THE SWAM USER MANUAL IN A NUTSHELL.

Pascual Julián-Iranzo and Clemente Rubio-Manzano

Department of Information Technologies and Systems,
University of Castilla-La Mancha.
{Pascual.Julian},{Clemente.Rubio}@uclm.es

1 Introduction

Bousi~Prolog (BPL for short) is an extension of the Prolog language with similarity relations. The implementation we are describing is based on (a subset of) the Warren Abstract Machine plus some new techniques incorporated inside it to deal with flexible querying. The result is what we call the Similarity Warren Abstract Machine (SWAM). The foundations and some implementation details of the SWAM can be found in [1].

At the present time the SWAM is a prototype implementation and some bugs may arise during its execution. Please, send your comments to: Clemente.Rubio@alu.uclm.es

2 Requirements and Instalation Procedure

You need a computer running Windows 2000 or XP operating system and the Java Virtual Machine version 1.5 (JVM 1.5) installed on it.

In order to install and execute the BPL environment you must follow these simple steps:

1. Checks if the JVM 1.5 is installed. If you need to install JVM 1.5, go to the URL address: http://java.sun.com/javase/downloads/index_jdk5.jsp
2. Download the file "bousi.jar" into the BPL home directory.
3. Simply double click on the "bousi.jar" file or, if you have installed the Java SE Development Kit (JDK), follow these steps:
   - Located in the BPL home directory, go to the Windows option and select "Init\Execute" in the tool bar and write "cmd". Then, a system console is opened.
   - Write "java jar bousi.jar" or

     C:\<BPL home directory>\java jar bousi.jar

     into the command line.

   Then the BPL environment is available.

3 The BPL Environment

In this section we give a brief tutorial in how to use the BPL environment wich is the interface between the user and the SWAM.

The BPL Environment is divided in two different zones: The command options zone (with the menu bar and the icon bar at the top of the screen) and the windows zone. The windows zone involves four kinds of windows:

- The **query window**, placed at the left-bottom of the screen, serves to introduce a query to the system.
- The **out window** shows the answers to a query an other system information. You can delete the information in the out window, by means of the command `clear` written into the query window.
- The **code area window** shows the SWAM machine code obtained after the compilation of the source program. It is the object program executed by the abstract machine. You can delete the information in the code area window, by typing the command `reset` into the query window
- The **visualization window** shows a pictorial representation of the Similarity Matrix, that is, an adjacency matrix representation of the reflexive, symmetric, transitive closure of the original fuzzy binary relation (which is computed starting from the set of similarity equations provided in the program).

As we shall comment is also possible to open several **edit windows** to create or modify programs.

3.1 Creating new programs

To create a new program, you need to open a new edit window. You select the `new` option on the `File` menu. This may be done either by pulling down the mouse and selecting `new` or by typing the keys "`ctrl-N`" or by clicking the blank paper icon. Once the new edit window has been created you may introduce your program.

A *edit window* is a basic full editor (with the usual options). The edit window does not send any input to the BPL system. It only contains your program. You can save your program into a file, placed into the directory where the file `bousi.wam` has been installed, by pulling down the mouse and selecting `save` or by typing the keys "`ctrl-G`" or by clicking the `diskette` icon. Also, you can save your program into a specific directory either by pulling down the mouse and selecting `save as` or by typing the keys "`ctrl-M`" or by clicking the `star diskette` icon.

You may have more than one edit windows open.

WARNING: It is important that the name of the file containing a BPL program has the suffix ".bpl".

3.2 Editing a program

Once a program has been stored on a disk, you can open it in an edit window either by using the `open` option in the File Menu or by typing the keys "`ctrl-A`" or by clicking the `opening archive` icon.

3.3 Compilation and execution of a Program and a Query

A BPL program is a set of facts, rules and similarity equations plus a goal. In order to launch it, you must follow these steps:

1. Create a new BPL program or open an old one.
2. Pull down the mouse and select the option `compile` in the `Action` menu or type the keys "`ctrl-C`" or click the `ok` icon. Once the program has been compiled, the result of the compilation can be visualized in the out window. If everything is ok the message is "la compilacion se realizo correctamente". Then the program can be executed.
3. For executing the program, select the `execute` option on the `Action` menu or type the keys "`ctrl-E`" or click on the `sun` icon.

WARNING: If the current query success, the answer is shown in the out window. In order to obtain all the alternative answers you must write the `interrogation` key on the query window. Also you can select the option `More responses` in the Action Menu, o you can type the keys "`Ctrl-S`.

3.4 Miscellaneous

The BPL environment provides two commands to reset the information shown by the out window and the code area window:

- The command `clear` deletes the information shown in the out window. In order to execute this command, type "`clear`" into the query window.
- The command `reset` deletes the information shown in the code area window. In order to execute this command, type "`reset`" into the query window.

We can impose a limit to the expansion of the search space in a computation by what we called a "lambdacut". When lambdacut is set, the weak unification process fails if the computed approximation degree goes below the stored lambdacut value. Therefore, the computation also fails and all possible branches starting from that choice point are discarded. The lambdacut command can be used to set a new lambdacut value. typing "`cut(N)`" into the query window you set the lambdacut value to `N`.

4 The Bousi∼Prolog programming language

In this section we briefly summarize the features of Bousi∼Prolog as it has been implemented in the present version supported by the SWAM. We concentrate on the syntactical aspects.

The programming language we call Bousi∼Prolog is an extension of the standard Prolog language with a similarity relation defined on a syntactic domain. Therefore, the syntax is mainly the Prolog syntax but enriched with a built-in symbol used for describing similarity relations (actually, fuzzy binary relations which are automatically converted into similarity relations) by means of *similarity equations* of the form:

```
<alphabet symbol> ~ <alphabet symbol> = <similarity degree>
```

meaning that two constants, n-ary function symbols or n-ary predicate symbols are similar with a certain degree.

A BPL program is a sequence of Prolog facts and rules plus a sequence of similarity equations.

Bousi∼Prolog uses the similarity-based SLD principle [2] (also called weak SLD resolution) as operational semantics.

4.1 The weak unification operator

Bousi∼Prolog implements a weak unification operator, denoted by "∼", which is the fuzzy counterpart of the syntactical unification operator "=" of standard Prolog. It can be used, in the source language, to construct expressions like "`Term1 ~ Term2 =:= Degree`" which is interpreted as follows: The expression is true if `Term1` and `Term2` are unifiable by similarity with approximation degree `AD` equal to `Degree`. In general, we can construct expressions

```
Term1 ~ Term2 <op> Degree
```

where "`<op>`" is a comparison arithmetic operator (that is, an operator in the set `{=:=, =\=, >, <, >=, =<}`). Observe that the expression "`Term1 ~~ Term2`" is syntactic sugar of "`Term1 ~~ Term2 > 0`". These expressions may be introduced in a query as well as in the body of a clause.

WARNING: In the present implementation, the weak unification operator is denoted by "`~`" instead of by "`~~`" (used for the high level implementation of Bousi∼Prolog). Also we use the symbol "`~`" for representing similarity equations.

WARNING: In the high level implementation of Bousi∼Prologis possible to use the following construction: `Term1 ~~ Term2 = Degree` which success if `Term1` and `Term2` are weak unifiable with approximation degree `Degree`; otherwise fails. When `Degree` is a variable it is bound to the unification degree of `Term1` and `Term2`. This construction is not available in the low level implementation of Bousi∼Prolog.

4.2 Some limitations

For the present version, the SWAM does not implement all the features of full Bousi∼Prolog. It implements a pure subset of Prolog with lists and arithmetic plus similarity equations and the weak unification operator. However it does not implement Input/Output predicates and the built-in predicates described in the manual of the high level implementation of Bousi∼Prolog.

In the future we want to develop the SWAM to cover all the present and future features of Bousi∼Prolog in a more efficient implementation.

OBSERVATION: Bousi∼Prolog uses the standard cut predicate, "`!`" of the Prolog language in an indirect way, embedded into more declarative predicates and operators, such as: `not` (weak negation as failure —see below—), `\+` (crisp negation as failure —see below—) and `->` (if-then and if-then-else operators). However, the low level implementation of Bousi∼Prolog we are describing implements the cut operator.

References

1. Pascual Julián-Iranzo and Clemente Rubio-Manzano. Introducing Weak Unification into the WAM. Dep. of Information Technologies and Systems, University of Castilla-La Mancha. Technical Report, 2005.
2. Maria I. Sessa. Approximate reasoning by similarity-based sld resolution. *Theoretical Computer Science*, 275(1-2):389–426, 2002.