P#: Translating Prolog to C#

Jon Cook

Abstract: We modify the Prolog to Java translator Prolog Cafe, so that it translates Prolog to C# source code instead of Java. We also add concurrency primitives on the Prolog side, so that C#'s built-in support for concurrency can be exploited.

Introduction

Translating Prolog to a high-level language, as opposed to native code, has many advantages. One of the most important must be that we gain the ability to interoperate with the target language.

P# translates Prolog source code to C# source code. P# was obtained by modifying the tool Prolog Cafe, which translates Prolog (or rather a linear logic extension of Prolog called LLP) to Java. P# inherits from Prolog Cafe the ability to write programs involving linear logic resources.

Prolog Cafe consists of a run-time system written in Java and a translator written in Prolog, which can compile itself into Java. We ported the run-time system to C# and modified the translator to take account of the differences in syntax between Java and C#.

Having done this we added some concurrency support to P#, by adding new built-in predicates.

`fork/1` and `wait_for/1`

The `fork/1` primitive takes as an argument a structure. A new thread is forked, then the structure is cloned onto the new Prolog engine's stacks, and executed. The thread terminates when this evaluation either succeeds or fails.

The `wait_for/1` primitive takes as an argument an uninstantiated variable which occurred in a structure passed to a fork. The call to `wait_for` blocks until a copy of that variable on another thread becomes instantiated. When this happens, the instantiation is cloned onto the thread calling `wait_for`, and the variable is unified with it. Thus, `wait_for` succeeds having made an instantiation.

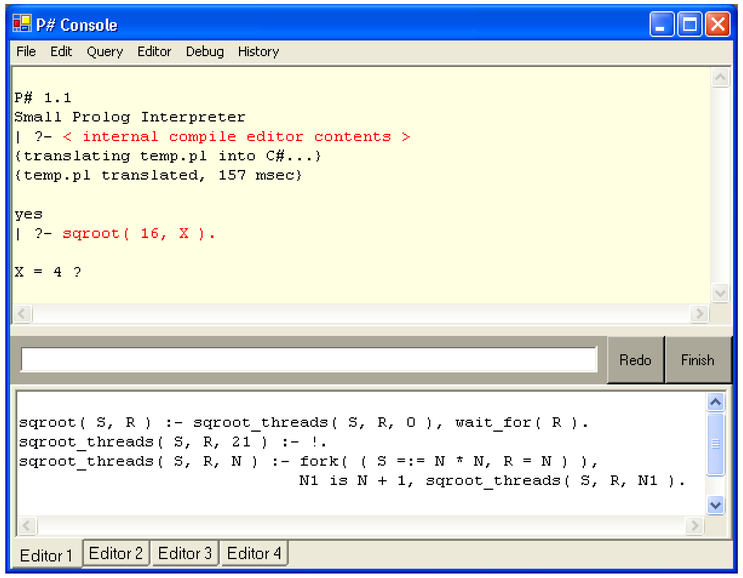
A Global Database

Each forked thread is equipped with a private database which it can use in the normal way. In addition we provide a global database, which is shared between all the threads. Accesses are automatically protected by a mutex. The database can be modified by primitives `global_assert/1`, `global_retract/1` and so on. To query the database a `global_call/1` predicate is provided. For example:

```
assert_test( Y ) :-
  fork( assert_test_thread( X ) ),
  wait_for( X ),
  global_call( info( Y ) ).

assert_test_thread( X ) :-
  global_assert( info( [1, 2, 3] ) ),
  X = []. % arbitrary instantiation
```

In this case, we could have just passed back the information in the variable `X`.



Screen-shot

Searching For a Square Root

The screen-shot shows a graphical interface to P#. The editor at the bottom contains a small P# program which calculates the square root of a square integer between 0 and 400. The program forks 21 threads, each of which tries one of the possible square roots. When one of them finds the root, this is reported back to the main thread. Because a conjunction of two goals is a structure with functor `,/2`, we can place such a conjunction in a call to fork. Extra brackets are needed to make clear that this conjunction is a single argument.

The following piece of code is the C# class which corresponds to the second clause of `sqroot_threads/3`. Some necessary symbols are statically generated. The clause is executed by calling the `exec()` method. This first constructs the needed terms and then adds to the stack of calls to be made, a call to `fork/1` followed by a call to `+/3` and then a tail recursive call to `sqroot_threads/3`. P# attempts to convert idiomatic Prolog names to idiomatic C# names.

```
sealed class SqrootThreads_3_2 : SqrootThreads_3 {

    static internal readonly SymbolTerm f1
      = SymbolTerm.makeSymbol(",", 2);
    static internal readonly SymbolTerm f2
      = SymbolTerm.makeSymbol(":=", 2);
    static internal readonly SymbolTerm f3
      = SymbolTerm.makeSymbol("*", 2);
    static internal readonly SymbolTerm f6
      = SymbolTerm.makeSymbol("=", 2);
    static internal readonly IntegerTerm s9
      = new IntegerTerm(1);

    public override Predicate exec( Prolog engine ) {
        Term a1, a2, a3, a4, a5, a6, a7, a8;
        Predicate p1, p2;
        a1 = engine.aregs[1].dereference();
        a2 = engine.aregs[2].dereference();
        a3 = engine.aregs[3].dereference();
        Predicate cont = engine.cont;

        Term[] h4 = {a3, a3};
        a6 = new StructureTerm(f3, h4);
        Term[] h5 = {a1, a6};
        a5 = new StructureTerm(f2, h5);
        Term[] h7 = {a2, a3};
        a7 = new StructureTerm(f6, h7);
        Term[] h8 = {a5, a7};
        a4 = new StructureTerm(f1, h8);
        a8 = engine.makeVariable();

        p1 = new Predicates.SqrootThreads_3(a1, a2, a8, cont);
        p2 = new Predicates.dollar_plus_3(a3, s9, a8, p1);
        return new Predicates.Fork_1(a4, p2);
    }
}
```

Further Information

The reader may wish to obtain P# which is available from

`http://www.dcs.ed.ac.uk/home/jjc/`

Division of informatics