A Simple Function for Parsing Model Formulae

Ross Ihaka

Department of Statistics
University of Auckland

1 Introduction

This code accepts a class of *model formulae* that describe models for graphical applications. The Backus-Naur Form for the grammar describing the formulae is as follows:

formula:	' ∼' *right-side*
	left-side ' ∼' *right-side*
left-side:	*variable-list*
right-side:	*variable-list*
	variable-list ' |' *variable-list*
variable-list:	*variable*
	variable-list ' +' *variable*

At the lowest level, a formula is composed of *variables*. In this context, a variable is any R expression other than a simple sum of two variables (i.e. $expr_1 + expr_2$).[1] A *variable-list* is a set of variables separated by '+.'

At the highest level, a *formula* has exactly one '∼' in it with an optional *left-side* and a required *right-side*. A *left-side* is a variable-list and a *right-side* consists of a variable-list followed by an optional ' | ' with a following variable-list.

Note that in this gammar, *left-side* is simply a synonym for *variable-list*. This means that the grammar can be simplified by removing the *left-side* production.

formula:	' ∼' *right-side*
	variable-list ' ∼' *right-side*
right-side:	*variable-list*
	variable-list ' |' *variable-list*
variable-list:	*variable*
	variable-list ' +' *variable*

[1] Such simple sums can be protected by *quoting* them with the `I()` function.

The code provided by this source module takes apart a model formula described by by the grammar and returns a list containing components named `lhs`, `rhs`, and `condition`.

The function provided by this R code is an example of a recursive-descent parser that can be used to parse a restricted class of model formulae.

2 Code Layout

This software consists of a *main function* together with a number of supporting *utility functions* and associated constant definitions. The code is collected in a file called `formulae.R`, which is structured as follows.

2a ⟨*formulae.R* 2a⟩≡

```
parseFormula = local({
    ⟨constants 2b⟩
    ⟨parsing functions 2c⟩
})
```

Defines:
`parseFormula`, never used.
This code is written to file `formulae.R`.

The constants, utility functions and main function are defined within a `local` block whose value is the main function. This provides a way of hiding the utility functions in a scope which is only visible within the body of the function `parseFormula`.

3 Constants

The local definition of constants provides a quick way of looking up some important values that are used in the parser. This is essentially an optimisation that avoids repeated calls to `as.name` to carry out a symbol look up.

2b ⟨*constants* 2b⟩≡

```
TILDE    = as.name("~")
BAR      = as.name("|")
PLUS     = as.name("+")
IDENTITY = as.name("I")
```

4 Parsing Functions

The actual parsing of formulae is carried out by three functions; one for each rule in the grammar. In addition, there is a function that removes any quotation using the `I()` mechanism.

2c ⟨*parsing functions* 2c⟩≡

```
⟨strip protection 3a⟩
⟨variable list processing 3b⟩
⟨right-hand side processing 4⟩
⟨main parsing function 5⟩
```

4.1 Stripping Quotation

For expressions of the form `I(`*expr*`)`, the value of *expr* is returned. Any occurences of `I` that do not have exactly one argument will trip an error.

3a ⟨*strip protection* 3a⟩≡

```
stripI =
    function(e) {
        if (is.call(e) && identical(e[[1]], IDENTITY)) {
            if (length(e) == 2)
                e[[2]]
            else
                stop("incorrect use of I()")
        }
        else e
    }
```

Defines:
`stripI`, used in chunk 3b.

4.2 Variable List Processing

Variable lists are processed as follows:

- If the argument consists of an expression of the form $expr_1 + expr_2$, the elements in $expr_1$ are extracted by a recursive call to `element.list` and then the element in $expr_2$ (with any protecting `I()` stripped) is appended to this list.
- Any other argument is treated as simple formula element. Such an elements is stripped of any protecting `I()` quotation and returned in a list.

3b ⟨*variable list processing* 3b⟩≡

```
element.list =
    function(expr) {
        if (is.call(expr) && identical(expr[[1]], PLUS))
            if (length(expr) == 3)
                c(element.list(expr[[2]]),
                  stripI(expr[[3]]))
            else
                stop("invalid element list")
        else
            list(stripI(expr))
    }
```

Defines:
`element.list`, used in chunks 4 and 5.
Uses `stripI` 3a.

4.3 Right-Hand Side Processing

If the argument to this function is an expression of the form $expr_1 \mid expr_2$ the result is a list containing a component called `rhs` containing the variable list extracted from $expr_1$ and a component `condition` containing the variables extracted from $expr_2$. In any other case, the result is similar, but the `condition` component is `NULL`.

4 ⟨*right-hand side processing* 4⟩≡

```
right.side =
    function(e) {
        if (is.call(e) && identical(e[[1]], BAR)) {
            if(length(e) == 3)
                list(rhs = element.list(e[[2]]),
                     condition = element.list(e[[3]]))
            else
                stop("invalid formula rhs")
        }
        else
            list(rhs = element.list(e),
                 condition = NULL)
    }
```

Defines:
`right.side`, used in chunk 5.
Uses `element.list` 3b and `formula` 5.

4.4 Main Parsing Function

The main parsing function processes an expression of the from $expr_1 \sim expr_2$ and returns a list containing components `lhs`, `rhs` and `condition` which hold the lists of variables extracted from the three (possibly empty) components of the formula.

5 ⟨*main parsing function* 5⟩≡

```
formula =
    function(expr) {
        if (is.call(expr)
                && identical(expr[[1]], TILDE)) {
            if(length(expr) == 2)
                c(lhs = list(),
                  right.side(expr[[2]]))
            else if (length(expr) == 3)
                c(lhs = list(element.list(expr[[2]])),
                  right.side(expr[[3]]))
            else
                stop("invalid formula")
        }
        else
            stop("missing ~ in formula\n")
    }
```

Defines:
`formula`, used in chunk 4.
Uses `element.list` 3b and `right.side` 4.

5 Defined Chunks

⟨*constants* 2b⟩
⟨*formulae.R* 2a⟩
⟨*main parsing function* 5⟩
⟨*parsing functions* 2c⟩
⟨*right-hand side processing* 4⟩
⟨*strip protection* 3a⟩
⟨*variable list processing* 3b⟩

6 Index

`element.list`: 3b, 4, 5
`formula`: 4, 5
`parseFormula`: 2a
`right.side`: 4, 5
`stripI`: 3a, 3b