A Register-free Abstract Prolog Machine with Jumbo Instructions

Neng-Fa Zhou

CUNY Brooklyn College & Graduate Center

1 Introduction

The majority of current Prolog systems are based on the WAM, in which registers are used to pass procedure arguments and store temporary data. In this paper, we present a stack machine for Prolog, named *TOAM Jr.*, which departs from the TOAM adopted in early versions of B-Prolog in that it employs no registers for temporary data and it offers variable-size instructions for encoding unification and procedure calls. TOAM Jr. is suitable for fast bytecode interpretation: the omission of registers facilitates instruction merging and the use of jumbo instructions results in more compact code and execution of fewer instructions than the use of fine-grained instructions. TOAM Jr. is employed in B-Prolog version 7.0. Benchmarking shows that TOAM Jr. helps significantly improve the performance: the execution speed is increased by 48% on a Windows PC and 77% on a Linux machine. Despite the overhead on standard Prolog programs caused by the adoption of a spaghetti stack to support event handling and constraint solving, B-Prolog version 7.0 compares favorably well with the state-of-the-art WAM-based Prolog systems.

2 The TOAM Jr. Instruction Set

TOAM Jr. inherites the memory architecture from the TOAM. There is a frame for each procedure call, which stores arguments, machine status registers, and local variables. A different set of machine status registers needs to be saved for each different procedure type, and hence frames for different types of procedures have different structures.

Instructions are classified into the following categories: *control* (`allocate`, `return`, `fork`, `cut`, and `fail`), *branch* (`jmpn_constant`, `switch_on_cons`, `jmpn_struct`, and `hash`), *move* (`move_list` and `move_struct`), *unify* (`unify_constant`, `unify_value`, `unify_list`, and `unify_struct`) and *call* (`call` and `last_call`). The following shows an example. An operand in the form of $y(Loc)$ denotes a frame slot where a positive offset refers to an argument and a negative offset refers to a local variable. A tagged operand can be an uninitialized frame slot $v(i)$, an initialized frame slot $u(i)$, or a constant $c(a)$. A singleton variable is denoted as $v(0)$. The binary literal '0b11101' is the layout bit vector for the last call, which indicates that all the arguments except for the second one (`Y`) are misplaced and need to be rearranged if the current frame is reused.

```
% p(X,Y,Z):-S=f(X,Y),q(S),r(Z,Y,X,W,9).
p/3: allocate_det(3,5)                 % arity=3; frame size=5
     move_struct(y(-1),f/2,u(3),u(2))  % S=f(X,Y)
     call(q/1,u(-1))                   % q(S)
     last_call(0b11101,r/5,u(1),u(2),u(3),v(0),c(9))
```

TOAM Jr. offers specialized instructions that carry the numbers and types of operands in their opcodes, and also merged instructions each of which combines two or more base instructions.

3 Experimental Results

Table 1 compares B-Prolog version 7.0 (BP7.0) with version 6.9 (BP6.9) and three fast WAM implementations (Yap 5.1.1, SICStus 4.0, and hProlog 2.7.14) on CPU time on a Linux machine (3.8GHz CPU and 2G RAM). B-Prolog outperforms SICStus and Yap but is about 10% slower than hProlog. Benchmarking on a Windows XP machine reveals that BP7.0 is on average 48% faster than BP6.9, 12% faster than SICStus, and 27% faster than Yap. No Windows version of hProlog was available.

The speedup of BP7.0 over BP6.9 is mainly attributed to the use of specialized jumbo instructions, which would be more difficult if registers were existent. Other factors affect the performance too. In B-Prolog, the top bit of a word is reserved for tagging and, because of this, pointers have to be repaired after being untagged on Linux machines. Repairing pointers imposes about 5% speed overhead. hProlog uses the lowest three bits for tags and therefore requires no repairing of pointers. The B-Prolog and hProlog compilers are able to detect the determinacy of a key predicate in `tak`. This is probably the main reason why they run faster on `tak` than the other two systems. B-Prolog has suspension frames stored on the stack (so called spaghetti stack), which facilitates context switching for action rules but incurs certain overhead on Prolog programs: even for a predicate that is made up of only one fact, the return instruction needs to check if the current frame is reusable since any predicate can be interrupted.

Table 1. Comparison on CPU times.

program	BP7.0	BP6.9	Yap	Sics	hProlog
boyer	1	1.69	1.28	1.87	1.04
browse	1	1.72	1.45	1.55	0.74
chat_parser	1	1.56	1.36	1.82	1.13
crypt	1	1.56	1.65	1.94	0.92
meta_qsort	1	1.54	1.21	1.22	0.70
nreverse	1	3.31	1.17	1.03	0.62
poly_10	1	1.63	1.81	1.59	0.87
queens_8	1	1.68	1.29	1.60	0.75
reducer	1	1.62	1.43	1.73	0.98
sendmore	1	1.96	1.81	2.33	1.13
tak	1	1.70	3.25	2.42	1.09
zebra	1	1.30	0.83	1.07	0.91
mean	1	1.77	1.55	1.68	0.91