# Best practices for parsing a formula in R programming

Ronald (Ryy) Glenn Thomas

2025-07-01

## Table of contents



## 1 Introduction

R developers can make use of the `formula` object class to allow succinct (unquoted) input to functions. However, its a bit unclear what the best strategy is to parse the formula inside the function to extract and validate the instructions the formula provides.

In this post we'll present several alternative methods for consideration. As with most coding challenges there is no single best answer but some are more elegant than others.

## 2 Methods

Consider the following function to parse a formula. The formulas we want to test are of the form `y ~ x1 + x2 | g` where `y` is the dependent variable, `x1` and `x2` are the independent variables and `g` is the grouping variable.

```
frm <- function(form) {
  vars <- all.vars(form)
  y_var <- deparse(form[[2]])
  g_bar <- deparse(form[[c(3, 1)]])
  g_var <- NULL
  if (g_bar == "|") {
    x_vars <- all.vars(form[[c(3, 2)]])
    g_var <- all.vars(form[[c(3, 3)]])
  } else {
    x_vars <- all.vars(form)[-1]
  }
}
```

Good question! The current logic correctly handles the presence or absence of a grouping variable by checking the structure of the formula (form[[3]]). Let's analyze how it behaves in both cases and adjust if necessary.

Case 1: Formula with Grouping Variable (y ~ x1 + x2 | g)

In this case: • The RHS of the formula (form[[3]]) has a structure where form[[3]][[1]] is "|". • This logic works because the formula structure explicitly includes the grouping operator |.

Example: form <- y ~ x1 + x2 | g form[[3]][[1]] → "|" (indicates the grouping operator is present). • The logic splits the RHS into independent variables (form[[3]][[2]]) and the grouping variable (form[[3]][[3]]).

`Case 2: Formula without Grouping Variable (y ~ x1 + x2)`

In this case: • The RHS of the formula (form[[3]]) is x1 + x2, and form[[3]][[1]] is "+". • The current logic (as.character(form[[3]][[1]]) == "|") will fail because form[[3]][[1]] is "+", not "|". • Problem: The function will incorrectly assume that form[[3]][[1]] exists and attempt to access it, leading to an error when the RHS has no grouping operator. Fixing the Logic

To address this, we must first check if the formula has a valid structure for grouping. Specifically: 1. Check if the RHS (form[[3]]) has multiple components: • If form[[3]][[1]] exists, it can be checked for "|". • If not, assume there is no grouping variable. 2. Updated Logic: • If length(form[[3]]) > 1 and as.character(form[[3]][[1]]) == "|", handle grouping. • Otherwise, treat it as a formula without grouping.

```
frm <- function(form, data) {
    # Validate inputs
    if (!inherits(form, "formula")) {
        stop("Input 'form' must be a formula.")
    }
    if (missing(data) || !is.data.frame(data)) {
        stop("Input 'data' must be a data.frame.")
    }
```

```
    # Extract all variable names from the formula
    vars <- all.vars(form)

    # Ensure all variables in the formula exist in the data
    missing_vars <- setdiff(vars, colnames(data))
    if (length(missing_vars) > 0) {
        stop("The following variables are not found in the data: ", paste(missing_vars, collaps
    }

    # Extract the dependent variable
    y_var <- vars[1]

    # Check for grouping operator
    g_var <- NULL
    if (length(form[[3]]) > 1 && as.character(form[[3]][[1]]) == "|") {
        # Extract independent and grouping variables
        x_vars <- all.vars(form[[3]][[2]])
        g_var <- all.vars(form[[3]][[3]])
    } else {
        # No grouping variable, extract independent variables
        x_vars <- vars[-1]
    }

    # Subset the data for the independent variables
    independent_data <- data[c(x_vars, g_var)]

    # Subset the data for the dependent variable
    dependent_data <- data.frame(dependent = data[[y_var]], group = if (!is.null(g_var)) data[

    if (is.null(g_var)) {
        # If no grouping variable, return independent and dependent data.frames
        return(list(
            independent = independent_data,
            dependent = data.frame(dependent = data[[y_var]])
        ))
    } else {
        # Ensure the grouping variable is unique and valid
        grouping_var <- g_var[1]  # Assume single grouping variable
        if (!grouping_var %in% colnames(data)) {
            stop("Grouping variable not found in the data.")
        }

        # Split independent and dependent variables by the grouping variable
        split_independent <- split(independent_data[, x_vars, drop = FALSE], independent_data[
        split_dependent <- split(dependent_data[, "dependent", drop = FALSE], dependent_data$gr

        return(list(
```

```
            independent = split_independent,
            dependent = split_dependent
        ))
    }
}
```

## 2.1 References:

"the formula is used to specify the symbolic model as well as generating the intended design matrix" The R Formula Method: The Good Parts · R Views

" You're probably familiar with formulas from linear models (e.g. lm(mpg ~ displ, data = mtcars)) but formulas are more than just a tool for modelling: they are a general way of capturing an unevaluated expression"

"because a formula captures two things:

```
An unevaluated expression.
The context (environment) in which the expression was created.
```

~ is a single character that allows you to say: "I want to capture the meaning of this code, without evaluating it right away". For that reason, the formula can be thought of as a "quoting" operator."

Non-standard evaluation

Advanced Programming and Non-Standard Evaluation with dplyr | by Ryan Boyer | Shipt Tech

## 2.2 Prerequisites

In development

## 2.3 Step-by-Step Implementation

In development

## 2.4 Key Takeaways

In development

## 2.5 Further Reading

In development