The **phylo4** S4 classes and methods

Ben Bolker & Peter Cowan

October 27, 2010

**Contents**

1 Introduction 1

2 Package overview 2

3 Using the S4 help system 3

4 Trees without data 3

5 Trees with data 6

6 Subsetting 8

7 Tree-walking 8

8 multiPhylo classes 8

9 Examples 9

9.1 Constructing a Brownian motion trait simulator 9

9.1.1 the easy way 9

9.1.2 The hard way 10

A Definitions/slots 11

A.1 phylo4 11

A.2 phylo4d 12

A.3 multiphylo4 12

1 Introduction

This document describes the new **phylo4** S4 classes and methods, which are intended to provide a unifying standard for the representation of phylogenetic trees and comparative data in R. The **phylobase** package was developed to help both end users and package developers by providing a common suite of tools likely to be shared by all packages designed for phylogenetic analysis, facilities for data and tree manipulation, and standardization of formats.

This standardization will benefit **end-users** by making it easier to move data and compare analyses across packages, and to keep comparative data synchronized with phylogenetic trees. Users will also benefit from a repository of functions for tree manipulation, for example tools
for including or excluding subtrees (and associated phenotypic data) or improved tree and data plotting facilities. phylobase will benefit developers by freeing them to put their programming effort into developing new methods rather than into re-coding base tools. We (the phylobase developers) hope phylobase will also facilitate code validation by providing a repository for benchmark tests, and more generally that it will help catalyze community development of comparative methods in R.

A more abstract motivation for developing phylobase was to improve data checking and abstraction of the tree data formats. phylobase can check that data and trees are associated in the proper fashion, and protects users and developers from accidently reordering one, but not the other. It also seeks to abstract the data format so that commonly used information (for example, branch length information or the ancestor of a particular node) can be accessed without knowledge of the underlying data structure (i.e., whether the tree is stored as a matrix, or a list, or a parenthesis-based format). This is achieved through generic phylobase functions which retrieve the relevant information from the data structures. The benefits of such abstraction are multiple: (1) easier access to the relevant information via a simple function call (this frees both users and developers from learning details of complex data structures), (2) freedom to optimize data structures in the future without breaking code. Having the generic functions in place to “translate” between the data structures and the rest of the program code allows program and data structure development to proceed somewhat independently. The alternative is code written for specific data structures, in which modifications to the data structure requires rewriting the entire package code (often exacting too high a price, which results in the persistence of less-optimal data structures). (3) providing broader access to the range of tools in phylobase. Developers of specific packages can use these new tools based on S4 objects without knowing the details of S4 programming.

The base phylo4 class is modeled on the the phylo class in ape. phylo4d and multiphylo4 extend the phylo4 class to include data or multiple trees respectively. In addition to describing the classes and methods, this vignette gives examples of how they might be used.

## 2 Package overview

The phylobase package currently implements the following functions and data structures:

- Data structures for storing a single tree and multiple trees: phylo4 and multiPhylo4?
- A data structure for storing a tree with associated tip and node data: phylo4d
- A data structure for storing multiple trees with one set of tip data: multiPhylo4d
- Functions for reading nexus files into the above data structures
- Functions for converting between the above data structures and ape phylo objects as well as ade4 phylog objects
- Functions for editing trees and data (i.e., subsetting and replacing)
- Functions for plotting trees and trees with data
3 Using the S4 help system

The S4 help system works similarly to the S3 help system with some small differences relating to how S4 methods are written. The plot() function is a good example. When we type ?plot we are provided the help for the default plotting function which expects x and y. R also provides a way to smartly dispatch the right type of plotting function. In the case of an ape phylo object (a S3 class object) R evaluates the class of the object and finds the correct functions, so the following works correctly.

```r
> library(ape)
> set.seed(1)  ## set random-number seed
> rand_tree <- rcoal(10)  ## Make a random tree with 10 tips
> plot(rand_tree)
```

However, typing ?plot still takes us to the default plot help. We have to type ?plot.phylo to find what we are looking for. This is because S3 generics are simply functions with a dot and the class name added.

The S4 generic system is too complicated to describe here, but doesn’t include the same dot notation. As a result ?plot.phylo4 doesn’t work, R still finds the right plotting function.

```r
> library(phylobase)
> rand_p4_tree <- as(rand_tree, "phylo4")
> plot(rand_p4_tree)
```

All fine and good, but how do we find out about all the great features of the phylobase plotting function? R has two nifty ways to find it, the first is to simply put a question mark in front of the whole call:

```r
> `?` (plot(rand_p4_tree))
```

R looks at the class of the rand_p4_tree object and takes us to the correct help file (note: this only works with S4 objects). The second ways is handy if you already know the class of your object, or want to compare to generics for different classes:

```r
> `?` (method, plot("phylo4"))
```

More information about how S4 documentation works can be found in the methods package, by running the following command.

```r
> help("Documentation", package = "methods")
```

4 Trees without data

You can start with a tree — an object of class phylo from the ape package (e.g., read in using the read.tree() or read.nexus() functions), and convert it to a phylo4 object.

For example, load the raw Geospiza data:

```r
> library(phylobase)
> data(geospiza_raw)
> ## what does it contain?
> names(geospiza_raw)
```

[1] "tree" "data"
Convert the S3 tree to a S4 phylo4 object using the `as()` function:

```r
> (g1 <- as(geospiza_raw$tree, "phylo4"))
```

<table>
<thead>
<tr>
<th>label</th>
<th>node</th>
<th>ancestor</th>
</tr>
</thead>
<tbody>
<tr>
<td>fuliginosa</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>fortis</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>magnirostris</td>
<td>3</td>
<td>23</td>
</tr>
<tr>
<td>conirostris</td>
<td>4</td>
<td>22</td>
</tr>
<tr>
<td>scandens</td>
<td>5</td>
<td>21</td>
</tr>
<tr>
<td>difficilis</td>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td>pallida</td>
<td>7</td>
<td>25</td>
</tr>
<tr>
<td>parvulus</td>
<td>8</td>
<td>27</td>
</tr>
<tr>
<td>psittacula</td>
<td>9</td>
<td>27</td>
</tr>
<tr>
<td>pauper</td>
<td>10</td>
<td>26</td>
</tr>
<tr>
<td>Platyspiza</td>
<td>11</td>
<td>18</td>
</tr>
<tr>
<td>fusca</td>
<td>12</td>
<td>17</td>
</tr>
<tr>
<td>Pinaroloxias</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>olivacea</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>22</td>
<td>21</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>25</td>
<td>19</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>26</td>
<td>25</td>
</tr>
<tr>
<td>&lt;NA&gt;</td>
<td>27</td>
<td>26</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>edge.length</th>
<th>node.type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05500</td>
<td>tip</td>
</tr>
<tr>
<td>0.05500</td>
<td>tip</td>
</tr>
<tr>
<td>0.11000</td>
<td>tip</td>
</tr>
<tr>
<td>0.18333</td>
<td>tip</td>
</tr>
<tr>
<td>0.19250</td>
<td>tip</td>
</tr>
<tr>
<td>0.22800</td>
<td>tip</td>
</tr>
<tr>
<td>0.08667</td>
<td>tip</td>
</tr>
<tr>
<td>0.02000</td>
<td>tip</td>
</tr>
<tr>
<td>0.02000</td>
<td>tip</td>
</tr>
<tr>
<td>0.03500</td>
<td>tip</td>
</tr>
<tr>
<td>0.46550</td>
<td>tip</td>
</tr>
<tr>
<td>0.53409</td>
<td>tip</td>
</tr>
<tr>
<td>0.58333</td>
<td>tip</td>
</tr>
<tr>
<td>0.88077</td>
<td>tip</td>
</tr>
<tr>
<td>NA</td>
<td>root</td>
</tr>
<tr>
<td>0.29744</td>
<td>internal</td>
</tr>
<tr>
<td>0.04924</td>
<td>internal</td>
</tr>
<tr>
<td>0.06859</td>
<td>internal</td>
</tr>
<tr>
<td>0.13404</td>
<td>internal</td>
</tr>
<tr>
<td>0.10346</td>
<td>internal</td>
</tr>
<tr>
<td>0.03550</td>
<td>internal</td>
</tr>
<tr>
<td>0.00917</td>
<td>internal</td>
</tr>
<tr>
<td>0.07333</td>
<td>internal</td>
</tr>
<tr>
<td>0.08550</td>
<td>internal</td>
</tr>
<tr>
<td>0.24479</td>
<td>internal</td>
</tr>
<tr>
<td>0.05167</td>
<td>internal</td>
</tr>
<tr>
<td>0.01500</td>
<td>internal</td>
</tr>
</tbody>
</table>

The (internal) nodes appear with labels `<NA>` because they are not defined:

```r
> nodeLabels(g1)
```

   15 16 17 18 19 20 21 22 23 24 25 26 27
NA NA NA NA NA NA NA NA NA NA NA NA NA

You can also retrieve the node labels with `labels(g1,"internal")`.

A simple way to assign the node numbers as labels (useful for various checks) is

```r
> nodeLabels(g1) <- paste("N", nodeId(g1, + "internal"), sep = "")
> head(g1, 5)
```
The `summary` method gives a little extra information, including information on the distribution of branch lengths:

```r
> summary(g1)

Phylogenetic tree : g1
Number of tips : 14
Number of nodes : 13
Branch lengths:
  mean : 0.1764008
  variance : 0.04624379
  distribution :
    Min. 1st Qu.  Median    Mean 3rd Qu.     Max. 
  0.00917 0.04985 0.08000 0.17640 0.21910 0.88080

Print tip labels:
> tipLabels(g1)

 1 2
"fuliginosa" "fortis"
 3 4
"magnirostris" "conirostris"
 5 6
"scandens" "difficilis"
 7 8
"pallida" "parvulus"
 9 10
"psittacula" "pauper"
11 12
"Platyspiza" "fusca"
13 14
"Pinaroloxias" "olivacea"

(lables(g1,"tip"); would also work.)

Print node numbers (in edge matrix order):
> nodeId(g1, type = "all")

[1]  1  2  3  4  5  6  7  8  9 10 11 12
[26] 25 26 27

Print edge labels (also empty in this case — therefore all NA):
> edgeLabels(g1)

 15-16 16-17 17-18 18-19 19-20 20-21
 NA NA NA NA NA NA
21-22 22-23 23-24 24-1 24-2 23-3
 NA NA NA NA NA NA
22-4 21-5 0-15 20-6 19-25 25-7
 NA NA NA NA NA NA
25-26 26-27 27-8 27-9 26-10 18-11
 NA NA NA NA NA NA
17-12 16-13 15-14
 NA NA NA
```
Is it rooted?
> isRooted(g1)
[1] TRUE

Which node is the root?
> rootNode(g1)
[1] 15

Does it contain any polytomies?
> hasPoly(g1)
[1] FALSE

Does it have information on branch lengths?
> hasEdgeLength(g1)
[1] TRUE

You can modify labels and other aspects of the tree — for example, to convert all the labels to lower case:
> tipLabels(g1) <- tolower(tipLabels(g1))

You could also modify selected labels, e.g. to modify the labels in positions 11 and 13 (which happen to be the only labels with uppercase letters):
> tipLabels(g1)[c(11, 13)] <- c("platyspiza",
+ "pinaroloxias")

5 Trees with data

The phylo4d class matches trees with data, or combines them with a data frame to make a phylo4d (tree-with-data) object.

Now we’ll take the Geospiza data from geospiza_raw$data and merge it with the tree. However, since G. olivacea is included in the tree but not in the data set, we will initially run into some trouble:
> g2 <- phylo4d(g1, geospiza_raw$data)

Error in formatData(phy = x, dt = tip.data, type = "tip", ...) :
  The following nodes are not found in the dataset: platyspiza, pinaroloxias, olivacea

We have two problems — the first is that we forgot to lowercase the labels on the data to match the tip labels:
> gdata <- geospiza_raw$data
> row.names(gdata) <- tolower(row.names(gdata))
To deal with the second problem (missing data for *G. olivacea*), we have a few choices. The easiest is to use `missing.data="warn"` to allow R to create the new object with a warning (you can also use `missing.data="OK"` to proceed without warnings):

```r
> g2 <- phylo4d(g1, gdata, missing.data="warn")
```

Another way to deal with this would be to use `prune()` to drop the offending tip from the tree first:

```r
> g1B <- prune(g1, "olivacea")
> phylo4d(g1B, gdata)
```

You can summarize the new object:

```r
> summary(g2)
```

**Phylogenetic tree:** `as(x, "phylo4")`
- **Number of tips:** 14
- **Number of nodes:** 13
- **Branch lengths:**
  - **mean:** 0.1764008
  - **variance:** 0.04624379
  - **distribution:**
    - Min. 1st Qu. Median Mean 3rd Qu. Max.
    - 0.00917 0.04985 0.08000 0.17640 0.21910 0.88080

**Comparative data:**
**Tips:** data.frame with 14 taxa and 5 variable(s)

<table>
<thead>
<tr>
<th>wingL</th>
<th>tarsusL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. : 3.975</td>
<td>Min. : 2.807</td>
</tr>
<tr>
<td>1st Qu.: 4.189</td>
<td>1st Qu.: 2.929</td>
</tr>
<tr>
<td>Median : 4.235</td>
<td>Median : 2.980</td>
</tr>
<tr>
<td>Mean : 4.236</td>
<td>Mean : 2.991</td>
</tr>
<tr>
<td>3rd Qu.: 4.265</td>
<td>3rd Qu.: 3.039</td>
</tr>
<tr>
<td>Max. : 4.420</td>
<td>Max. : 3.271</td>
</tr>
<tr>
<td>NA's : 1.000</td>
<td>NA's : 1.000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>culmenL</th>
<th>beakD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. : 1.294</td>
<td>Min. : 1.191</td>
</tr>
<tr>
<td>1st Qu.: 1.207</td>
<td>1st Qu.: 1.191</td>
</tr>
<tr>
<td>Median : 1.231</td>
<td>Median : 1.207</td>
</tr>
<tr>
<td>Mean : 1.233</td>
<td>Mean : 1.208</td>
</tr>
<tr>
<td>3rd Qu.: 1.233</td>
<td>3rd Qu.: 1.234</td>
</tr>
<tr>
<td>Max. : 1.275</td>
<td>Max. : 1.284</td>
</tr>
<tr>
<td>NA's : 1.000</td>
<td>NA's : 1.000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>gonysW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. : 1.401</td>
</tr>
<tr>
<td>1st Qu.: 1.845</td>
</tr>
<tr>
<td>Median : 1.962</td>
</tr>
<tr>
<td>Mean : 2.014</td>
</tr>
<tr>
<td>3rd Qu.: 2.222</td>
</tr>
<tr>
<td>Max. : 2.676</td>
</tr>
<tr>
<td>NA's : 1.000</td>
</tr>
</tbody>
</table>

**Nodes:** data.frame with 13 internal nodes and 5 variables

<table>
<thead>
<tr>
<th>wingL</th>
<th>tarsusL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. : NA</td>
<td>Min. : NA</td>
</tr>
<tr>
<td>1st Qu.: NA</td>
<td>1st Qu.: NA</td>
</tr>
<tr>
<td>Median : NA</td>
<td>Median : NA</td>
</tr>
<tr>
<td>Mean : NaN</td>
<td>Mean : NaN</td>
</tr>
<tr>
<td>3rd Qu.: NA</td>
<td>3rd Qu.: NA</td>
</tr>
<tr>
<td>Max. : NA</td>
<td>Max. : NA</td>
</tr>
<tr>
<td>NA's : 13</td>
<td>NA's : 13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>culmenL</th>
<th>beakD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. : NA</td>
<td>Min. : NA</td>
</tr>
<tr>
<td>1st Qu.: NA</td>
<td>1st Qu.: NA</td>
</tr>
<tr>
<td>Median : NA</td>
<td>Median : NA</td>
</tr>
<tr>
<td>Mean : NaN</td>
<td>Mean : NaN</td>
</tr>
<tr>
<td>3rd Qu.: NA</td>
<td>3rd Qu.: NA</td>
</tr>
</tbody>
</table>
Or use `tdata()` to extract the data (i.e., `tdata(g2)`). By default, `tdata()` will retrieve tip data, but you can also get internal node data only (`tdata(tree, "internal")`) or — if the tip and node data have the same format — all the data combined (`tdata(tree, "allnode")`).

If you want to plot the data (e.g. for checking the input), `plot(tdata(g2))` will create the default plot for the data — in this case, since it is a data frame [this may change in future versions but should remain transparent] this will be a `pairs` plot of the data.

6 Subsetting

The `subset` command offers a variety of ways of extracting portions of a `phylo4` or `phylo4d` tree, keeping any tip/node data consistent.

- `tips.include` give a vector of tips (names or numbers) to retain
- `tips.exclude` give a vector of tips (names or numbers) to drop
- `mrca` give a vector of node or tip names or numbers; extract the clade containing these taxa
- `node.subtree` give a node (name or number); extract the subtree starting from this node

Different ways to extract the *fuliginosa-scandens* clade:

```r
> subset(g2, tips.include = c("fuliginosa", "fortis", "magnirostris", "conirostris", "scandens"))
> subset(g2, node.subtree = 21)
> subset(g2, mrca = c("scandens", "fortis"))
```

One could drop the clade by doing

```r
> try(subset(g2, tips.exclude = c("fuliginosa", "fortis", "magnirostris", "conirostris", "scandens")), silent = TRUE)
> try(subset(g2, tips.exclude = names(descendants(g2, MRCA(g2, c("difficilis", "fortis"))))), silent = TRUE)
```

7 Tree-walking

`getnodes, children, parent, descendants, ancestors, siblings, MRCA` ...

generally take a `phylo4` object, a node (specified by number or name) and return a named vector of node numbers.

8 multiPhylo classes

Fix me!
9 Examples

9.1 Constructing a Brownian motion trait simulator

This section will describe two (?) ways of constructing a simulator that generates trait values for extant species (tips) given a tree with branch lengths, assuming a model of Brownian motion.

9.1.1 the easy way

We can use `as(tree,"phylo4vcov")` to coerce the tree into a variance-covariance matrix form, and then use `mvtnorm` from the MASS package to generate a set of multivariate normally distributed values for the tips. (A benefit of this approach is that we can very quickly generate a very large number of replicates.) This example illustrates a common feature of working with phylobase — combining tools from several different packages to operate on phylogenetic trees with data.

We start with a randomly generated tree using `rcoal()` from ape to generate the tree topology and branch lengths:

```r
> set.seed(1001)
> tree <- as(rcoal(12), "phylo4")
```

Next we generate the phylogenetic variance-covariance matrix (by coercing the tree to a phylo4vcov object) and pick a single set of normally distributed traits (using MASS:mvtnorm to pick a multivariate normal deviate with a variance-covariance matrix that matches the structure of the tree).

```r
> vmat <- as(tree, "phylo4vcov")
> vmat <- cov2cor(vmat)
> library(MASS)
> trvec <- mvrnorm(1, mu = rep(0, + 12), Sigma = vmat)
```

The last step (easy) is to convert the phylo4vcov object back to a phylo4d object:

```r
> treed <- phylo4d(tree, tip.data = as.data.frame(trvec))
> plot(treed)
```
9.1.2 The hard way

```r
nodeLabels(tree) <- as.character(nodeId(tree, "internal"))
# ordering will make sure that we have ancestor value
# defined before descendant
> tree <- reorder(tree, "preorder")
edgemat <- edges(tree)
# set aside space for values
nodevals <- numeric(nrow(edgemat))
# label data in edge matrix order
names(nodevals) <- labels(tree, "all")[nodeId(tree, "all")]
# variance is proportional to edge length; drop first
# element of edge length, which is NA
dvals <- rnorm(nrow(edgemat) - 1, sd=edgeLength(tree)[-1]^2)
# indexing: ind[node number] gives position in edge matrix
> ind <- order(nodeId(tree, "all"))
> for (i in 2:nrow(edgemat)) {
+ # value of ancestor node plus change
+ nodevals[i] <- nodevals[ind[edgemat[i, 1]]] + dvals[i - 1]
+ }
nodevals <- data.frame(nodevals)
treed2 <- phylo4d(tree, all.data=nodevals)
```
A Definitions/slots

This section details the internal structure of the phylo4, multiphylo4, phylo4d, and multiphylo4d classes. The basic building blocks of these classes are the phylo4 object and a dataframe. The phylo4 tree format is largely similar to the one used by phylo class in the package ape.¹

We use “edge” for ancestor-descendant relationships in the phylogeny (sometimes called “branches”) and “edge lengths” for their lengths (“branch lengths”). Most generally, “nodes” are all species in the tree; species with descendants are “internal nodes” (we often refer to these just as “nodes”, meaning clear from context): “tips” are species with no descendants. The “root node” is the node with no ancestor (if one exists).

A.1 phylo4

Like phylo, the main components of the phylo4 class are:

- **edge**: a 2-column matrix of integers, with \(N\) rows for a rooted tree or \(N-1\) rows for an unrooted tree and column names *ancestor* and *descendant*. Each row contains information on one edge in the tree. See below for further constraints on the edge matrix.

- **edge.length**: numeric list of edge lengths (length \(N\) (rooted) or \(N-1\) (unrooted) or empty (length 0))

- **tip.label**: character vector of tip labels (required), with length=# of tips. Tip labels need not be unique, but data-tree matching with non-unique labels will cause an error.

- **node.label**: character vector of node labels, length=# of internal nodes or 0 (if empty). Node labels need not be unique, but data-tree matching with non-unique labels will cause an error.

- **order**: character: “preorder”, “postorder”, or “unknown” (default), describing the order of rows in the edge matrix. “pruningwise” and “cladewise” are accepted for compatibility with ape

The edge matrix must not contain NAs, with the exception of the root node, which has an NA for *ancestor*. phylobase does not enforce an order on the rows of the edge matrix, but it stores information on the current ordering in the @order slot — current allowable values are “unknown” (the default), “preorder” (equivalent to “cladewise” in ape) or “postorder” ².

The basic criteria for the edge matrix are similar to those of ape, as documented it’s tree specification³. This is a modified version of those rules, for a tree with \(n\) tips and \(m\) internal nodes:

- Tips (no descendants) are coded 1, \ldots, \(n\), and internal nodes (\(\geq 1\) descendant) are coded \(n+1, \ldots, n+m\) \((n+1\) is the root). Both series are numbered with no gaps.

- The first (ancestor) column has only values \(> n\) (internal nodes): thus, values \(\leq n\) (tips) appear only in the second (descendant) column.

---

¹[http://ape.mpl.ird.fr/](http://ape.mpl.ird.fr/)
²see [http://en.wikipedia.org/wiki/Tree_traversal](http://en.wikipedia.org/wiki/Tree_traversal) for more information on orderings. (ape’s “pruning-wise” is “bottom-up” ordering).
• all internal nodes [not including the root] must appear in the first (ancestor) column at least once [unlike ape, which nominally requires each internal node to have at least two descendants (although it doesn’t absolutely prohibit them and has a collapse.singles function to get rid of them), phylobase does allow these “singleton nodes” and has a method hasSingle for detecting them]. Singleton nodes can be useful as a way of representing changes along a lineage; they are used this way in the ouch package.

• the number of occurrences of a node in the first column is related to the nature of the node: once if it is a singleton, twice if it is dichotomous (i.e., of degree 3 [counting ancestor as well as descendants]), three times if it is trichotomous (degree 4), and so on.

phylobase does not technically prohibit reticulations (nodes or tips that appear more than once in the descendant column), but they will probably break most of the methods. Disconnected trees, cycles, and other exotica are not tested for, but will certainly break the methods.

We have defined basic methods for phylo4: show, print, and a variety of accessor functions (see help files). summary does not seem to be terribly useful in the context of a “raw” tree, because there is not much to compute.

A.2 phylo4d

The phylo4d class extends phylo4 with data. Tip data, and (internal) node data are stored separately, but can be retrieved together or separately with tdata(x,"tip"), tdata(x,"internal") or tdata(x,"all"). There is no separate slot for edge data, but these can be stored as node data associated with the descendant node.

A.3 multiphylo4