A package for booleans expressions

Didier Rémy

October 6, 2006

Abstract

This package provides macros for combining booleans expressions, for which there is poor support in low-level TEX or even LATEX. By contrast with the package `ifthen`, boolean expressions are *first class values*. We represent booleans in Church's style: we need not and "ifthenelse", as applying then to two arguments will just pick the right branch.

1 Boolean expressions throw examples

Consider the following expressions:

```
\def \b {\AND{\TRUE}{\OR {\FALSE}{\TRUE}}}
\b->\AND {\TRUE }{\OR {\TRUE }{\FALSE }}
```

We can turn the expression into a conditional that will take two arguments and evaluate one according to the condition. `\IF{\b}{Yes}{No}`, producing "*Yes*".

Note that if we use the conditional several times it will be evaluated several times. We may evaluate a boolean and bind its value to a command as follows:

```
\defboolval{\bv}{\b}
\bv#1#2->#1
```

Of course, we may now replace `\b` by `\bv` in the above expression, *i.e.* write `\IF{\bv}{Yes}{No}` leading to the same resultThe macro `\newboolval` is equivalent to `\letboolval` except that it is an error if the command name passed as first argument is already bound.

We could as well have required evaluation in the first place:

```
\letboolval{\bv}{\AND{\TRUE}{\OR {\FALSE}{\TRUE}}}
\bv#1#2->#1
```

One may observe that there is no difference between `\bv` and `\TRUE`.

```
\TRUE#1#2->#1
```

For convenience also provide n-ary versions `\ANDL` and `\ORL` of `\AND` and `\OR` operations, using comma-separated arguments (forming a single latex argument). Finally, here is a large example that summarizes all operators.

```
\def \b
  {\ANDL
    {\ANDL{},%
     \ANDL{\TRUE},%
     \ANDL{\TRUE ,\NOT \FALSE},\TRUE ,\NOT {\FALSE},%
     \ORL{},%
     \ORL{\FALSE},%
     \ORL {\NOT \TRUE,\FALSE}}}
\letboolval {\bv} {\b}
```

```
\b->\ANDL {\ANDL {},\ANDL {\TRUE },\ANDL {\TRUE ,\NOT \FALSE
},\TRUE ,\NOT {\FALSE },\ORL {},\ORL {\FALSE },\ORL {\NOT
\TRUE ,\FALSE }}
```

Which we may evaluate to a boolean value:

```
\letboolval {\bv} {\b}
```

```
\bv#1#2->#1
```

Boolean constanst and operators have uppercase names to avoid conflict with basic latex commands or other packages.

In some contexts, when there is no conflict, one may use the command `\BooleanLowerNames` to define lowercase abbreviations for all uppercase names.

2 Lifting primitive TEX conditionals

The TEX primitive are rather inconvenient to use, as they require the use of ad hoc patterns. We lift the most frequent patterns as LATEX church booleans.

The most general command is `\booltex` which takes a TEX condition as argument and returns a boolean expression. For example,

```
\def \b {\booltex{\ifnum 1>2}}
```

```
\b->\booltex {\ifnum 1>2}
```

of which we may also force evaluation to a boolean value, as for any other boolean expresssion.

```
\def \b {\booltex{\ifnum 1>2}}
\letboolval \bv {\texbool{\ifnum 1>2}}
```

```
\bv#1#2->#2
```

We may also combine it with other expressions, indeed, as in:

```
\ORL{\texbool{\ifnum 1>2},\texbool{\ifhmode}}{Yes}{No}
```

which evaluates to "*Yes*"

Other lifting primitives are:

- `\ifequalbool` takes two arguments and returns true if both arguments are equal.

  For TEX experts, this is equivalent to the following definition (except for the names of `\testa` and `\testb`):
  `#1#2->\def\testa{#1}\def\testb{#2}\texbool{\ifx\testa\testb}`
  Putting the argument in a macro first avoids the pitfall of `\ifx #1#2` when one of the argument is empty. However, in some cases, one may need the primitive TEX test `\ifx`.

- `\ifemptybool` is equivalent to `\ifequalbool{}`, that is, it simply tests its argument for emptiness.

- `\ifxbool` is a shorthand for `#1#2->\texbool{\ifx #1#2}`. This is usually true if the two arguments expands to the same token, except for pathological cases such as when one of the argument is empty. See the TEX-book for details.

- `\ifybool` is a shorthand for the common TEX idiom that puts only the first argument in a macro before testing, so as to avoid the empty argument pitfall. However, it keeps the second arguemt as given, as this is usually a macro whose definition and not the macro itself should be used for comparisson. `#1#2->\def \test{#1}\ifxbool{\test}{#2}`.

3 String matching

This package also defines string matching, mamely three macros `\ifstrprefix`, `\ifstrinfix`, `\ifstrsuffix` to analyze strings of tokens.

They take two arguments, a pattern sequence and a string. The pattern is search for in the string. The macros returns a boolean that tells whether the

pattern is a prefix, infix, or suffix of the string. Here are examples:

Command followed by `{search in}{Y}{N}`	**Expansion**
`\ifstringinfix{search in}`	**Y**
`\ifstringinfix{search}`	**Y**
`\ifstringinfix{ear}`	**Y**
`\ifstringinfix{in}`	**Y**
`\ifstringinfix{ }`	**Y**
`\ifstringinfix{ch in}`	**Y**
`\ifstringinfix{searchin}`	**N**
`\ifstringinfix{ing}`	**N**
`\ifstringprefix{sea}`	**Y**
`\ifstringprefix{in}`	**N**
`\ifstringprefix{thesea}`	**N**
`\ifstringsuffix{in}`	**Y**
`\ifstringsuffix{ch in}`	**Y**
`\ifstringsuffix{sea}`	**N**

An more general auxiliary function `\stringmatch` is actually used for sharing all comparissons. It takes an extra parameter as first argument that is the action to performed after matching. This actions must three-argument function receiving in order, a boolean value that tells whether the match succeeded, the prefix and the suffix (which are meaningless in case of failure).

For example the following action may be defined:

```
\def\gnu#1#2#3{#3{#1\_#2}{draft}}
```

to cut off the infix and return a default value. Then, we have:

`\stringmatch {\gnu }{job}{ifjobname}` evaluates to "*if_name*"

`\stringmatch {\gnu }{draft}{ifjobname}` evaluates to "*draft*"