

Due: 5pm, Thursday, 24 October 2002

The objective of this project is to further practice and assess your understanding of logic programming and Prolog. You will write a pretty printer for Prolog terms. The input for this will be any ground Prolog term, and the output will be a box term, similar to what we used in project 1, specifying how to display that Prolog term.

For this project, there are core requirements that you must meet to be awarded the full 15 marks for this project. Beyond the core requirements, there are several optional features, each of which has an associated bonus value. For each optional feature correctly implemented, you will receive the corresponding bonus value *in addition* to the marks received for the core functionality. Thus it will be possible to receive more than 15 marks for this project. However, it will not be possible to receive more than 30 marks for the two projects together, so the bonus marks may only be used to make up for points lost on project 1, not for points lost on the labs or final exam.

Note well: any questions regarding the specification for this project should be posted to the `cs.255` newsgroup or emailed to `ask255@cs.mu.oz.au`. Questions asked to the newsgroup are likely to be answered sooner.

New Box Term

To simplify this project, we introduce a new kind of box term. **You do not need to implement the box term described in this section.** You will be provided with a new `formatter.pl` program which implements this box term; this is provided for reference only, it is not actually needed to implement this project.

The new box term is of the form

```
flex(Bboxes, Hsep, Indent)
```

This box is similar to an `horv` box, in that its *Bboxes* will be arranged horizontally if they fit, or vertically otherwise. However, it is more flexible, in that you may specify where line breaks are permitted by including `n1` terms in the list of *Bboxes*. Each `n1` term marks a place where a line break is permitted. If the entire `flex` box will fit on a single line, then the `n1` boxes are simply ignored. Otherwise, each `n1` box marks a place for a line break;

the boxes between `n1` boxes are assembled into a single row, and the rows are assembled vertically. The list of boxes may also contain space boxes. When the `flex` box is laid out horizontally, each space box is replaced with a box the width of *Hsep*; otherwise it is replaced with a box the width of *Indent*. Finally, a `sep` box is equivalent to an `n1` followed immediately by a space box.

Core Functionality

You will write the following predicate:

```
boxify(Term, Box)
```

Box is the appropriate box term that could be passed to `format_text` or `format_postscript` from project 1 to display *Term* in an attractive form. *Term* may be assumed to be a ground term.

An atom *X* should produce the box term `text(X)`. A number *N* should produce the box term `text('N')`; that is, it should produce `text` of an atom that spells out the digits of the number (e.g., by calling `number_codes(N,C)`, `atom_codes(Name,C)`).

A compound term $f(A_1, A_2, \dots, A_n)$ should produce the box term:

```
flex([text(f), text(' '), B1, text(' '), sep, B2, text(' '), sep,  
    ..., Bn, text(' ')],  
    text(n), h([text(f), text(' '), empty]))
```

where each *B_i* is the box term corresponding to term *A_i*.

For example:

```
?- boxify(f(a,b), Box).
```

```
Box = flex([text(f), text(' '), text(a), text(' '), sep,  
           text(b), text(' ')],  
           text(n), h([text(f), text(' '), empty])) ;
```

No

Bonus Functionality

In addition to the 15 marks for implementing the core functionality above, you may earn extra marks for implementing some or all of the bonus features described below. These features are independent of one another; you may get marks for any of them without implementing the others.

Handling quoting [1 bonus mark]

When boxing an atom or functor that does not follow the rules for unquoted Prolog atoms (i.e., atoms that do not begin with a small letter or that contain a character other

than a letter, digit, or underscore), you may quote it. This is done by beginning and ending its name with a single quote character, and doubling any single quote character inside it. Also, quoted atoms should be rendered in courier font, to make them stand out. Thus the box term for atoms needing quoting should be `font(text(Quoted), [family=courier])`, where *Quoted* is the atom, quoted as specified above.

Handling '\$VAR' terms [2 bonus marks]

As mentioned above, this code works only for ground terms; thus output will not show variables. To allow output to show variable names, you may treat terms whose functor is '\$VAR' and arity is 1 specially. If the argument of such a term is an atom, it will look like a variable name (it will begin with a capital letter or underscore). In such cases you should ignore the '\$VAR' wrapper and take the atom as the variable's name.

If the argument of the '\$VAR' term is an integer, this is taken to indicate a variable name as follows. 0 indicates variable name 'A', 1 indicates 'B', and so on up to 25 indicates 'Z'. 26 indicates 'A1', 27 'B1', up to 51 indicates 'Z1', 52 indicates 'A2', and so on.

In any case, to make the output more attractive, variable names should be shown in italic, so the result should be a box of the form

```
font(text(Name), [family=times, slant=italic])
```

where *Name* is the variable name as described above.

Handling list notation [2 bonus marks]

As described, the boxifier does not handle list notation specially, so lists would appear as ordinary terms with '?' as functor. Instead lists should be shown with the usual square bracket notation. Therefore, a list that ends with a [] term should be turned into box terms of the form

```
flex([text('[]'), B1, text(' '), sep, B2, ..., text(' '), sep, Bn, text('[]')],  
text(n), text('[]'))
```

where B_1, \dots, B_n are the box terms corresponding to the elements of the list. A list that ends with any term T other than [] should be boxified to

```
flex([text('[]'), B1, text(' '), sep, B2, ..., text(' '), sep, Bn-1,  
nl, v([text('[]'), width(text('[]'))], empty)], Bn, text('[]')],  
text(n), text('[]'))
```

where B_1, \dots, B_{n-1} are the box terms corresponding to the proper elements of the list, and B_n is the box corresponding to the final tail of the list.

Submission

You should write this program entirely in a single source file named `prettyprint.pl`. You should include your name, login, and student number in a header comment at the beginning of the file. You should also briefly document what this program does.

You should submit your `prettyprint.pl` file using

```
submit 255 proj2 prettyprint.pl
```

and verify it using

```
verify 255 proj2
```

The output from `verify` should contain the output from test runs of your program; if it does not, your program did not work. *It is your responsibility to verify your submission and check the output of verify.*

Late submission will be made using the command

```
submit 255 proj2.late prettyprint.pl
```

Late submissions will incur a penalty of 0.5% per hour late, including evening and weekend hours. This means that a perfect project that is much more than 4 days late will fail! If you have a medical or similar compelling reason for being late, you should contact Chris Charnes (`charnes@cs.mu.oz.au`) as early as possible to ask for an extension (preferably before the due date).

Your program will be compiled by the Prolog command `[prettyprint].` of the SWI Prolog system installed on the department's student spare machines (eg, `cat`, `lister`, `holly`, etc., but not `queeg`). It should compile with no warnings or error messages. Note that a program that compiles and runs correctly using some other Prolog system, or even the same system on a different architecture, may not compile cleanly, and may not execute correctly, when compiled and run as specified above. *It is your responsibility to test your program on a student spare machine as specified above.*

Assessment

Your program will be assessed entirely on whether or not it correctly executes a series of test cases. Correctness will be determined based on whether or not your code succeeds with the correct solution term. Bonus marks will be awarded on the same basis.

Some of the test cases and their expected output will shortly be made available in the student data area for the subject. Other test cases will not be made public.

Note Well:

This project is part of your final assessment, so cheating is not acceptable. Any form of material exchange, whether written, electronic or any other medium, is considered cheating, and so is the soliciting of help from electronic newsgroups. Providing undue assistance is considered as serious as receiving it, and in the case of similarities that indicate exchange of more than basic ideas, formal disciplinary action will be taken for all involved parties.