Funding Proposal

Compiling Idiomatic Prolog to Idiomatic C#

Jonathan Cook and Stephen Gilmore

Laboratory for Foundations of Computer Science
The University of Edinburgh
September 2002

1 Introduction

This document is a proposal for funding to support further research into an existing Prolog implementation developed for the .NET platform in the Laboratory for Foundations of Computer Science at The University of Edinburgh. The University of Edinburgh is an internationally leading centre for Informatics. The work of the LFCS has been rated excellent in both teaching and research (5*A).

We have developed a new Prolog implementation for .NET, called P# [7], which is implemented as a set of C# classes. P# operates by compiling Prolog source code into C# source code. It is based on the LLP/Prolog to Java translator Prolog Café [8]. New built-in predicates were added to enable the user to exploit the concurrency support in C# from the Prolog side. We now seek funding for one year of work on optimising the translated code by detecting common programming idioms and by allowing the programmer to add mode annotations to their code.

Translating Prolog to C# source code provides one way of using Prolog within the .NET Framework. By translating to MSIL via C# we can exploit the C# compiler's ability to produce well optimised MSIL. It is also possible to exploit language features of C#, in particular its rich graphical, networking and other libraries, by building equivalent features in Prolog.

Papers describing work on P#: The adaptation of Prolog Café to produce C# instead is described in the paper [2]. The addition of predicates for forking Prolog threads and for transmitting data between them is described in the paper [3]. Related work in adding concurrency support to Prolog can be found in [1].

Examples of the use of P#: As an example of the use of P# we have implemented a game of 'noughts and crosses', with a Web Application front-end and a Prolog back-end, which allows users to play against the computer.

We have also implemented a system of disconnected collaborative agents allowing a number of users to asynchronously assert facts to a database, then to disconnect and alter their private copies of the database. On reconnection their private copies are synchronised with the central database by searching for conflicts between their copies and the central database.

Relation to other Prolog implementations: A number of other Prolog implementations are available under Windows. For example, Jinni 2002 [5] is described as "a high performance Java and .NET based Object Oriented Prolog compiler". P# has the advantage over Jinni 2002 that it depends only on C# and not Java. Visual Prolog [9] is a traditional Prolog for Windows, and unlike P# is not .NET or Windows XP friendly.

2 Intended Work

2.1 Generating More Idiomatic C#

The current compilation scheme leaves open the possibility of compiling a predicate, and all predicates deeper than it in the tree, into more idiomatic C#. Thus, if we could detect instances where this would be possible, it may be the case that much more efficient C# could be generated.

In particular tail recursive predicates which involve no cuts could be compiled into `while` loops. For example consider the usual Prolog code for finding the length of a list:

```
len( [], Z, Z ).
len( [_|T], A, Z ) :-
    A1 is A + 1,
    len( T, A1, Z ).

len( List, Length ) :-
    len( List, 0, Length ).
```

This could be compiled into:

```
a = 0;
while( !list.isEmpty( ) ) {
    list = list.tail( );
    a++;
}
return a;
```

A number of stages would be involved in such a compilation, for example mode inference to determine that `Length` is an output parameter of `len/2`

and a liveness analysis to determine that `a` should be incremented by one in each iteration of the loop.

In fact much tail recursive code can be characterised as conforming to the following general pattern:

```
p( ..., Z, Z ). % base 1
p( ..., Z, Z ). % base 2
...             % base i

p( ..., A, Z ) :- ..., p( ..., A1, Z ). % step 1
p( ..., A, Z ) :- ..., p( ..., A1, Z ). % step 2
...                                     % step j
```

It would be possible, though maybe quite involved, to detect code which looks like this and then to translate it into iterative code. However, it is not clear that the code would run significantly faster since much of the translated code is the same as it was before. In the list length example above, we are still calling the method which returns a list's tail. Nevertheless making the generated code more idiomatic should ensure that we are working with the C# compiler's optimiser rather than against it.

2.2 Mode Inference

Related projects, such as Mercury [6] and HAL [4], support mode declarations. A mode declaration provides extra information to the compiler regarding which arguments of a predicate are input arguments and which are output arguments. This information can be optional, only being used to provide more efficient compiled code when some mode declarations have been provided.

Notwithstanding backtracking, as the program runs forwards variables change from being uninstantiated to instantiated. Together with any mode data, we can then infer some of the modes which have not been provided.

Even if only the external interfaces to the Prolog program are given mode declarations, a large number of modes could be inferred.

3 Relevance to .NET

The .NET platform offers an unparallelled opportunity to build systems based on a number of interoperating languages. Logic programming is currently underrepresented on the .NET platform, with Mercury a notable exception. Prolog is the seminal logic programming language, and P# is an efficient and reliable Prolog implementation for .NET useful in particular to programmers who wish to use legacy Prolog code which cannot be easily ported to Mercury.

References

[1] Ciancarini, P., (1992) Parallel Programming with Logic Languages: A Survey, In Journal Computer Languages 17(4) 213–239.

[2] Cook, J. J., (2002) P#: Using Prolog within the .NET Framework. University of Edinburgh Technical Report EDI-INF-RR-0145. Available from `http://www.dcs.ed.ac.uk/home/jjc`.

[3] Cook, J. J., (2002) A Concurrent Prolog for Interoperation with C#. Submitted for publication. Available from `http://www.dcs.ed.ac.uk/home/jjc`.

[4] The HAL home page: `http://www.csse.monash.edu.au/~mbanda/hal/`

[5] Jinni 2002 home page: `http://www.binnetcorp.com/binnet.html`

[6] The Mercury home page: `http://www.cs.mu.oz.au/research/mercury/`

[7] P# home page: `http://www.dcs.ed.ac.uk/home/jjc/psharp/dlpsharp.html`

[8] Prolog Café home page: `http://pascal.cs.kobe-u.ac.jp/~banbara/PrologCafe/index-jp.html`

[9] Visual Prolog home page: `http://www.visual-prolog.com/`

Contact Details

Jonathan Cook
Laboratory for Foundations of Computer Science
The University of Edinburgh
Edinburgh EH9 3JZ
Scotland
Phone: +44 (0)131 650 7354
Email: `Jon.Cook@ed.ac.uk`

Stephen Gilmore
Laboratory for Foundations of Computer Science
The University of Edinburgh
Edinburgh EH9 3JZ
Scotland
Phone: +44 (0)131 650 5189
Email: `Stephen.Gilmore@ed.ac.uk`