A Novel Term Compression Scheme and Data Representation in the BinWAM

Paul Tarau\textsuperscript{1} and Ulrich Neumerkel\textsuperscript{2}

\textsuperscript{1} Universit\textsuperscript{e} de Moncton
D\textsuperscript{e}partement d'Informatique
Moncton, N.B. Canada, E1A 3E9,
tarau@info.umoncton.ca
\textsuperscript{2} Technische Universit\textsuperscript{a}t Wien
Institut f\textsuperscript{u}r Computersprachen
A-1040 Wien, Austria
ulrich@mips.complang.tuwien.ac.at

Abstract. In this paper we present the novel term representation of the BinWAM (a simplified WAM engine for executing binary logic programs) and evaluate its impact in BinProlog, a C-emulated system\textsuperscript{3} based on the BinWAM and on the mapping of logic programs to binary Prolog introduced in [13]. Terms in the BinWAM are compressed with a new technique called last argument overlapping which takes advantage of an unconventional untagged pointer representation, called tag-on-data. A Cheney-style copy\_term algorithm using these term representations is described for BinProlog's fast copy once implementation of findall. While BinProlog's performance is competitive with the best commercial Prolog systems, its implementation is significantly simpler. Our analysis shows that this term representation and a limited amount of instruction folding on top of a reduced basic instruction set make the BinWAM a realistic alternative to its more complex forerunner.

\textbf{Keywords:} implementation of Prolog, WAM, term representation, last argument overlapping, continuation passing style.

1 Introduction

BinProlog is a C-emulated Prolog engine for a subset of Prolog [13] that uses a simplified WAM [16] called BinWAM [9]. A transformation called binarization [13] maps full Prolog to binary logic programs. In this manner the BinWAM is used to implement full Prolog.

Our primary motivation was to investigate whether specializing the WAM to binary programs yields new optimization opportunities that compensate for the extra heap consumption. Because WAM's highly optimized environments were traded for a heap-only run-time system, a slow-down was expected, although it was clear from the start that, at least, the implementation will be much simpler. Our experience with BinProlog shows that its execution speed is competitive with highly optimized implementations of the WAM. This paper

\textsuperscript{3} Available by anonymous ftp from clement.info.umoncton.ca.
compares WAM and BinWAM and presents a novel term-compression technique called last argument overlapping.

Contents. In Section 2 we present binary Prolog a subset of full Prolog used as the intermediate language for the Prolog-engine. Section 3 presents our simplified WAM, called BinWAM, and discusses the impacts of its simplified design. The term representation of the BinWAM is compared with traditional structure copying representations. Section 4 compares the actual implementation, BinProlog, to existing systems.

2 Binary Prolog

Binary Prolog is a subset of Prolog based on binary definite programs of a very simple form: one atom in the head and one atom in the body. This subset, enhanced by a labeled cut and some in-line built-ins [5], executes on a WAM engine without using WAM’s environments.

Binarization: from full Prolog to binary Prolog. The natural framework of binarization covers the transformation from definite metaprograms, i.e., programs with metavariables in atom positions to binary definite programs, i.e., programs with atomic head and body. We refer to [13] for a formal definition and a study of certain semantic properties of this transformation. We give here only a few examples:

<table>
<thead>
<tr>
<th>Source clause:</th>
<th>Binary clause:</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(X) ←</td>
<td>p(X,Cont) ← q(X, r(X, Y, s(Y, Cont))).</td>
</tr>
<tr>
<td>q(X), r(X,Y), s(Y).</td>
<td></td>
</tr>
<tr>
<td>append([],Ys,Ys).</td>
<td>append([],Ys,Ys,Cont) ← true(Cont).</td>
</tr>
<tr>
<td>and(X,Y) ←</td>
<td>and(X,Y,Cont) ← call(X, call(Y,Cont)).</td>
</tr>
<tr>
<td>X, Y</td>
<td></td>
</tr>
</tbody>
</table>

Note that the goal true(Cont) executes the continuation Cont.

3 The BinWAM

Since binary Prolog is a subset of full Prolog every Prolog engine can be used to execute binary Prolog, although less efficiently than the BinWAM. We discuss in this section the simplified design of the BinWAM with respect to the full WAM.