P#: A Concurrent Prolog for the .NET Framework

Jon Cook

Overview

- P# and .NET
- Architecture
- Concurrency
- Producer and Consumer Example
- Disconnected Collaborative Agents Example
- Object-Oriented Assistant Example
- Future Work

Introduction

.NET is Microsoft's new multiple language platform. All supported languages are compiled down to an intermediate language called MSIL.

(MSIL is similar to Java bytecode, although Java bytecode was designed with only Java in mind.)

Yet, the only support for logic programming in the .NET framework is a functional logic language called Mercury.

It can be difficult to translate existing Prolog programs to Mercury.

P#

Prolog is the seminal logic programming language.

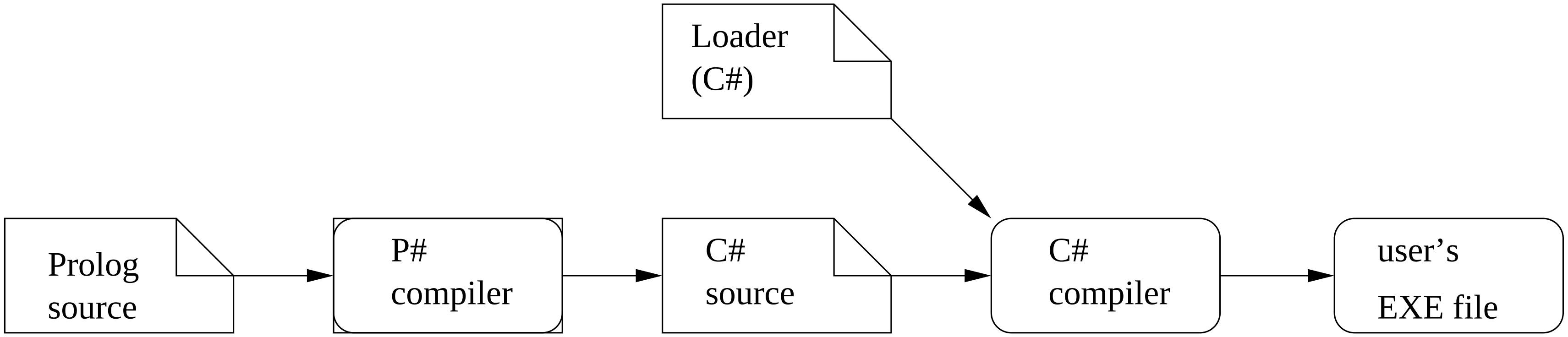
C# is the principal .NET language.

P# translates Prolog to C# source code. This allows applications to be developed which have

- Prolog back-ends (for search problems etc.)
- C# front-ends (for graphics and networking etc.)

The resultant C# code is compiled together with the P# runtime system to produce a Windows Application or Web Service.

How the User Generates an Executable File



The Loader class calls the P# runtime system, which then runs the user's program's main predicate.

P# and Prolog Café

Prolog Café is a translator from Prolog/LLP to Java. P# is a modified port of Prolog Café. Thus, P# also supports linear logic programming.

Prolog Café is bootstrapped. A bootstrapped P# was obtained from Prolog Café by a two stage bootstrapping.

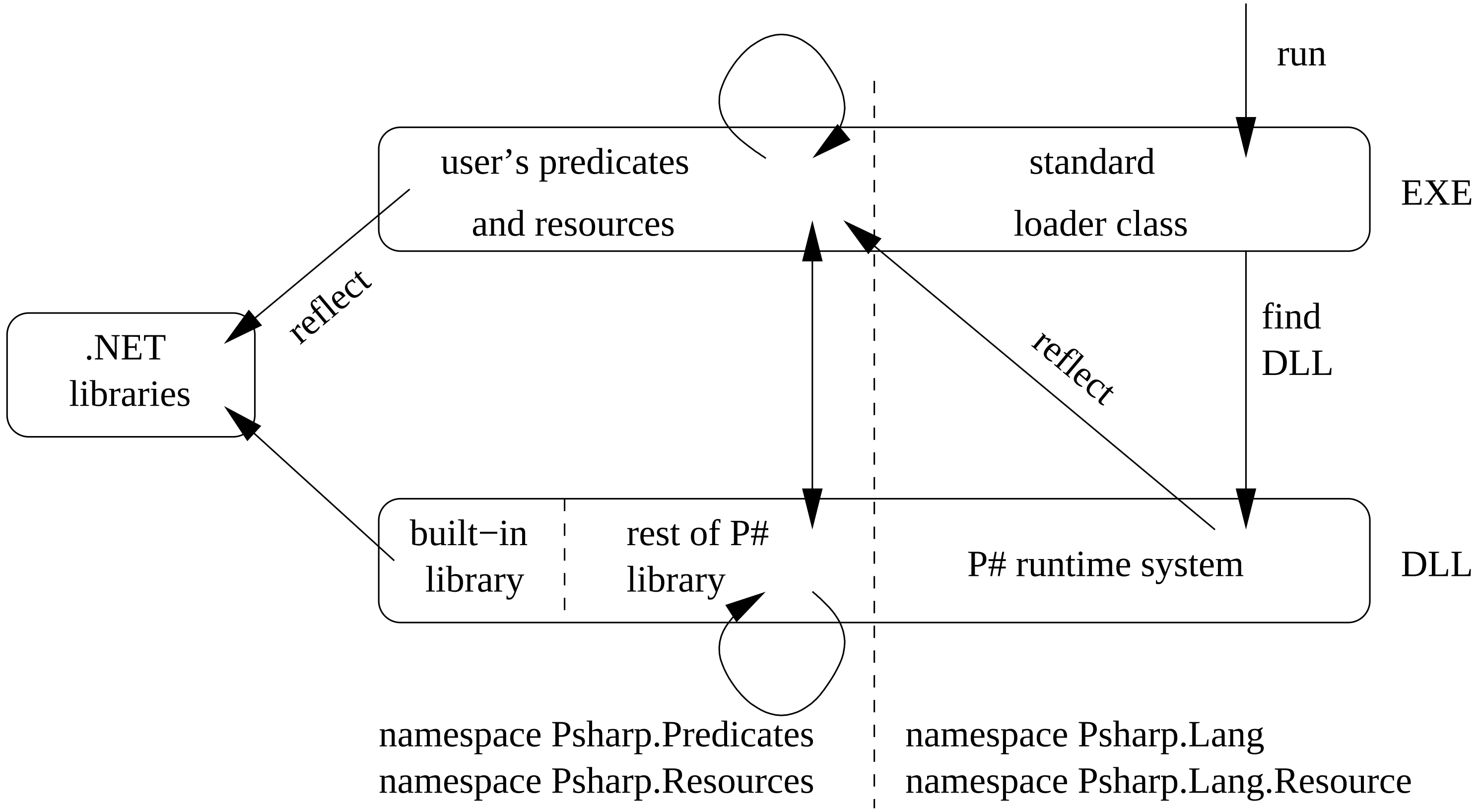
Porting the runtime system (written in Java) to C# and modifying the translator (written in Prolog/LLP) was straightforward because of the close similarity between the language and libraries of Java and C#.

```
public final class A extends B
```

becomes

```
public sealed class A : B
```

Architecture



The `fork` and `wait_for` Predicates

The `fork/1` predicate evaluates its argument on a new thread.

Having called `fork` with a structure containing an uninstantiated variable, anywhere in its tree, a thread can use that variable to interact with the newly forked thread.

The `wait_for/1` predicate waits for a variable passed through a `fork` to become instantiated, and then succeeds with that instantiation. We call such variables *concurrent variables*.

If for a given concurrent variable, all the threads which possess a copy of that variable are waiting for that variable, then all of those calls to `wait_for` fail.

This occurs, for example, if all producers terminate having produced no results; leaving the consumer waiting for results that will never appear.

Example 1

For example, consider the program:

```
a( 2, 7 ).

and( Y ) :- fork( a(1, Y) ), fork( a(2, Y) ), fork( a(3, Y ) ),
            wait_for( Y ).
```

A call to `and( Y )` succeeds with `Y = 7`.

A call to `and( 5 )` fails, since all the forked threads terminate without making a binding, and then it is detected that the only thread with a copy of `Y` which is still running is waiting for `Y`.

Example 2: Producer and Consumer

```
main :-
  fork( prod( X ) ),
  cons( X ).

cons( X ) :-
  wait_for( X ),
  X2 is X * 2,
  write( X2 ), nl,
  fail.
cons( _ ).
```

```
prod( X ) :-
  enum( X, 0 ).

enum( _, 11 ) :- !.
enum( X, Y ) :-
  pulse( X, N ),
  N1 is N + 1,
  enum( X, N1 ).

pulse( X, N ) :-
  X = N, fail.
pulse( _, _ ).
```

A Global Table

In Prolog, facts can be asserted to and retracted from tables. This allows imperative code to be written.

In P#, each thread has a local table which it can use in the normal way, free from interference by other threads.

In addition a global table is provided which is shared between all of the threads. Facts can be asserted (`global_assert` predicate), retracted and called (`global_call` predicate).

Message passing is implemented using the global table. The queue of messages waiting to be consumed is stored on the global table.

A unification asserts a fact on the global table and a successful `wait_for` retracts the fact.

Example 3: Disconnected Collaborative Agents

A central database stores a set of facts. Individual users can connect to the database and modify the facts. They can also modify their own copy when disconnected. On reconnecting conflicts are detected and removed.

We use 'key fields' to provide a notion of conflict. One or more of the arguments can be marked as such. Then there cannot be two facts with the same key fields but differing other fields.

For example suppose we have `person( key, nonkey, nonkey ).` Then `person( 24182, 'Peter', 18 )` and `person( 24182, 'John', 18 )` are conflicting facts. The user would be asked which one should be retracted.

Example 3: Implementation of Agents Example

The conflict detection works as follows:

For each predicate we find whether there are any conflicts.

- If there are not, we move onto the next predicate.
- If there are, we fork a thread which asks the user which of the conflicting facts to retain and move onto the next predicate.

We create a variable and pass it to all of these forked threads. Having checked all of the predicates we wait for this variable. It will never be instantiated, but the `wait_for` will wait until all of the forked threads have terminated.

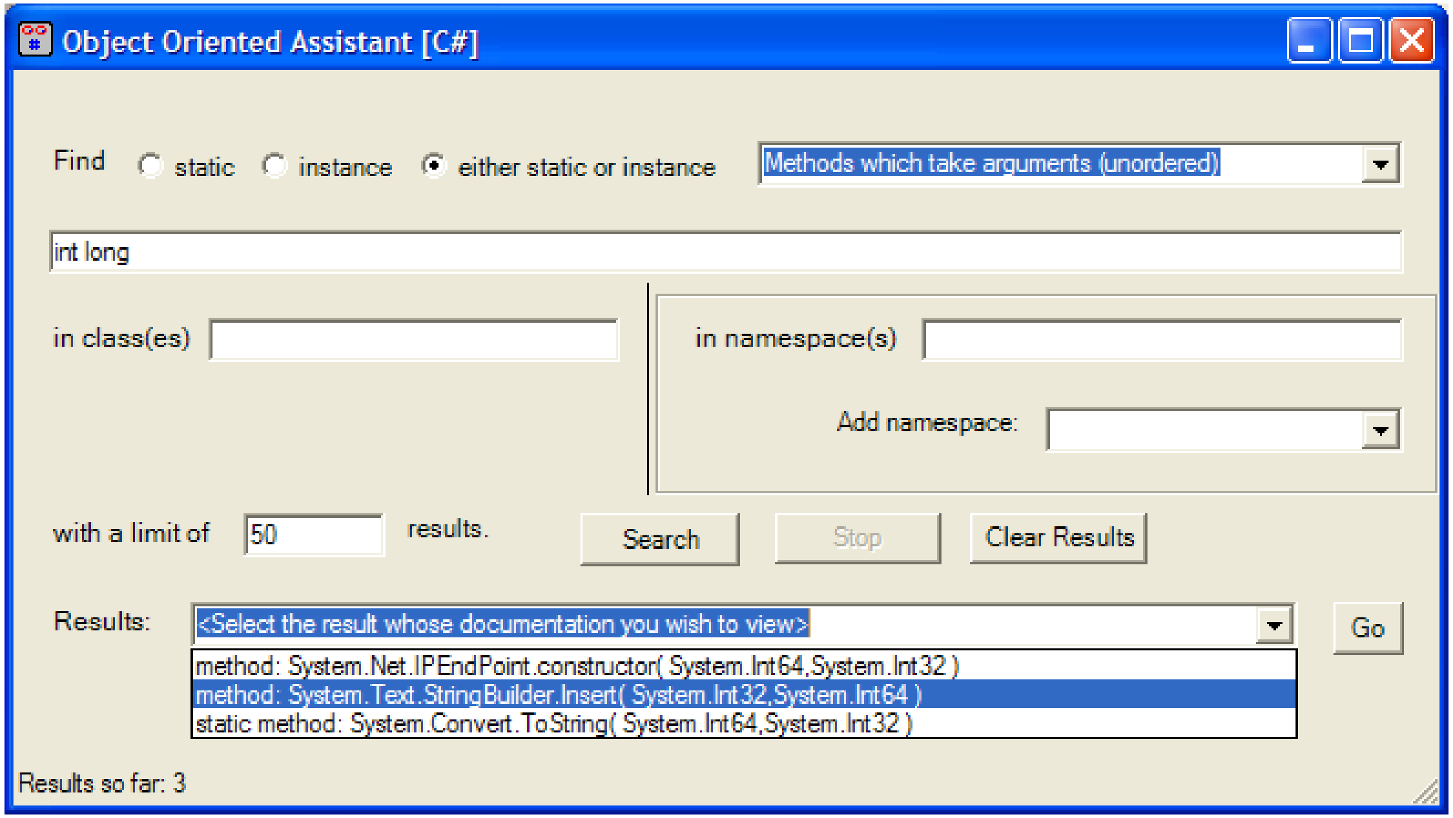
Example 4: An Object-Oriented Assistant

Faced with a new Java package or C# namespace, a programmer needs to discover how the classes interoperate with one another.

We developed a tool which allows the user to query the Java and C# libraries.

- Each package/namespace is compiled into a set of Prolog facts (using reflection).
- Then this flat database is converted into a tree structured database to avoid deep recursion.
- This database is compiled into C# by P#.
- Then, this C# code is compiled into a DLL by the C# compiler.

Example 4: Object-Oriented Assistant



Performance

Comparison of P# with Prolog Café (under Sun's JVM for Windows).

On average P# produces code which is

- 1.5 times smaller
- 1.8 times faster.

This difference is largely due to differences between Java and C#.

Future Work

Translating Prolog to more idiomatic C#, for example:

```
len( [], Z, Z ).
len( [_|T], A, Z ) :- A1 is A + 1, len( T, A1, Z ).
length( List, Length ) :- len( List, 0, Length ).
```

could be translated to:

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

Conclusion

Prolog is suited to applications requiring logical deduction from a set of facts.

Nowadays such a program usually requires graphical, networking and multithreading capabilities.

With P# this can be easily achieved.

P# can be obtained from

`http://www.lfcs.ed.ac.uk/psharp`