MODLOG

Modular **O**bject-Oriented **D**eclarative **LOG**ic

USER MANUAL

R. Alan Whitehurst, John Pietrzak
U.S. Army Construction Engineering Research Laboratory
P.O. Box 9005
Champaign, IL

June 15, 1993

Abstract

This report is intended to serve as a user's manual for the ModLog extension to the ModSim programming language.

Forward

MODLOG was created for our sponsor.

Contents

List of Figures

Chapter 3

Tutorial

3.1 Introduction

MODLOG, as all declarative programming systems, is a language intended primarily for symbolic (i.e., non-numeric) computation. It is particularly well suited to its role in object-oriented simulations, because it allows objects and the relationship between objects to be specified and reasoned about. Using MODLOG is like using Prolog; and someone familiar with one system should be able to specify declarative programs in the other with little or no trouble. In this chapter, we present a number of small examples to help introduce programming in the MODLOG system.

As our first example, let us consider the problem of calculating the number of elements in a list. Before we get into the definition of list length, we need to know how lists are represented in MODLOG. The syntax for lists in MODLOG is a comma separated list enclosed by square brackets. Therefore, the list containing the symbols `a, b,` and `c` would be written as: `[a,b,c]`. A special notation is also provided when writing list patterns to split a list into head/tail components: a vertical bar followed by a variable will cause the tail of the list to be bound to the variable during unification. As an example, unifying the pattern `[X|R]` with `[a,b,c]` would produce these bindings: `X=a, R=[b,c]`.

Now, back to our example problem. As opposed to procedural programming, where we would begin by specifying *how* to calculate the length of a list, in MODLOG we focus of *what* it means. We will use an inductive strategy to capture the semantics of list length. As in an induction proof, we begin by capturing the *base case*. For `length` the base case is that of the empty list; and the length of the empty list is zero. Next, we consider the inductive step: given that we know the length N of an arbitrary list, what can we say about a list with one more element: we know its length is $N+1$. So, put in the syntax of MODLOG, this definition becomes:

```
length([],0).
if length(L,N),inc(N,N1) then length([_|L],N1).
```

The first line represents the fact that the length of the empty list is always zero. The second line specifies the relationship between the length N of an arbitrary list L, and the length of a list formed by adding some unspecified element ("_") to L. Here we see the use of the MODLOG primitive function `inc`, which is short for *increment*, and states that N1 is the value of N incremented by one. We can now introduce these facts and rules into MODLOG, and test our definition:

```
>length([],0).
TRUE

>if length(L,N),inc(N,N1) then length([_|L],N1).
TRUE

>listing(length)?
```

```
length([],0).
IF length(L,N),inc(N,N1) THEN length([_|L],N1).
TRUE
```

Here, we have entered the two relations into the systems, and verified their presence by listing all the definitions of the predicate `length`. At each step, the system responds with `TRUE`, signifying that it has accepted and successfully completed our requests. Next, we present some *queries* to the system, to test the new rules:

```
>length([],X)?
TRUE
X=0

>length(X,0)?
TRUE
X=[]

>length([a,b,c,d,e],X)?
TRUE
X=5

>length([a,b,c],3)?
TRUE


>length(X,4)?
TRUE
X=[_1,_2,_4,_7]
```

Here, we have presented five queries. The first requests that the system find the length of the empty list, returning that length in the variable X. The system responds with `TRUE` {`X=0`}. Next, we request that the system find a list whose length is zero, the system responds by returning the empty list. Then, we request the length of a list consisting of a, b, c, d and e. The system correctly responds with the answer 5. Next, we ask if the length of the list consisting of a, b, and c is 3. The system confirms that this is the case by responding with `TRUE`. Finally, we ask the system to construct a list with four elements; it returns a list with four arbitrary elements, where the arbitrary element is denoted by the underscore (_).

It is instructive to see how MODLOG goes about using these definitions to actually derive the length of a list. If we turn tracing on in the MODLOG interpreter (by entering the query: `trace?`), and then submit the query `length([a,b,c,d],X)?`, the system responds with:

```
>length([a,b,c,d],X)?
Trying to prove: |length([a,b,c,d],X)
Trying to prove: ||length([b,c,d],N1)
Trying to prove: |||length([c,d],N2)
Trying to prove: ||||length([d],N3)
Trying to prove: |||||length([],N4)
Trying to prove: |||||inc(0,N14)
Trying to prove: ||||inc(1,N13)
Trying to prove: |||inc(2,N12)
Trying to prove: ||inc(3,N11)
no goals left
TRUE
{X=4}
```