DLV External Built-In Predicates Testing Tool
User Guide

1 Overview

This user guide describes how to install and run the 'DLV External Built-in Testing' (DEBT) software, a tool which aims to help a DLV user testing an external built-in predicate he has defined.

We suppose the reader is a DLV programmer, so he already knows the DLV system and language. In addition, we suppose he is aware of the opportunity offered from the most recent DLV releases, to perform calls to external built-in predicates defined by means of external C++ functions. However, for a full description of the usage and capabilities of DLV please refer to the DLV homepage (http://www.dlvsystem.com), the DLV online user manual (http://www.dbai.tuwien.ac.at/proj/dlv/man/), the DLV online user tutorial (http://chkoch.home.cern.ch/chkoch/dlv/dlv_tutorial.html). For a description of the external built-ins new feature please refer to the External Builtins User Manual. Moreover, in the DLV External Builtins Tutorial many practical examples of built-in definitions may be found.

The DEBT tool is designed as a script to be used in a standard UNIX environment. Starting from the built-in predicate name and a table of input and expected output values for this, it generates proper DLV programs, runs them, compares the results and prints a report.

In the following sections we first describe what is needed to run the tool and what has to be modified to adapt the script to a specific environment. Then we show in detail how to run the script and all the options which may be specified. Finally, some complete usage examples are shown.

1. Installation notes

This section specifies the software your system need to be installed for using the DEBT tool and what to do for correctly installing the DEBT files.

1.1. Requirements

- The DEBT tool is based on the BETL II test toolkit. Both software and documentation may be downloaded from: http://www.comnex.net/projects/betl/download.htm.
- DEBT and BETL II tools are both Perl script so PERL 5.004 or up is required (see http://www.perl.com).
- The BETL II test toolkit exploits the following Perl modules: Getopt::Long, Data::Dumper, IO::File, nsgmls. Any good Perl distribution already includes this modules, anyway please refer to BETL II documentation for further details.

1.2. Installation

1. Verify all requirements specified in the previous paragraph;
2. Download the DEBT installation file named: 'debt.tar.gz', a compressed archive file;
3. Install the downloaded archive file into a library folder (possibly a folder within PATH environment variable, to invoke the tool from anywhere), for example:

   mkdir /home/username/debt

   cd /home/username/debt

   tar –xvzf /path/to/download/debt.tar.gz
4. modify the 'testbuiltin' shell script:
   - replace the value of DLV_TO_TEST variable with the full path to your DLV executable;
   - replace the value of PATH_TO_LIB variable with the path to your DLV external built-in libraries folder (usually the LIB folder, including extpred.h and extpred.o);
   - replace the value of PATH_TO_BETL variable with the path to your BETL compiler and interpreter folder (that one including BETLc and BETLi files).

Note: all the installed files must reside in the same folder.

2. Test execution

This section shows the syntax of the DEBT tool. The meaning and correct form of input parameters is also specified. Finally, all the accepted command line options are listed and explained.

2.1. Synopsis

To invoke the DEBT tool just type:

```
$ testbuiltin builtinName tableName [options]
```

where *builtinName* and *tableName* are two mandatory command line parameters and [options] stands for a list of optional settings to better fit your needs. See below for an explanation of what builtinName and tableName represent, as well as for a complete specification of all the possible options.

2.2. Command line parameters

The first parameter is the name of the external built-in predicate being tested.

The second parameter is the name of a text file containing a table with a set of records representing some input values and expected output values for the external built-in predicate being tested. Such a text file has to respect the following syntax: a record for each line, and record values separated by the comma character.

Example: a valid text file with a table values for the external built-in predicate '#fatt' computing the factorial of a given number may have the form:

```
2,2
3,6
4,24
... and so on ...
```

If a value is a string, you have to enclose it between double quotes to distinguish it from a symbolic value.

Note: If no different option is specified, the first parameter, that is the name of the external built-in predicate, is also used:

- as the name (with a .C suffix) for locating the C++ file containing the source code for the built-in predicate. Note that the source file is always expected to be

found in the folder indicated by the PATH_TO_LIB variable (whose value has to be set at installation time);

– as the name of the dynamic library (with a .so suffix) containing the object code for the built-in predicate.

2.3. Command line options

The command line options accepted are:

- -s | --source sourceFileName
  to specify the file name with C++ source code for the external built-in predicate to be tested. This option should be used only if the source file has different name with respect to the built-in. Note that the source file is always expected to be found in the folder indicated by the PATH_TO_LIB variable (whose value has to be set at installation time);
- -l | --lib libName
  to specify the dynamic library name including the object code for the external built-in predicate to test. This option should be used only if the dynamic library has different name with respect to the built-in;
- -h | --help
  to show an help message and terminate;
- -e | --extendTable
  to test the built-in predicate also on different values, obtained considering all possible combination of values included in the *tableName* file. At the moment, if the built-in fails on some of those extended values the error is not reported. This will be fixed in a future release of the tool;

Each option can be specified both in the short form (‘-’ followed by a single character) and in the long one (‘--’ followed by the option name). The order of the options is meaningless.

3. Examples

The following examples illustrate DEBT tool usage to test two specific built-in predicates for string manipulation. In particular, for each example, we will show: built-

in source code, table values used for testing, DEBT tool invocation using different command line options and the corresponding produced output.

3.1. String length

In this first example we consider an external built-in function computing the length of a given character string. Two ‘oracle’ functions were defined: the “base” one having pattern *ii* and another one with pattern *io* taking as input a character string and returning as output the string length. The name given to this built-in function is ‘strlen’, so a DLV program may use the predicate #strlen.

```
#include "extpred.h"

#ifdef __cplusplus
extern "C" {
#endif

      BUILTIN(strlen,ii) {
            if (argv[0].isString() && argv[1].isInt()) {
                  const char* s = argv[0].toString();
                  int l = argv[1].toInt();
                  if (l == (int)strlen(s))
                        return true;
            }
            return false;
      }

      BUILTIN(strlen,io) {
            if (argv[0].isString()) {
                  const char* s = argv[0].toString();
                  argv[1] = CONSTANT(strlen(s)-1);
                  return true;
            }
            return false;
      }

#ifdef __cplusplus
}
#endif
```

Suppose now that this source code was saved in a text file named ‘stringLength.C’ and it was compiled in order to obtain the dynamic library ‘stringLength.so’. Now create a text file containing the following table value and name it ‘strlenValues’:

```
"mickey",6
"mouse",5
"charlie",7
"woodstock",9
"beepbeep",8
```

Writing on a UNIX shell the following command:

```
$ testbuiltin strlen strlenValues –s stringLength –l stringLength
```

we obtain this report message:

```
*WARNING* Oracle function(s) for pattern(s): 'oi' not defined.
  test 2: *FAIL*    strlen_io
= line    1 wanted: strlen("beepbeep",8) got: strlen("beepbeep",7)
= line    2 wanted: strlen("charlie",7) got: strlen("charlie",6)
= line    3 wanted: strlen("mickey",6) got: strlen("mickey",5)
= line    4 wanted: strlen("mouse",5) got: strlen("mouse",4)
= line    5 wanted: strlen("woodstock",9) got: strlen("woodstock",8)
= ---
```

The warning message just remarks that the oracle function having pattern *oi* is not defined for this built-in predicate, in order to ensure that it is intentional. The rest of the message points out the failing of *io* oracle test, showing the expected result (‘= line 1 wanted: …’) and what the function returns instead (‘got: …’). This is because we made on purpose a mistake in the source code shown above (returning the string length minus one). If we replace the line:

```
argv[1] = CONSTANT(strlen(s)-1);
```

with:

```
argv[1] = CONSTANT(strlen(s));
```

and then recompile the library, we obtain this report message:

```
*WARNING* Oracle function(s) for pattern(s): 'oi' not defined.
OK! All defined oracle functions succeeded.
```

Using the -e|--extendTable option for this built-in predicate we do not obtain any new answer set element, because all possible combination of given input values do not produce new valid results.

3.2. String concatenation

The second example consists in the concatenation of two strings. For this predicate, having arity 3, we define four different oracles with the following patterns: *iii*, *iio*, *ioi*, *oii*. First and second terms are the head part and the tail part of the third term respectively. The name given to this built-in function is ‘strcat’, so a DLV program may use the predicate #strcat.

```
#include "extpred.h"

#ifdef __cplusplus
extern "C" {
#endif

      BUILTIN(strcat,iii) {
            if (argv[0].isString() && argv[1].isString() &&
    argv[2].isString()) {
                  const char* s1 = argv[0].toString();
                  const char* s2 = argv[1].toString();
                  const char* s = argv[2].toString();
                  char* dest = new char[strlen(s1)+strlen(s2)+1];
                  strcpy(dest,s1);
                  strcat(dest,s2);
                  bool ris = (strcmp(dest,s) == 0);
                  delete [] dest;
                  if (ris)
                        return true;
            }
            return false;
      }

      BUILTIN(strcat,iio) {
            if (argv[0].isString() && argv[1].isString()) {
                  const char* s1 = argv[0].toString();
                  const char* s2 = argv[1].toString();
                  char* dest = new char[strlen(s1)+strlen(s2)+1];
                  strcpy(dest,s1);
                  strcat(dest,s2);
                  argv[2] = CONSTANT(dest,true);
                  return true;
            }
            return false;
      }

      BUILTIN(strcat,ioi) {
            if (argv[0].isString() && argv[2].isString()) {
                  const char* s1 = argv[0].toString();
                  const char* s = argv[2].toString();
                  int ls1 = strlen(s1);
                  int ls = strlen(s);
                  if (ls > ls1)
                        if (strncmp(s,s1,ls1) == 0) {
                              char* dest = new char[ls-ls1+1];
                              strcpy(dest,s+ls1);
                              argv[1] = CONSTANT(dest,true);
                              return true;
                        }
            }
            return false;
      }

      BUILTIN(strcat,oii) {
            if (argv[1].isString() && argv[2].isString()) {
                  const char* s2 = argv[1].toString();
                  const char* s = argv[2].toString();
                  int ls2 = strlen(s2);
                  int ls = strlen(s);
```

```
                    if (ls > ls2)
                          if (strncmp(s+(ls-ls2),s2,ls2) == 0) {
                                char* dest = new char[ls-ls2+1];
                                strncpy(dest,s,ls-ls2);
                                dest[ls-ls2] = '\0';
                                argv[0] = CONSTANT(dest,true);
                                return true;
                          }
                    }
                    return false;
              }

        #ifdef __cplusplus
        }
        #endif
```

Suppose now that this source code was saved in a text file named ‘strcat.C’ and it was compiled in order to obtain the dynamic library ‘strcat.so’. Now create a text file containing the following table value and name it ‘strcatValues’:

```
"Mickey",”Mouse”,”MickeyMouse”
"spider”,”man”,”spiderman”
"super”,”man”,”superman”
```

Writing on a UNIX shell the following command:

```
$ testbuiltin strcat strcatValues
```

we obtain this report message:

```
*WARNING* Oracle function(s) for pattern(s): 'ioo oio ooi' not
defined.
OK! All defined oracle functions succeeded.
```

The warning message just remarks that the oracle functions having patterns *ioo*, *oio*, *ooi* are not defined for this built-in predicate, in order to ensure that is intentional.
Note that, in this case, we have not used the `–s` and `-l` command line option because both source code file and dynamic library have the same name as the built-in predicate.
Using the -e|--extendTable option:

```
$ testbuiltin –e strcat strcatValues
```

in addition to the usual report message, we obtain a (long) table of further values computed by the built-in predicate taking as input different combination of user provided values:

The following extended table values were also generated and exploited for testing:

```
"Mickey","Mickey","MickeyMickey"
"Mickey","MickeyMouse","MickeyMickeyMouse"
"Mickey","spider","Mickeyspider"
"Mickey","man","Mickeyman"
"Mickey","spiderman","Mickeyspiderman"
"Mickey","super","Mickeysuper"
"Mickey","superman","Mickeysuperman"
"Mouse","Mickey","MouseMickey"
"Mouse","Mouse","MouseMouse"
"Mouse","MickeyMouse","MouseMickeyMouse"
"Mouse","spider","Mousespider"
"Mouse","man","Mouseman"
"Mouse","spiderman","Mousespiderman"
"Mouse","super","Mousesuper"
"Mouse","superman","Mousesuperman"
"MickeyMouse","Mickey","MickeyMouseMickey"
"MickeyMouse","Mouse","MickeyMouseMouse"
"MickeyMouse","MickeyMouse","MickeyMouseMickeyMouse"
"MickeyMouse","spider","MickeyMousespider"
"MickeyMouse","man","MickeyMouseman"
"MickeyMouse","spiderman","MickeyMousespiderman"
"MickeyMouse","super","MickeyMousesuper"
"MickeyMouse","superman","MickeyMousesuperman"
"spider","Mickey","spiderMickey"
"spider","Mouse","spiderMouse"
"spider","MickeyMouse","spiderMickeyMouse"
"spider","spider","spiderspider"
"spider","spiderman","spiderspiderman"
"spider","super","spidersuper"
"spider","superman","spidersuperman"
"man","Mickey","manMickey"
"man","Mouse","manMouse"
"man","MickeyMouse","manMickeyMouse"
"man","spider","manspider"
"man","man","manman"
"man","spiderman","manspiderman"
"man","super","mansuper"
"man","superman","mansuperman"
"spiderman","Mickey","spidermanMickey"
"spiderman","Mouse","spidermanMouse"
"spiderman","MickeyMouse","spidermanMickeyMouse"
"spiderman","spider","spidermanspider"
"spiderman","man","spidermanman"
"spiderman","spiderman","spidermanspiderman"
"spiderman","super","spidermansuper"
"spiderman","superman","spidermansuperman"
"super","Mickey","superMickey"
"super","Mouse","superMouse"
"super","MickeyMouse","superMickeyMouse"
"super","spider","superspider"
"super","spiderman","superspiderman"
"super","super","supersuper"
"super","superman","supersuperman"
"superman","Mickey","supermanMickey"
"superman","Mouse","supermanMouse"
"superman","MickeyMouse","supermanMickeyMouse"
```

```
"superman","spider","supermanspider"
"superman","man","supermanman"
"superman","spiderman","supermanspiderman"
"superman","super","supermansuper"
"superman","superman","supermansuperman"

*WARNING* Oracle function(s) for pattern(s): 'ioo oio ooi' not
defined.
OK! All defined oracle functions succeeded.
```