F evaluates to.

Question 1

Define *eval*(*Exp*, *Val*, *Env*) that is true if expression *Exp* evaluates to *Val* in environment *Env*. [You do not need to worry about user-defined functions for this question.]

For example, the query

```
ask eval(3*a+2*b+c, Val, [val(a,4),val(b,5),val(c,7)]).
```

should return $Val = 25$.

The only CILog built-in predicates you may use are *number* and *is*.

Solution

eval(*Exp*, *Val*, *Env*) is true if expression *Exp* evaluates to *Val* in environment *Env*.

```
eval(N,N,E) <-
   number(N).
eval(C,V,E) <-
   member(val(C,V),E).
eval((X+Y),V,E) <-
   eval(X,XV,E)&
   eval(Y,YV,E)&
   V is XV+YV.
eval((X*Y),V,E) <-
   eval(X,XV,E)&
   eval(Y,YV,E)&
   V is XV*YV.
eval((X-Y),V,E) <-
   eval(X,XV,E)&
   eval(Y,YV,E)&
   V is XV-YV.
eval((X/Y),V,E) <-
   eval(X,XV,E)&
   eval(Y,YV,E)&
   V is XV/YV.
```

member(E, L) is true if E is a member of list L.

```
member(E,[E|R]).
member(E,[H|T]) <-
   member(E,T).
```

Question 2

Define *update*(*Par*, *Val*, *Env*1, *Env*2) that is true if *Env*2 is the same as environment *Env*1 except that *Par* has the value *Val*. You can assume that *Par* is already assigned a value in *Env*1.

For example, the query

```
ask update(b,9,[val(a,4),val(b,5),val(c,7)],E2)
```

should return $E2 = [val(a, 4), val(b, 9), val(c, 7)]$.

Solution

update(*Par*, *Val*, *Env*1, *Env*2) is true if *Env*2 is the same as environment *Env*1 except that *Par* has the value *Val*.

```
update(X,Val,[val(X,OV)|Rest],[val(X,Val)|Rest]).
update(X,Val,[val(X1,V1)|Rest0],[val(X1,V1)|Rest1]) <-
   update(X,Val,Rest0,Rest1).
```

Question 3

Define *arprove*(*Body*, $E1$, $E2$) that is true if the *Body* can be proved with initial environment $E1$ and resulting environment $E2$. The only CILog built-in predicate you can use is $>$.

For example, suppose the knowledge base is:

```
addato(X,Y) <= Y is X+a.
foo(X,Y) <= X is a+3 & assign(a,X) & Y is a+3.
```

The query

```
ask arprove(addato(3,Y),[val(a,4),val(b,5),val(c,7)],E2).
```

should return $Y = 7$ and $E = [val(a, 4), val(b, 5), val(c, 7)]$.

The query

```
ask arprove(foo(X,Y),[val(a,4),val(b,5),val(c,7)],E2).
```

should return $X = 7$, $Y = 10$, $E2 = [val(a, 7), val(b, 5), val(c, 7)]$.

Solution

arprove(*Body*, $E1$, $E2$) is true if the *Body* can be proved with initial environment $E1$ and resulting environment $E2$. The only CILog built-in predicate you can use is $>$.

```
arprove(A&B,E1,E3) <-
   arprove(A,E1,E2) &
   arprove(B,E2,E3).
arprove(G,E1,E2) <-
   (G <= B) &
   arprove(B,E1,E2).
```

```
arprove(true,E,E).
arprove((V is Exp),E1,E1) <-
   eval(Exp,V,E1).
arprove(assign(X,Exp),E1,E2) <-
   eval(Exp,Val,E1) &
   update(X,Val,E1,E2).
arprove(X > Y,E1,E1) <-
   eval(X,XV,E1) &
   eval(Y,YV,E1) &
   XV > YV.
```

Question 4

[Optional] Modify *eval* (add a new clause) to allow for user-defined functions.

For example, consider the clauses:

```
sumsq(X,Y)=X*X+Y*Y <= true.
fact(N)=N*fact(N1) <= N>0 & N1 is N-1.
fact(0)=1 <= true.
```

then query

```
ask arprove(X is sumsq(3,4)-fact(4),[],E2).
```

should return $X = 1, E2 = []$.

Solution

```
eval(UD,V,E) <-
   (UD=Exp <= B) &
   arprove(B,E,_) &
   eval(Exp,V,E).
```