Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J. Heering, P. Klint

The efficiency of the Equation Interpreter compared with the UNH Prolog interpreter

Department of Computer Science Report CS-R8509 April

The efficiency of the Equation Interpreter compared with the UNH Prolog interpreter

J. Heering
P. Klint
Centre for Mathematics and Computer Science
Amsterdam

There are several alternatives for transforming algebraic specifications into executable prototypes. In this note the Equation Interpreter (a rewrite rule interpreter) and the University of New Hampshire Prolog interpreter are viewed as target systems for executing prototypes. The efficiencies of these systems are compared with each other.

Key words & phrases: Performance Evaluation, Prolog, Equation Interpreter, Prototyping, Algebraic Specification.

1983-84 CR Categories: D.2.1, F.4.1, D.3.4.

1980 Mathematics Subject Classification: 68B99. 69D21, 69D44, 69F41

Note: Partial support received from the European Communities under ESPRIT project no. 348 (Generation of Interactive Programming Environments).

Note: This report has been submitted for publication elsewhere.

1. Motivation

Transforming formal specifications into executable prototypes has several applications: one can either use the executable prototype to validate the specification or one may be interested in using the prototype system itself. Two alternatives for deriving executable prototypes from *algebraic* specifications are

(1) transform the specification into a complete (conditional) term rewriting system and execute it by means of an existing rewrite rule interpreter;

(2) transform the specification into a set of Horn clauses and use an existing Prolog system for their execution.

Here, we are interested in the relative efficiency of the end products which can be obtained along these two lines using the Equation Interpreter [HOD82a, HOD82b] and the UNH Prolog interpreter from the University of New Hampshire, respectively.

Two issues will *not* be addressed:

Report CS-R8509
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

(1) The way in which an algebraic specification can be transformed into either a term rewriting system or Horn clauses.

(2) The relative merits of either the Equation Interpreter or Prolog as programming systems *per se*.

We restrict ourselves to the relative efficiency of both systems considered as (abstract) computing machines.

In the remainder of this note the measurement method and the measurements themselves are described and some conclusions are drawn. The appendices give detailed information on the programs used for the measurements.

2. Measurement method

The efficiency of the Equation Interpreter and Prolog have been compared by executing a series of examples using both systems. Each example consists of a program and input for that program. The listings of the programs, the input, and resulting output are given in the appendices. In choosing the examples we had to avoid violating implementation limitations of the systems involved. We have avoided, for instance, very long input expressions (which cause overflow of the parse stack used in the Equation Interpreter), input expressions using too many different variables (a restriction of the Prolog interpreter), or too many user defined symbols (a restriction of the Equation Interpreter). Any of these limitations could have been removed by increasing the relevant parameter in each system, but we decided not to do that and to use the standard version.

The examples are now described in more detail. The first program (EMPTY) is the empty program. It serves to measure the initialisation times for both systems.

The second program (REV) performs list reversal. It reads a list of 7 elements from input and replicates it 16 times. The resulting list of 112 elements is reversed two times and finally its length is determined. This program serves to measure the processing of large data structures.

The third program (ACK) computes Ackermann's function for the value (3,2). This program serves to measure the speed of recursion and integer arithmetic.

The fourth program (ALPHA) is actually a series of programs of increasing size. These programs define an alphabet of N characters with an equality predicate. Each program defines the Boolean functions `and` and `or`, the conditional function `if`, and the successor (`succ`) and equality (`eq_INTEGER`) functions on natural numbers. For given N, each program defines N constants (representing the characters in the alphabet), a function `ord` that injects these constants in the integers, and an equality function on characters (`eq_CHAR`) that is defined by means of `ord` and `eq_INTEGER`. The input for each program is a conditional expression containing fifteen applications of `eq_CHAR` with the fifteen last characters in the alphabet as argument; this conditional expression returns the last character in the alphabet as value. This program has as purpose to measure the effect of an increasing number of equations on the time needed for preprocessing and for execution.

Measurements have been performed on a VAX11/780 with Berkeley Unix Version 4.2. We used the first distribution of the Equation Interpreter dated 5-16-83 and version 1.3 of UNH Prolog from the University of New Hampshire.

Initial experiments showed that the timing of the Equation Interpreter presented problems due to the fact that it has been implemented as a pipeline of five concurrent processes: two preprocessors, the actual interpreter and two postprocessors. This organisation makes the timing highly sensitive to the scheduling of the individual processes in the pipeline. To avoid these fluctuations, we have replaced the pipeline by a sequence of five processes. This causes a slight increase in the execution times measured, but we observed that the execution time of the whole system is completely dominated by the execution time of the actual interpreter (this accounts for more than 95% of the total execution time).

3. Measurements

The results of the experiment are summarized in Table I. Preprocessing times have been measured 5 times. Execution times have been measured 10 times. The table gives the averages of these measurements in seconds. The standard deviation, expressed as percentage of the average of each series of measurements, never exceeded 6%.

Example	Equation Interpreter		Prolog
	Preprocessing time t_E	Execution time T_E	Execution time T_P
EMPTY	136.5	2.1	0.2
REV	172.6	61.4	50.5
ACK	155.5	18.5	3.6
ALPHA (N=15)	331.9	7.8	4.3
ALPHA (N=20)	428.5	10.7	7.3
ALPHA (N=25)	528.3	13.7	9.7
ALPHA (N=30)	688.0	16.3	12.9
ALPHA (N=40)	911.2	22.1	19.7
ALPHA (N=50)	1278.4	28.5	27.5
ALPHA (N=60)	1681.1	34.1	33.3
ALPHA (N=70)	2168.4	39.8	41.7
ALPHA (N=80)	2668.1	45.5	54.4
ALPHA (N=90)	3277.0	50.8	62.7

Table I. Summary of measurements.

Preprocessing times will be denoted by t and execution times by T. The total preprocessing time t_E of the Equation Interpreter includes syntactic and semantic checking of the input program, generation of an equivalent Pascal program (which includes tables for fast pattern matching of terms at execution time) and compilation of this program. This compilation time varies between 95 and 160 seconds in the above examples. T_E indicates the execution time of the Equation Interpreter.

The Prolog system does no preprocessing, i.e. $t_P=0$. The execution times T_P given include the time needed by the Prolog system to read the example programs.

4. Conclusions

(1) It is surprising that a system without preprocessing performs so well as compared with a system with extensive preprocessing.

(2) The preprocessing time t_E of the Equation Interpreter tends to become prohibitive. The trends in the measurements suggest that the Equation Interpreter outperforms Prolog on large sets of equations. It depends on the particular application which system should be chosen. In the case of prototyping the same program will probably only be executed a few times. In that case, the disadvantage of considerable preprocessing time outweighs the advantage of the shorter execution time. If the number of executions is larger than $n_0=\frac{t_E-t_P}{T_P-T_E}$ the large preprocessing time of the Equation Interpreter starts to pay off. In example ALPHA, $n_0=1141$, 300 and 275 for $N=70$, 80, 90, respectively.

(3) All Prolog programs in the measurements were *interpreted* and not compiled. If compilation instead of interpretation will be used one may expect a speed up of the execution time by a factor between 5 and 15.

5. References

[HOD82a] Hoffmann, C.M. & O'Donnell, M.J., "Programming with equations", *ACM Transactions on Programming Languages and Systems*, **4** (1982)1, 83-112.

[HOD82b] Hoffmann, C.M. & O'Donnell, M.J., "Pattern matching in trees", *Journal of the ACM*, **29** (1982), 68-95.

[OD77] O'Donnell, M.J., *Computing in Systems Described by Equations*, Lecture Notes in Computer Science **58**, Springer-Verlag, Berlin, 1977.

Appendix I: EMPTY

I.1 Equational program

```
Symbols
        a:0;
        noop:1.
For all x:
        noop(x) = x.
```

I.2. Input

```
a
```

I.3. Output

```
a
```

I.4. Prolog program

```
(* empty program *)
```

I.5. Input

Note: all Prolog programs are assumed to reside on the file "prodef".

```
[prodef].
```

I.6. Output

```
                     --- UNH Prolog 1.3 ---

| ?-
[ prodef consulted ]



yes
| ?-
```

Appendix II: REV

II.1 Equational program

```
Symbols
        cons:   2;
        nil:    0;
        rev:    1;
        append: 2;
        repl2:  1;
        repl4:  1;
        repl16: 1;
        length: 1;
        job:    1;
        add:    2;
        include atomic_symbols;
        include integer_numerals.

For all x, y, z, h, t, l:

        include addint;

        append(nil, x)           = cons(x, nil);
        append(cons(x, y), z)    = cons(x, append(y, z));
        rev(nil)                 = nil;
        rev(cons(x, y))          = append(rev(y), x);
        repl2(nil)               = nil;
        repl2(cons(h, t))        = cons(h, cons(h, repl2(t)));
        repl4(l)                 = repl2(repl2(l));
        repl16(l)                = repl4(repl4(l));
        length(nil)              = 0;
        length(cons(x, y))       = add(length(y), 1);
        job(l)                   = length(rev(rev(repl16(l)))).
```

II.2. Input

```
job(cons(a,cons(b,cons(c,cons(d, cons(e, cons(f, cons(g, nil))))))))
```

II.3. Output

```
112
```

II.4. Prolog program

```
append(nil, L, L).
append(cons(X, L1), L2, cons(X,L3)) :- append(L1, L2, L3).

rev(nil, nil).
rev(cons(H,T), L) :- rev(T,Z), append(Z, cons(H, nil), L).

repl2(nil, nil).
repl2(cons(H,T1), cons(H, cons(H, T2))) :- repl2(T1, T2).

repl4(X, Y) :- repl2(X, Z), repl2(Z, Y).
repl16(X,Y) :- repl4(X,Z), repl4(Z,Y).

len(nil,0).
len(cons(H, T), N) :- len(T, M), N is M+1.

job(L, R) :- repl16(L, X), rev(X, Y), rev(Y, Z), len(Z, R).
```

II.5. Input

```
[prodef].
job(cons(a,cons(b,cons(c,cons(d,cons(e,cons(f,cons(g,nil))))))), N).
```

II.6. Output

Note: in all following Prolog output we have removed irrelevant system messages and have only retained essential information.

```
N = 112
```

Appendix III: ACK

III.1 Equational program

```
Symbols
        add:    2;
        subtract:        2;
        equ:    2;
        if:     3;
        ack:    2;
        include atomic_symbols;
        include truth_values;
        include integer_numerals.

For all m, n:

        include addint, subint, equint;

        if(true, m, n)  = m;
        if(false, m, n) = n;
        ack(m, n)       = if(equ(m, 0), add(n,1),
                             if(equ(n, 0),ack(subtract(m, 1), 1),
                                ack(subtract(m,1), ack(m, subtract(n,1))))).
```

III.2. Input

```
ack(3,2)
```

III.3. Output

```
29
```

III.4. Prolog program

```
ack(0, N, R) :- R is N+1.
ack(M, 0, R) :- M1 is M-1, ack(M1, 1, R).
ack(M, N, R) :- M1 is M-1, N1 is  N-1, ack(M, N1, R1), ack(M1, R1, R).
```

III.5. Input

```
[prodef].
ack(3, 2, R).
```

III.6. Output

```
R = 29
```

Appendix IV: ALPHA

Note: we only show the ALPHA example for the case N = 15.

IV.1 Equational program

```
Symbols
        char_0: 0;
        char_1: 0;
        char_2: 0;
        char_3: 0;
        char_4: 0;
        char_5: 0;
        char_6: 0;
        char_7: 0;
        char_8: 0;
        char_9: 0;
        char_10: 0;
        char_11: 0;
        char_12: 0;
        char_13: 0;
        char_14: 0;
        char_15: 0;
        eq_INTEGER: 2;
        eq_CHAR: 2;
        TRUE: 0;
        FALSE: 0;
        AND: 2;
        IF: 3;
        succ: 1;
        ord: 1;
        include integer_numerals;
        include atomic_symbols.
For all x, y, c1, c2:
AND(TRUE, TRUE) = TRUE;
AND(TRUE, FALSE) = FALSE;
AND(FALSE, TRUE) = FALSE;
AND(FALSE, FALSE) = FALSE;
IF(TRUE, x, y) = x;
IF(FALSE, x, y) = y;
eq_INTEGER(0, 0)          = TRUE;
eq_INTEGER(succ(x), succ(y)) = eq_INTEGER(x, y);
eq_INTEGER(0, succ(x)) = FALSE;
eq_INTEGER(succ(x), 0) = FALSE;
eq_CHAR(c1, c2) = eq_INTEGER(ord(c1), ord(c2));
ord(char_0) = 0;
ord(char_1) = succ(succ(succ(succ(succ(ord(char_0))))));
ord(char_2) = succ(succ(succ(succ(succ(ord(char_1))))));
ord(char_3) = succ(succ(succ(succ(succ(ord(char_2))))));
ord(char_4) = succ(succ(succ(succ(succ(ord(char_3))))));
ord(char_5) = succ(succ(succ(succ(succ(ord(char_4))))));
ord(char_6) = succ(succ(succ(succ(succ(ord(char_5))))));
ord(char_7) = succ(succ(succ(succ(succ(ord(char_6))))));
ord(char_8) = succ(succ(succ(succ(succ(ord(char_7))))));
ord(char_9) = succ(succ(succ(succ(succ(ord(char_8))))));
ord(char_10) = succ(succ(succ(succ(succ(ord(char_9))))));
ord(char_11) = succ(succ(succ(succ(succ(ord(char_10))))));
ord(char_12) = succ(succ(succ(succ(succ(ord(char_11))))));
ord(char_13) = succ(succ(succ(succ(succ(ord(char_12))))));
ord(char_14) = succ(succ(succ(succ(succ(ord(char_13))))));
ord(char_15) = succ(succ(succ(succ(succ(ord(char_14)))))).
```

IV.2. Input

```
IF(AND(eq_CHAR(char_0, char_0),
AND(eq_CHAR(char_1, char_1),
AND(eq_CHAR(char_2, char_2),
AND(eq_CHAR(char_3, char_3),
AND(eq_CHAR(char_4, char_4),
AND(eq_CHAR(char_5, char_5),
AND(eq_CHAR(char_6, char_6),
AND(eq_CHAR(char_7, char_7),
AND(eq_CHAR(char_8, char_8),
AND(eq_CHAR(char_9, char_9),
AND(eq_CHAR(char_10, char_10),
AND(eq_CHAR(char_11, char_11),
AND(eq_CHAR(char_12, char_12),
AND(eq_CHAR(char_13, char_13),
AND(eq_CHAR(char_14, char_14),
    eq_CHAR(char_15, char_15)))))))))))))))),
char_15, FALSE)
```

IV.3. Output

```
char_15
```

IV.4. Prolog program

```
and(true, true, true).
and(true, false, false).
and(false, true, false).
and(false, false, false).
if(true, X, Y, X).
if(false, X, Y, Y).
eq_INTEGER(0, 0, true).
eq_INTEGER(succ(X), succ(Y), R) :- eq_INTEGER(X, Y, R).
eq_INTEGER(0, succ(X), false).
eq_INTEGER(succ(X), 0, false).
eq_CHAR(C1, C2, R) :- ord(C1, N1), ord(C2, N2), eq_INTEGER(N1, N2, R).
ord(char_0, 0).
ord(char_1, succ(succ(succ(succ(succ(R)))))) :- ord(char_0, R).
ord(char_2, succ(succ(succ(succ(succ(R)))))) :- ord(char_1, R).
ord(char_3, succ(succ(succ(succ(succ(R)))))) :- ord(char_2, R).
ord(char_4, succ(succ(succ(succ(succ(R)))))) :- ord(char_3, R).
ord(char_5, succ(succ(succ(succ(succ(R)))))) :- ord(char_4, R).
ord(char_6, succ(succ(succ(succ(succ(R)))))) :- ord(char_5, R).
ord(char_7, succ(succ(succ(succ(succ(R)))))) :- ord(char_6, R).
ord(char_8, succ(succ(succ(succ(succ(R)))))) :- ord(char_7, R).
ord(char_9, succ(succ(succ(succ(succ(R)))))) :- ord(char_8, R).
ord(char_10, succ(succ(succ(succ(succ(R)))))) :- ord(char_9, R).
ord(char_11, succ(succ(succ(succ(succ(R)))))) :- ord(char_10, R).
ord(char_12, succ(succ(succ(succ(succ(R)))))) :- ord(char_11, R).
ord(char_13, succ(succ(succ(succ(succ(R)))))) :- ord(char_12, R).
ord(char_14, succ(succ(succ(succ(succ(R)))))) :- ord(char_13, R).
ord(char_15, succ(succ(succ(succ(succ(R)))))) :- ord(char_14, R).
job(T) :- eq_CHAR(char_0, char_0, R0),
and(R0, R0, T0),
eq_CHAR(char_1, char_1, R1),
and(T0, R1, T1),
eq_CHAR(char_2, char_2, R2),
and(T1, R2, T2),
eq_CHAR(char_3, char_3, R3),
and(T2, R3, T3),
eq_CHAR(char_4, char_4, R4),
and(T3, R4, T4),
eq_CHAR(char_5, char_5, R5),
and(T4, R5, T5),
eq_CHAR(char_6, char_6, R6),
and(T5, R6, T6),
eq_CHAR(char_7, char_7, R7),
and(T6, R7, T7),
eq_CHAR(char_8, char_8, R8),
and(T7, R8, T8),
eq_CHAR(char_9, char_9, R9),
and(T8, R9, T9),
eq_CHAR(char_10, char_10, R10),
and(T9, R10, T10),
eq_CHAR(char_11, char_11, R11),
and(T10, R11, T11),
eq_CHAR(char_12, char_12, R12),
and(T11, R12, T12),
eq_CHAR(char_13, char_13, R13),
and(T12, R13, T13),
eq_CHAR(char_14, char_14, R14),
and(T13, R14, T14),
eq_CHAR(char_15, char_15, R15),
and(T14, R15, T15),
if(T15, char_15, false, T).
```

IV.5. Input

```
[prodef].
job(T).
```

IV.6. Output

```
T = char_15
```