Arithmetic in Prolog

If you read Chapter 3 and feared that you were going to have to define the natural numbers and all the basic operators just to do simple arithmetic, you're in luck.

Prolog implementers provide you with predefined system predicates to let you get at the arithmetic capabilities that already exist in your computer.

You should read all about arithmetic in Chapter 8.

Arithmetic in Prolog

Prolog arithmetic is, in a word, unusual.

In other programming languages, an arithmetic expression is evaluated wherever it is found. In Prolog, an arithmetic expression is just a predicate with arguments written in an infix notation for convenience. Prolog doesn't evaluate anything. To evaluate an arithmetic expression in Prolog, we have to call on extra-logical system predicates that make our Prolog "impure".

Arithmetic in Prolog

For example, 2 + 3 may automatically evaluate to 5 in other languages, but Prolog sees it as just another way to write the predicate +(2,3). Pure Prolog will let you unify that term with another, but it won't evaluate the expression.

```
?- 2 + 3 = 2 + 3.
true.

?- 2 + 3 = +(2,3).
true.

?- 2 + 3 = 5.
false.

?- 2 + 3 = 3 + 2.
false.
```

Arithmetic in Prolog

To force the evaluation, the expression must be the right-hand argument of an is operator...

```
?- X is 2 + 3.
X = 5.

?- 5 is 2 + 3.
true.

?- 2 + 3 is 2 + 3.
false.
```

Arithmetic in Prolog

...or the expression must be either the left-hand or right-hand argument of a relational operator.

```
?- 3 < 5.
true.

?- 5 < 3.
false.

?- 4 + 1 > 4 - 1.
true.

?- 2 + 3 =:= 2 + 3.
true.

?- 3 + 2 =:= 2 + 3.
true.
```

Arithmetic in Prolog

is forces the evaluation of an expression on its right hand side and then unifies the result with its left hand side (a variable). Variables in the expression on the right hand side must be instantiated (unified with a numeric value)

```
?- X is 1 + 2.
X = 3.

?- Y is 3 + 2, X is Y + 1.
Y = 5,
X = 6.

?- X is 5/2, Y is 5//2, Z is 5 mod 2.
X = 2.5,
Y = 2,
Z = 1.
```

Arithmetic in Prolog

is forces the evaluation of an expression on its right hand side and then unifies the result with its left hand side (a variable). Variables in the expression on the right hand side must be instantiated (unified with a numeric value)

```
?- X is 1 + 2.
X = 3.
```

Note once again that variables in Prolog aren't true variables in the strict vonNeumann sense, and is is not an assignment operator.

This is still just unification.

Arithmetic in Prolog

Keep in mind the differences between these operators, which can be confusing:

is takes an expression (on the right), evaluates it, and unifies the result with its argument on the left

=:= this is one of several relational operators, each of which evaluates the expressions on both sides, compares the results, and returns true or false accordingly

= which unifies two terms (which need not be expressions and, if expressions, will not be evaluated)

Arithmetic in Prolog

Here are some of the basic relational operators (for more, see section 4.26 of the SWI-Prolog Reference Manual):

X > Y	X is greater than Y
X < Y	X is less than Y
X >= Y	X is greater than or equal to Y
X =< Y	X is less than or equal to Y
X =:= Y	the values of X and Y are equal
X =\= Y	the values of X and Y are not equal

Arithmetic in Prolog

Here are some of the basic arithmetic operators (for more, see section 4.26 of the SWI-Prolog Reference Manual):

+ addition
- subtraction
* multiplication
/ division
// integer division
mod modulo, the remainder of integer division

Arithmetic in Prolog

Things to keep in mind about Prolog arithmetic

Arithmetic in Prolog

Things to keep in mind about Prolog arithmetic

- There are built-in, non-pure, procedures for arithmetic.

Arithmetic in Prolog

Things to keep in mind about Prolog arithmetic

- There are built-in, non-pure, procedures for arithmetic.
- Arithmetic operations have to be explicitly requested by the built-in procedure is or a relational operator.

Arithmetic in Prolog

Things to keep in mind about Prolog arithmetic

- There are built-in, non-pure, procedures for arithmetic.
- Arithmetic operations have to be explicitly requested by the built-in procedure is or a relational operator.
- At the time that evaluation is carried out, all arguments on the right of the is or on either side of the relational operator must be already instantiated to numbers (it's all about unification and the interpreter).

Arithmetic in Prolog

Things to keep in mind about Prolog arithmetic

- There are built-in, non-pure, procedures for arithmetic.
- Arithmetic operations have to be explicitly requested by the built-in procedure is or a relational operator.
- At the time that evaluation is carried out, all arguments on the right of the is or on either side of the relational operator must be already instantiated to numbers (it's all about unification and the interpreter).
- The values of arithmetic expressions can be compared by operators such as <, =<, etc. Relational operators force the evaluation of both their arguments.

Arithmetic in Prolog

One other thing -- using these arithmetic features in your Prolog procedure means you may not have a multi-purpose procedure.

So this works:

```
append([],X,X).
append([H1|T1],X,[H1|T2]) :- append(T1,X,T2).

?- append([a,b],X,[a,b,c,d]).

X = [c, d]
```

Arithmetic in Prolog

One other thing -- using these arithmetic features in your Prolog procedure means you may not have a multi-purpose procedure.

So this works:

```
add(X,Y,Z) :- Z is X + Y.

?- add(3,5,Z).
Z = 8.
```

But this doesn't work:

```
?- add(3,Y,8).
ERROR: is/2: Arguments are not sufficiently instantiated
```

Arithmetic in Prolog

Why doesn't it work?

Arithmetic in Prolog

Why doesn't it work?

First, variables on the right hand side of the `is` must be instantiated. `Y` is not instantiated. Why must everything be instantiated? Only the expression `X + Y` (where `X` is unified with `3`) is being handed off to an extra-logical arithmetic routine. If I said to you "Here's the expression `3 + Y` -- now tell me the value of `Y`" you'd tell me there's just not enough information there.

Arithmetic in Prolog

Why doesn't it work?

Second, the multi-purpose use of `append` comes from a search through a finite set of rules to find the substitutions that will make the query true. The rules tell Prolog what the possibilities are and constrain the search. There are no rules to constrain the search with our `add` procedure. All Prolog might do is try all possible numeric values for `Y` and see if one of them makes the query true. On average, that will take forever, so why try?

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

You need to prove mylength([ ],0)

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

You need to prove mylength([ ],0) That's easy

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

You need to prove mylength([ ],0) That's easy

```
mylength([ ],0).
```

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

You need to prove mylength([ ],0) That's easy

```
mylength([ ],0).
```

Now what?

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

You need to prove mylength([ ],0) That's easy

```
mylength([ ],0).
mylength(      , ) :- mylength( ,   ),
                      ?? is ?? ?? ??
```

Now what?

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

You need to prove mylength([ ],0) That's easy

```
mylength([ ],0).
mylength([H|T], ) :- mylength( ,  ),
                     ?? is ?? ?? ??
```

Now what?

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

You need to prove mylength([ ],0) That's easy

```
mylength([ ],0).
mylength([H|T], ) :- mylength(T,  ),
                     ?? is ?? ?? ??
```

Now what?

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

You need to prove mylength([ ],0) That's easy

```
mylength([ ],0).
mylength([H|T],N) :- mylength(T,   ),
                     ?? is ?? ?? ??
```

Now what?

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

You need to prove mylength([ ],0) That's easy

```
mylength([ ],0).
mylength([H|T],N) :- mylength(T,N-1),
                     ?? is ?? ?? ??
```

Now what?

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

You need to prove mylength([ ],0) That's easy

```
mylength([ ],0).
mylength([H|T],N) :- mylength(T,N-1),
                     ?? is ?? ?? ??
```

Now what?

NO!!

Arithmetic example with recursion

Why? Because mylength is a logical predicate. You can't pass an arithmetic expression to a logic predicate and expect something good to happen.

Recall that arithmetic in Prolog is extra-logical (not done by pure Prolog but by separate arithmetic procedures).

```
mylength([ ],0).
mylength([H|T],N) :- mylength(T,N-1),
                     ?? is ?? ?? ??
```

Now what?

NO!!

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

You need to prove mylength([ ],0) That's easy

```
mylength([ ],0).
mylength([H|T],N) :- mylength(T,N1),
                     ?? is ?? ?? ??
```

Now what?

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

You need to prove mylength([ ],0) That's easy

```
mylength([ ],0).
mylength([H|T],N) :- mylength(T,N1),
                     N  is ?? ?? ??
```

Now what?

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

You need to prove mylength([ ],0) That's easy

```
mylength([ ],0).
mylength([H|T],N) :- mylength(T,N1),
                     N  is N1 ?? ??
```

Now what?

Arithmetic example with recursion

How do you find the number of (top-level) elements in a list? For example, how could you prove

mylength([a,b,c],3) ?

You need to prove mylength([b,c],2)and 3 = 2 + 1

You need to prove mylength([c],1) and 2 = 1 + 1

You need to prove mylength([ ],0) That's easy

```
mylength([ ],0).
mylength([H|T],N) :- mylength(T,N1),
                     N  is N1 +  1.
```

Now what?

Questions?

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent =

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {H = a,T = [],N = X}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {H = a,T = [],N = X}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength(T,N1), N is 1 + N1

Θ = {H = a,T = [],N = X}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength(T,N1), N is 1 + N1

Θ = {H = a,T = [],N = X}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([],N1), X is 1 + N1

Θ = {H = a,T = [],N = X}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([],N1), X is 1 + N1

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([],N1), X is 1 + N1

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([],N1), X is 1 + N1

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([],N1), X is 1 + N1

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([],N1), X is 1 + N1

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([],N1), X is 1 + N1

Θ = {N1 = 0}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause $A' :- B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = mylength([],N1), X is 1 + N1

Θ = {N1 = 0}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = X is 1 + N1

Θ = {N1 = 0}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = X is 1 + N1

Θ = {N1 = 0}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = X is 1 + 0

Θ = {N1 = 0}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = X is 1 + 0

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = X is 1 + 0

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 +
N1.
```

?- mylength([a],X).

resolvent = X is 1 + 0

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 + N1.
```

?- mylength([a],X).

resolvent = X is 1 + 0

Θ = {}

Prolog has to handle "goals" like X is 1 + 0 differently. It's not intended to unify with the head of a rule in P. It's a directive to Prolog to evaluate the arithmetic expression to the right of the 'is' and add the resulting equivalence to Θ.

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause $A' :- B_1,\ldots,B_n$
    from program *P* such that *A* and *A'* unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 + N1.
```

?- mylength([a],X).

resolvent = X is 1

Θ = {}

Prolog has to handle "goals" like X is 1 + 0 differently. It's not intended to unify with the head of a rule in P. It's a directive to Prolog to evaluate the arithmetic expression to the right of the 'is' and add the resulting equivalence to Θ.

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 + N1.
```

?- mylength([a],X).

resolvent =

Θ = {X = 1}

Prolog has to handle "goals" like X is 1 + 0 differently. It's not intended to unify with the head of a rule in P. It's a directive to Prolog to evaluate the arithmetic expression to the right of the 'is' and add the resulting equivalence to Θ.

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 + N1.
```

Prolog has to handle "goals" like X is 1 + 0 differently. It's not intended to unify with the head of a rule in P. It's a directive to Prolog to evaluate the arithmetic expression to the right of the 'is' and add the resulting equivalence to Θ.

?- mylength([a],X).

resolvent =

Θ = {X = 1}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 + N1.
```

Prolog has to handle "goals" like X is 1 + 0 differently. It's not intended to unify with the head of a rule in P. It's a directive to Prolog to evaluate the arithmetic expression to the right of the 'is' and add the resulting equivalence to Θ.

?- mylength([a],1).

resolvent =

Θ = {X = 1}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify with mgu Θ
    (if no such goal and clause exist, exit the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 + N1.
```

?- mylength([a],1).

resolvent =

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 + N1.
```

?- mylength([a],1).

resolvent =

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N is 1 + N1.
```

?- mylength([a],1).

resolvent =

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Questions?

But wait…

Why won't this work?

```
mylength([],0).
mylength([H|T],N) :-
   mylength(T,N1), N1 is N - 1.
```

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent =

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {H = a,T = [],N = X}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([a],X)

Θ = {H = a,T = [],N = X}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength(T,N1), N1 is N - 1

Θ = {H = a,T = [],N = X}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength(T,N1), N1 is N - 1

Θ = {H = a,T = [],N = X}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([],N1), N1 is X - 1

Θ = {H = a,T = [],N = X}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([],N1), N1 is X - 1

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([],N1), N1 is X - 1

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
choose a goal *A* from the resolvent
choose a (renamed) clause A' :- $B_1,\ldots,B_n$
from program *P* such that *A* and A' unify with mgu Θ
(if no such goal and clause exist, exit the while loop)
replace *A* by $B_1,\ldots,B_n$ in the resolvent
apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([],N1), N1 is X - 1

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([],N1), N1 is X - 1

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause $A' :- B_1,\ldots,B_n$
    from program *P* such that *A* and *A'* unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([],N1), N1 is X - 1

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([],N1), N1 is X - 1

Θ = {N1 = 0}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause $A' :- B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = mylength([],N1), N1 is X - 1

Θ = {N1 = 0}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = N1 is X - 1

Θ = {N1 = 0}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = N1 is X - 1

Θ = {N1 = 0}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = 0 is X - 1

Θ = {N1 = 0}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = 0 is X - 1

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = 0 is X - 1

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N -
1.
```

?- mylength([a],X).

resolvent = 0 is X - 1

Θ = {}

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify
    with mgu Θ
    (if no such goal and clause exist, exit
    the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N - 1.
```

?- mylength([a],X).

resolvent = 0 is X - 1 OOPS!

ERROR: Arguments are not sufficiently instantiated

Sadly, the variable X is not instantiated on the right side of the 'is' operator.

Initialize resolvent to goal *G* (the query)
while resolvent not empty *do*
  choose a goal *A* from the resolvent
  choose a (renamed) clause A' :- $B_1,\ldots,B_n$
    from program *P* such that *A* and A' unify with mgu Θ
    (if no such goal and clause exist, exit the while loop)
  replace *A* by $B_1,\ldots,B_n$ in the resolvent
  apply Θ to the resolvent and to *G*
If the resolvent is empty, *then* output *G*,
  else output *no*

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N - 1.
```

?- mylength([a],X).

resolvent = 0 is X - 1 OOPS!

ERROR: Arguments are not sufficiently instantiated

As your book notes one could enhance Prolog to do the necessary manipulations to put the instantiated parts on the right, the variable on the left, and solve the equation.

Simple Prolog interpreter

The program (*P*):

```
mylength([],0).

mylength([H|T],N) :-
  mylength(T,N1), N1 is N - 1.
```

?- mylength([a],X).

resolvent = 0 is X - 1 OOPS!

ERROR: Arguments are not sufficiently instantiated

How hard could it be to write some sort of widget for Prolog that, upon encountering 0 is X - 1, says *"Hmmm, there's an uninstantiated variable on the right side of that 'is'...that's probably just the typical arithmetic mistake that Prolog programmers make...hey! I'm an incredibly powerful symbol manipulator...can I find a way to manipulate that equation so that the variable is on the left, everything else is on the right, and then move forward with this computation (after letting the user know, of course)?"*

Questions?