

Inapplicable data

*Martin Brazeau (m.brazeau@imperial.ac.uk), Thomas Guillerme (guillert@tcd.ie) and
Martin Smith (martin.smith@durham.ac.uk)*

2018-06-29

Contents

Inapplicable data in a parsimony setting	5
1 The Fitch algorithm	7
1.1 Downpass	10
1.2 Uppass	10
1.3 Resolving ambiguous resolutions	12
2 Problems with the Fitch algorithm	17
2.1 Red tails, blue tails	17
2.2 Reductive coding	18
2.3 Inapplicable as an extra state	20
2.4 A single multi-state character	22
2.5 Sankoff matrices	24
2.6 Why counting steps cannot work	26
2.7 Conclusion	29
3 A solution	31
3.1 Minimising homoplasy	31
3.2 Algorithmic implementation	38
3.3 Software implementation	48
4 Coding data	49
4.1 Multiple dependencies	49
4.2 Invariant characters can inform parsimony	52
4.3 Variable but ‘parsimony uninformative’ characters can inform parsimony	54
4.4 This may not be desirable in neomorphic characters	54
5 Coding ambiguity	61
5.1 Principal character ambiguous	61
5.2 Principal character known	62
5.3 Recommendation	65
6 Examples	67
6.1 Some caterpillars	67
6.2 Three equally suboptimal alternatives	67
6.3 A better caterpillar tree	70
6.4 De Laet’s caterpillars	70
Bibliography	73

Inapplicable data in a parsimony setting

This document (also available as an HTML webpage*) provides a detailed explanation of the algorithm for handling inapplicable data proposed by Brazeau et al. (2018).

We first discuss how the Fitch algorithm works and introduce the problems that it encounters in the face of inapplicable character states.

We then introduce our solution, a new algorithm, implemented in various software packages, and discuss its implications for the coding of characters and ambiguity.

We close with some example trees that demonstrate how our algorithm behaves in more complicated cases.

* rawgit.com/TGuillerme/Inapp/master/inst/gitbook/_book/

Chapter 1

The Fitch algorithm

This algorithm was proposed by Fitch (1971) and is implemented in phylogenetic software that employs maximum parsimony (Swofford, 2001; Goloboff and Catalano, 2016) and probabilistic methods (Ronquist et al., 2012; Stamatakis, 2014). The simple and elegant procedure entails going down the tree to count the number of transformations, then going back up the tree to finalise the ancestral state reconstructions.

The way the algorithm goes up and down depends on the software, but often employs a *traversal*: a recursive function that can visit all tips and nodes in a logical fashion. For example, consider the following tree:

A downpass traversal will first evaluate the first cherry (or pair of taxa) (A below), then save the results in **n1** and evaluate the second pair of tips/nodes (B), saving the results in **n2**, proceeding until all the nodes and tips have been visited.

An uppass traversal works identically but in the other direction, going from the nodes towards the tips.

In both traversals, the sequence in which nodes are visited (i.e. which cherry to pick first in a downpass traversal, or whether to continue left or right in an uppass traversal) is arbitrary, provided that all tips and nodes are eventually visited.

Now let's consider a more complex tree (((a, b), c), d), (e, (f, (g, h)))); with a binary character distributed (((1, 0), 0), 1), (1, (0, (0, 1))):

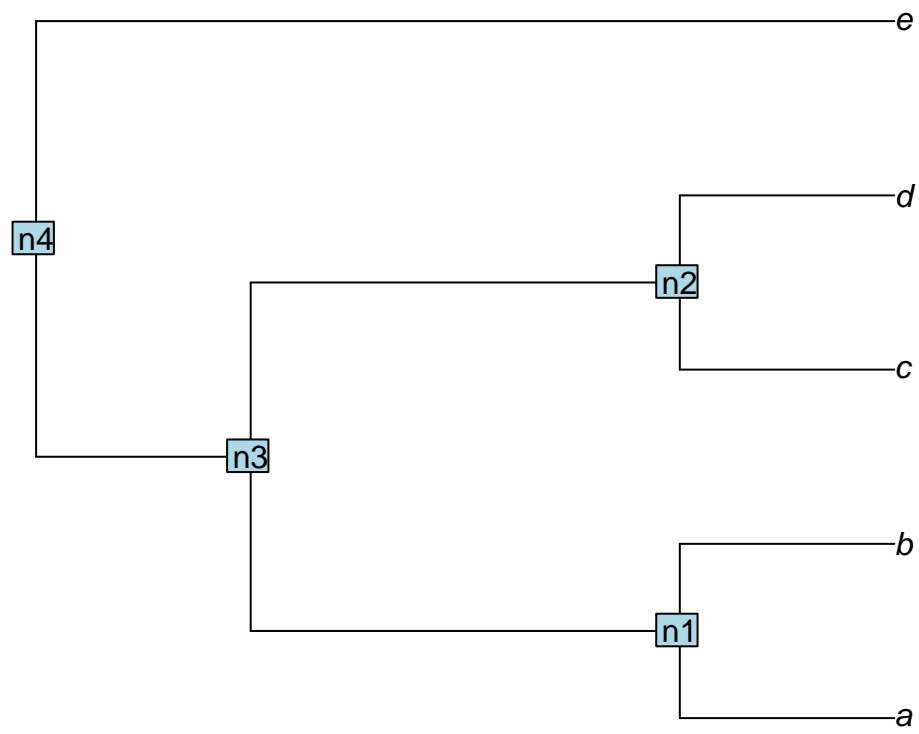


Figure 1.1: A five-tip tree

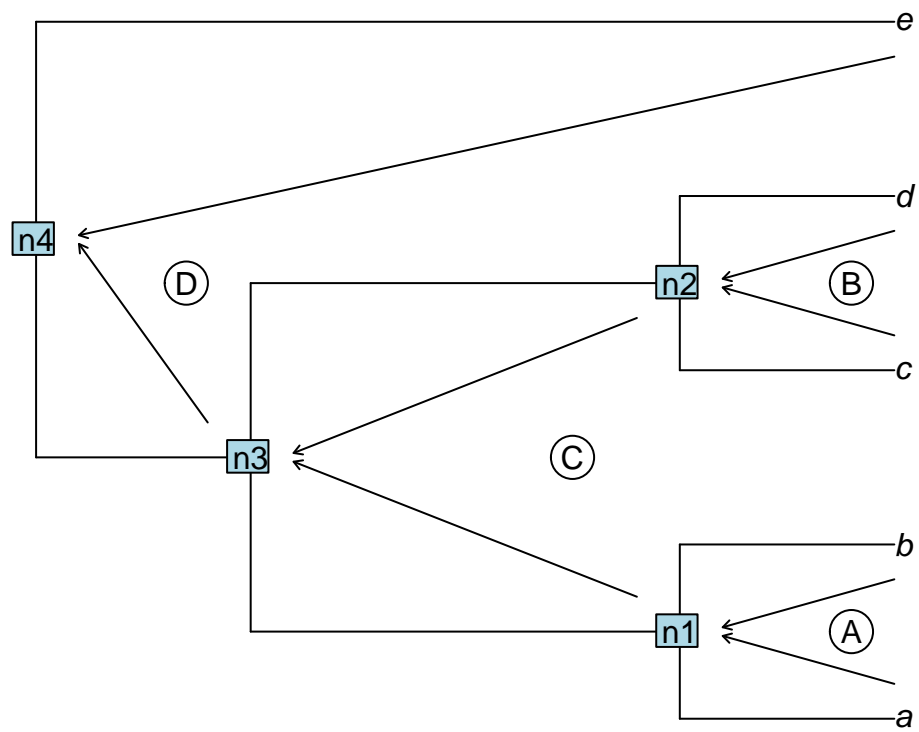


Figure 1.2: Downpass traversal

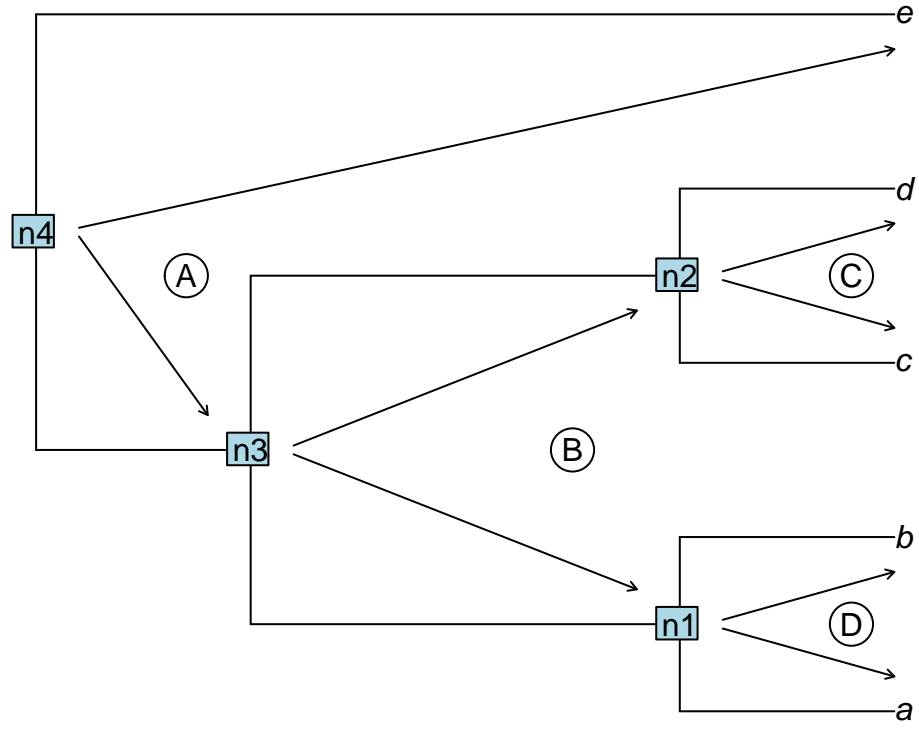
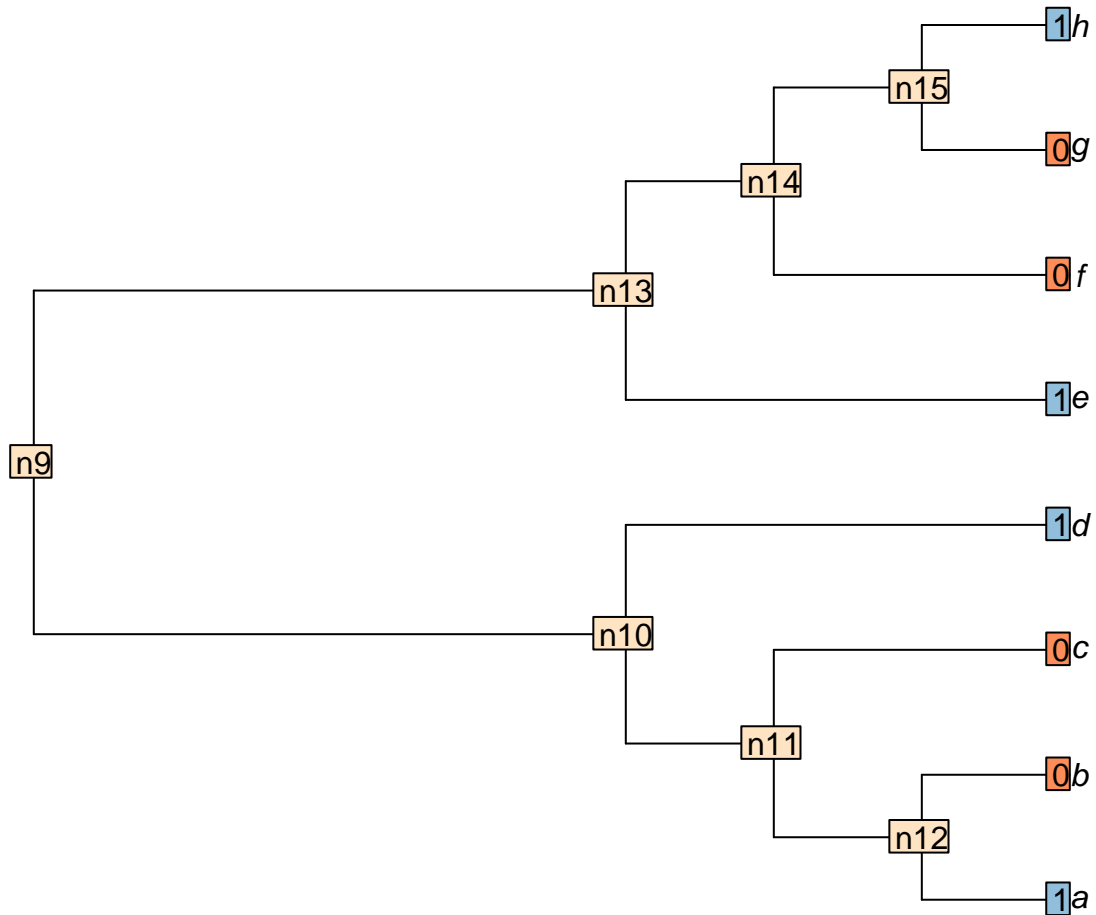


Figure 1.3: Uppass traversal



We can use the `Inapp` package to apply the Fitch algorithm for this character on this tree.

```
## Loading the Inapp package
library(Inapp)

## The tree
tree <- read.tree(text = "((((a, b), c), d), (e, (f, (g, h)))));")

## The character
character <- "10011001"

## Applying the Fitch algorithm
matrix <- apply.reconstruction(tree, character, method = "Fitch", passes = 2)
```

1.1 Downpass

The downpass is quite simple and follows these rules for the two possible cases in the traversal (Fitch, 1971):

1. If the two considered tips or nodes have *at least one state in common*, set the node to be these states in *common*.
2. Else, if there is *nothing in common* between the two tips or nodes, set the node to be the *union* of the two states.

For example, node `n15` is case 2 because there is *nothing in common* between the tips `h` and `g` (states 1 and 0 respectively). Node `n14` is case 1 because there is *at least one state in common* between the tip `f` and the node `n15` (state 0).

Important: when the case 2 is encountered, a transformation is implied in the descendants of the considered node. The score of the tree is incremented by +1.

In the following example, nodes that are in case 1 are in white and nodes in case 2, which imply a transformation (i.e. that adds to the tree score), are in green.

```
## Plotting the first downpass
plot.states.matrix(matrix, passes = 1, counts = 2, show.labels = c(1,2))
tiplabels(char, cex = 1, bg = Inapp::brewer[[2]][char + 1], adj = 1)
```

1.2 Uppass

The score of the tree is known after the downpass, but the states of some nodes might not yet be properly resolved. For example, parsimonious reconstructions exist that reconstruct node `n14` as state 1, as opposed to the 0 presently reconstructed. The present reconstruction seemingly indicates a change from state 1 to state 0 in an ancestor of `n14`, and a subsequent change from state 0 to state 1 in the ancestor of `h`, but an alternative reconstruction is equally parsimonious: `n13`, `n14` and `n15` may all have state 1, with a change from state 1 to state 0 in each of the two lineages leading to `f` and `g`.

The uppass traversal employs the following rules (Fitch, 1971):

1. If the current node and its ancestor have *all states in common*, the node is already resolved.
2. If there is *at least one state in common* between both left and right tips or nodes directly descended from the current node, resolve the node as being *the states in common* between its ancestor and both his descendants.
3. If there is *there are no states in common* between its descendants, resolve the node as being *the states in common* between the ancestor and the current node.

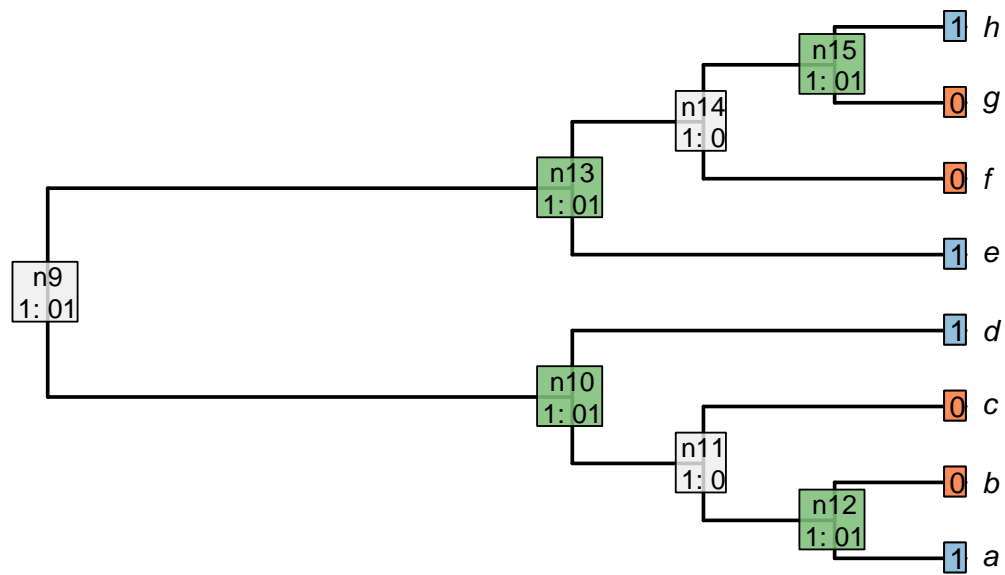


Figure 1.4: Node reconstructions after downpass

For example, node **n13** is already resolved (it has all its states in common with the ancestor - case 1). Node **n14** has not all its states in common with its ancestor but its two descendants (**n15** and **f**) have at least one state in common (0). This node is thus solved to be the states in common between both descendants (01) and its ancestor (01 as well).

```
## Plotting the first uppass
plot.states.matrix(matrix, passes = 1:2, counts = 2,
                   show.labels = c(1, 2), col.states=TRUE)
```

More complex cases can be studied in the Inapp App (running in your favourite web browser) by switching the **Reconstruction method** to **Normal Fitch**.

```
## Running the Inapp App
runInapp()
```

1.3 Resolving ambiguous resolutions

In certain cases, it is necessary to go further and discriminate between the equally-parsimonious reconstructions provided by the Fitch algorithm. A number of approaches have been proposed concerning which resolution of ambiguous nodes is preferable.

The two most familiar approaches to resolving ambiguous node are the Accelerated Transformation (ACCTRAN) and Delayed Transformation (DELTRAN) approaches (Farris et al., 1970; Swofford and Maddison, 1987).

The ACCTRAN approach reconstructs transformations as occurring as close to the root as possible; the DELTRAN, as far from the root as possible.

In this case, the ambiguous resolution of the root leaves two options for the latter:

If the states 0 and 1 represent states of a transformational character – whether an organism’s tail is red or blue, say – then there is no reason to prefer any of the equally-parsimonious reconstructions, as none implies any more homology than any other.

With neomorphic characters, however, state 0 stands for the absence of a character – for example, a tail – and state 1 its presence. On one view, a reconstruction that minimises the number of times that such a character evolves attributes more similarity to homology than an equally parsimonious reconstruction in which said character is gained multiple times independently.

1.3.1 Maximising homology

Neither ACCTRAN nor DELTRAN is guaranteed to maximise homology (Agnarsson and Miller, 2008). In this particular case, the DELTRAN reconstruction maximises homology. If the character denotes the presence or absence of a tail, then this reconstruction invokes the presence of a tail in the common ancestor of all taxa, meaning that the tails present in tips **a**, **d**, **e** and **h** are homologous with one another. The ACCTRAN reconstruction, in contrast, identifies a loss of a tail at nodes 11 and 14, with a tail evolving independently in tips **a** and **h**. Under this reconstruction, the tails of **a** and **h** are not homologous with each other, or with the tails of **d** and **e**. (The alternative DELTRAN approach, which could arguably be described as ACCTRAN instead, invokes four independent origins of the character and clearly does not maximise its homology.)

Where we wish to maximise homology, we modify the Fitch uppass such that any node whose final state reconstruction would be ambiguous is instead reconstructed as present when that node is encountered. This approach maximises homology in the problematic trees presented by Agnarsson & Miller [(2008); “A&M” below]:

This approach is also robust to missing entries:

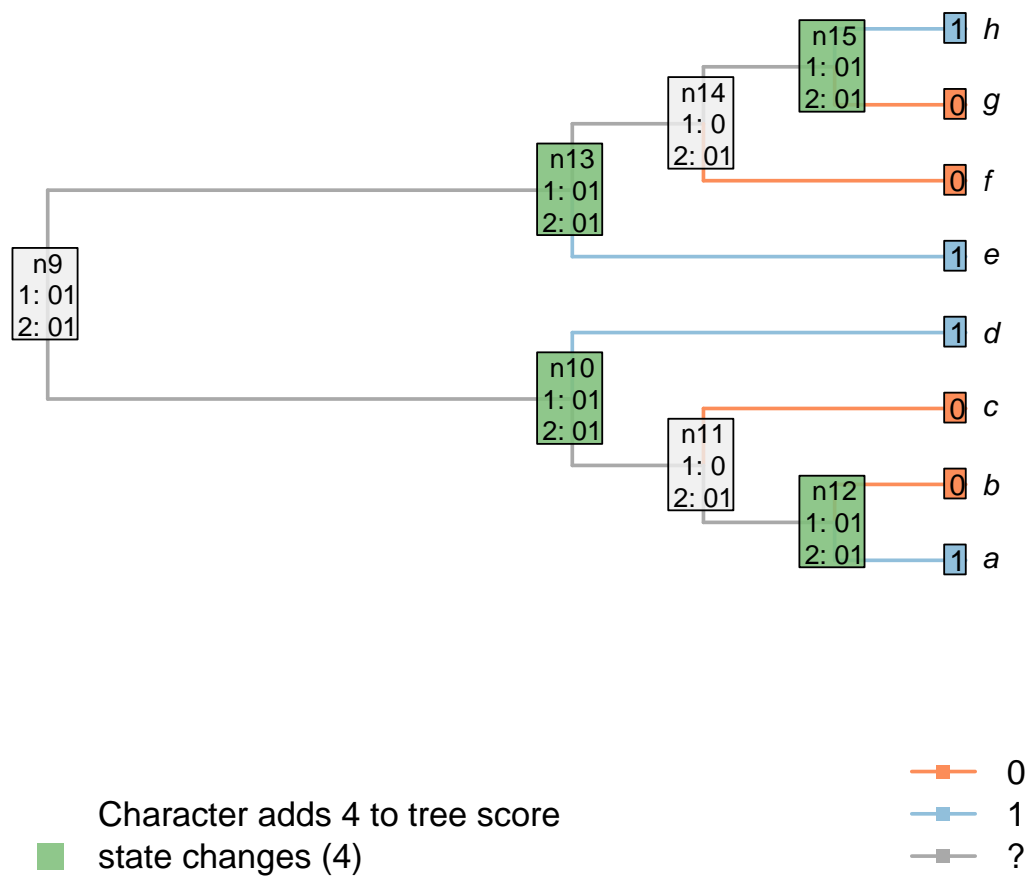


Figure 1.5: Node reconstructions after uppass

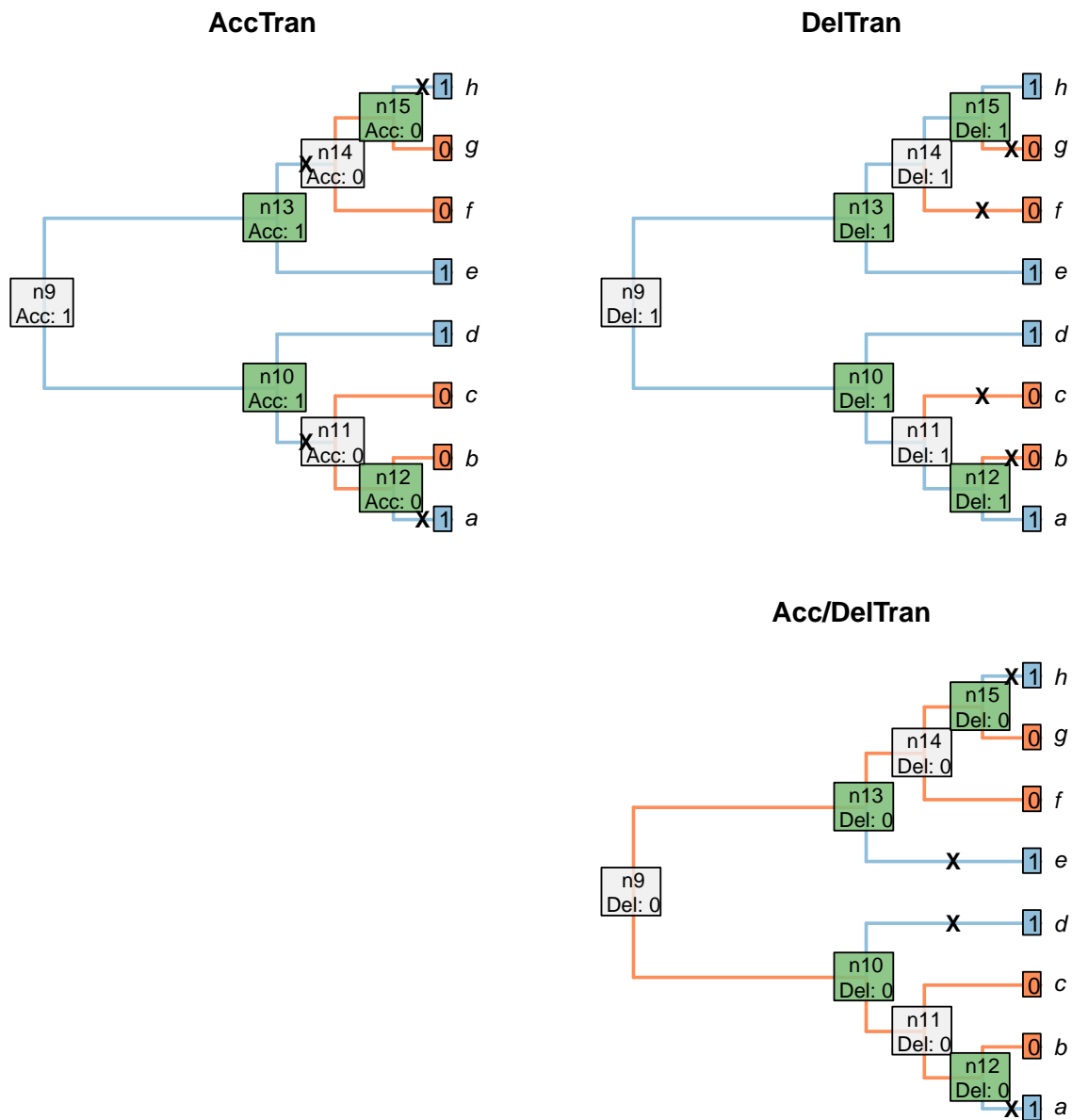
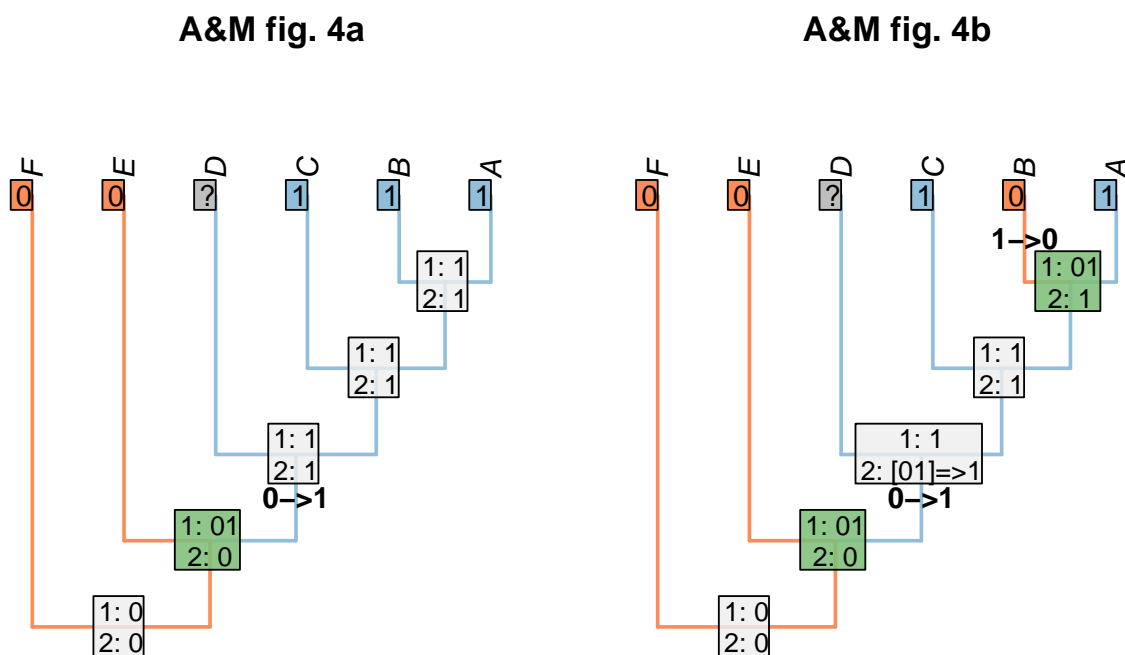
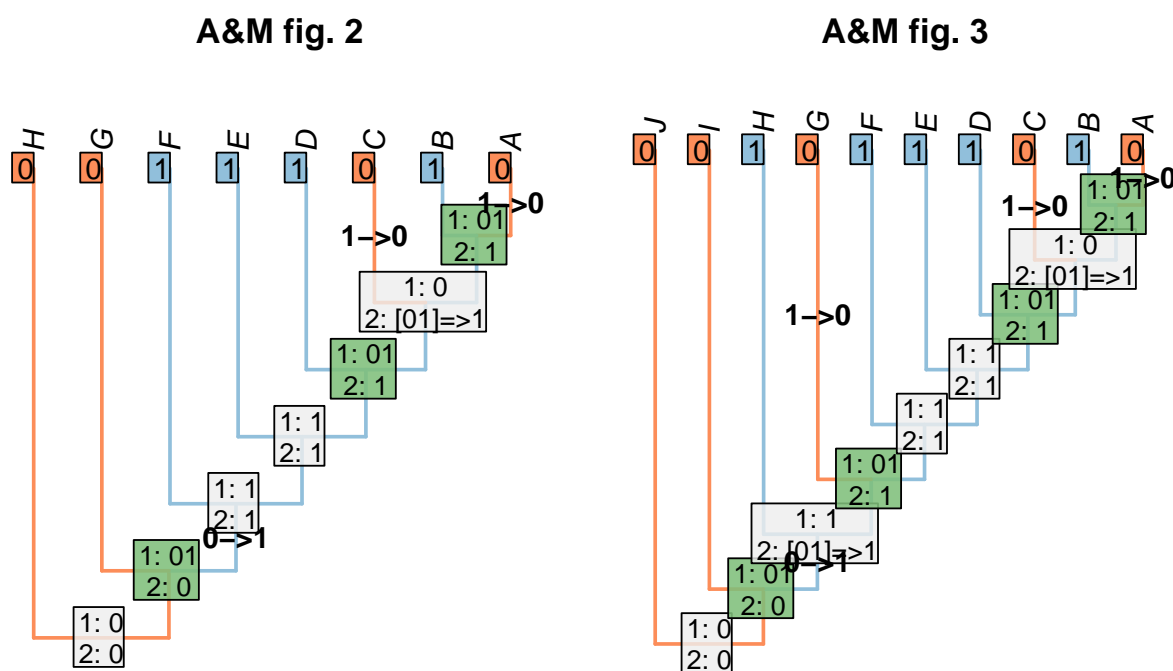


Figure 1.6: Node reconstructions after ACCTRAN or DELTRAN optimizations



Chapter 2

Problems with the Fitch algorithm

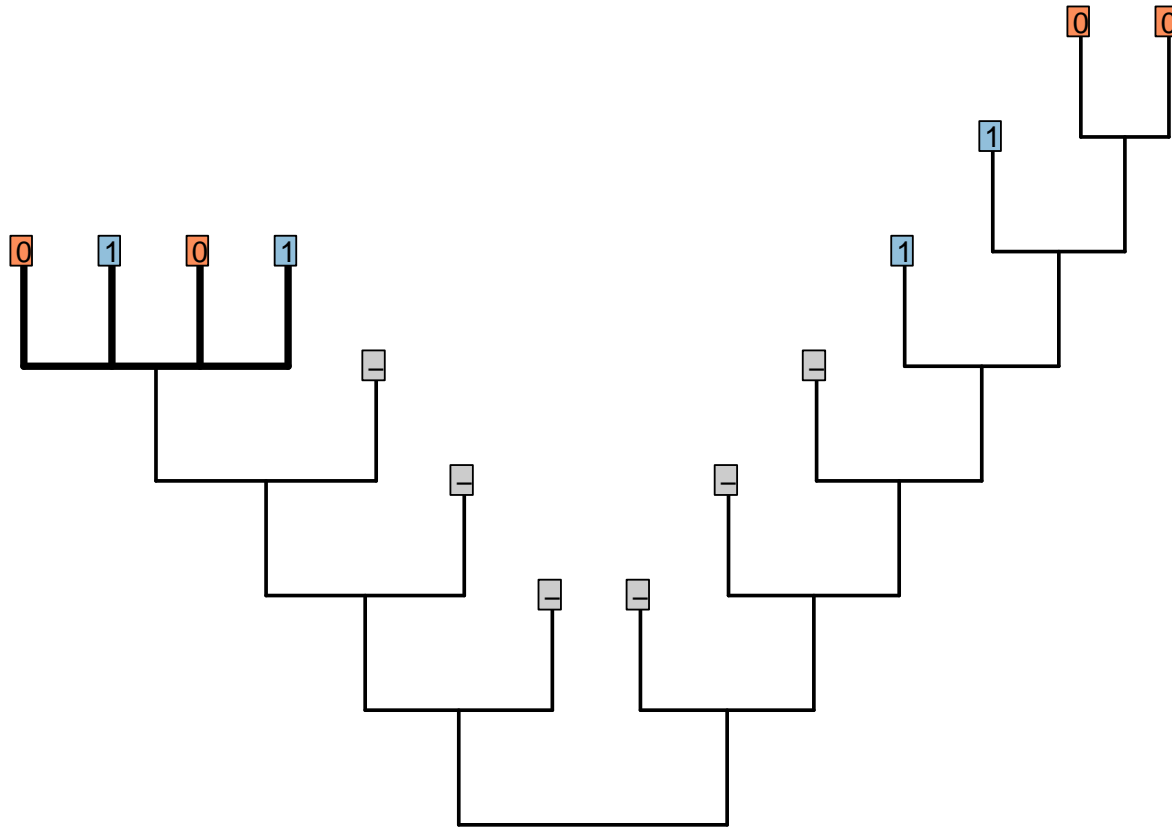
The Fitch algorithm (Fitch, 1971) counts changes in a character. It assumes that the character is applicable throughout the tree. This assumption does not lead to error if:

- The character is inapplicable in fewer than three tips; or
- In the trees being considered, applicable and inapplicable tokens occur in distinct regions of the tree (Maddison, 1993).

2.1 Red tails, blue tails

Maddison (1993) provided the following example to demonstrate the problem encountered by the Fitch algorithm when inapplicable characters were present.

Consider the following tree, each node of which is supported by a number of characters. Tail colour (illustrated; 0 = red, 1 = blue) has not yet been considered, but has the potential to resolve the polytomy on the left hand side (bold).



In the bold region, tail colour should group the red-tailed tips together, and the blue-tailed tips together, but does not establish whether the ancestor of the left-hand tail-bearing clade had a red or blue tail.

2.2 Reductive coding

Under reductive coding, the tail and its colour are described in two character statements:

Tail: (0), absent; (1), present.

Tail, colour: (0), red; (1), blue; (?), inapplicable.

Consider the following two trees, each of which receives a score of two for the first character (presence of tail). The score of the second character (tail colour) is not as desired.

The Fitch algorithm will prefer trees in which the left-hand tail-bearing clade has a blue tail, simply because the right-hand tail-bearing clade ancestrally did.

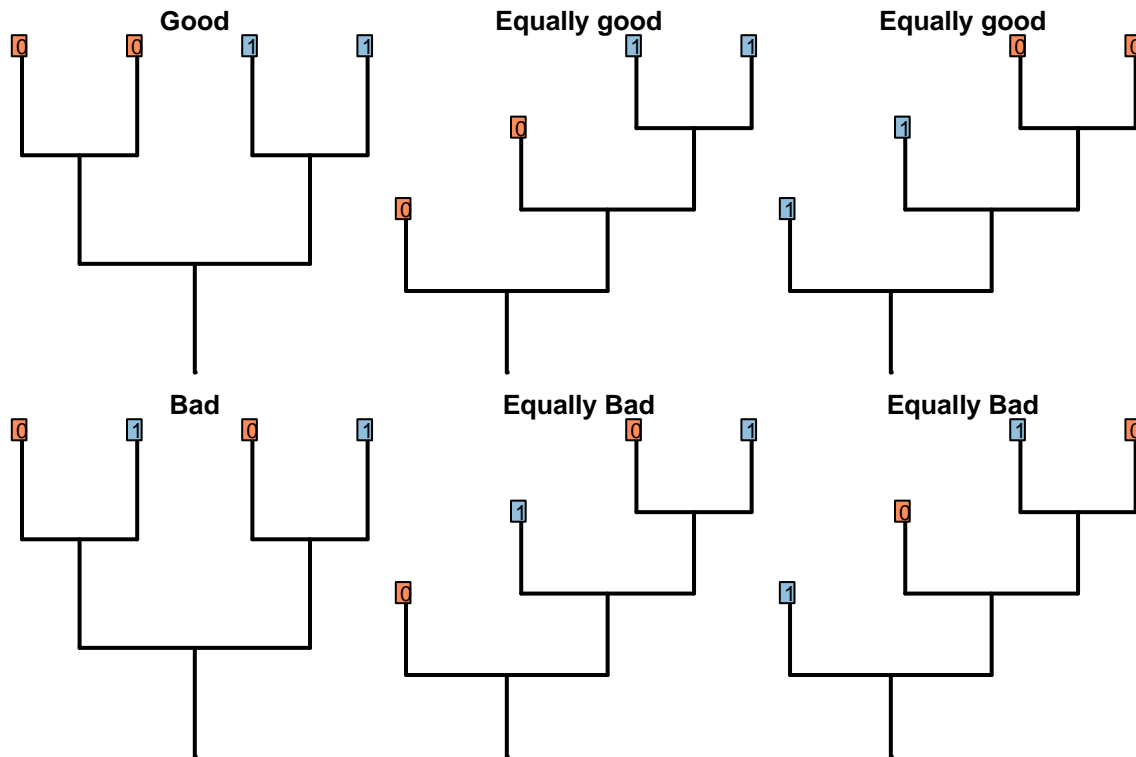
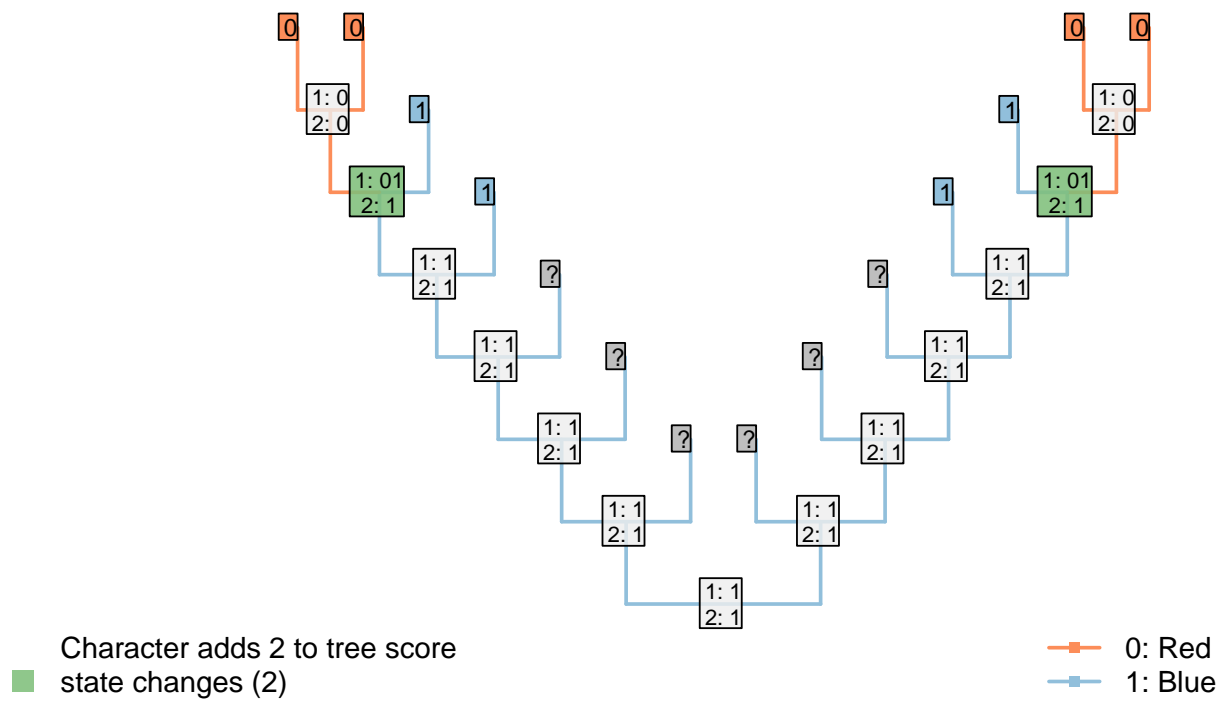


Figure 2.1: Possible resolutions for bold region of tree. Good resolutions imply one change; bad ones, two.



Notice the additional step reconstructed at the root node: the Fitch algorithm reconstructs a change in tail colour in a taxon that doesn't have a tail!

This reconstruction is not logically consistent.

2.2.1 An exception

If the parent character can parsimoniously be reconstructed as present at every internal node in a single unbroken region of a tree, and nowhere else, then reductive coding does work successfully. Reductive coding may therefore be appropriate if only a subset of all possible trees are under consideration, and is always (i.e. for all trees) appropriate if a character exhibits fewer than three inapplicable tokens.

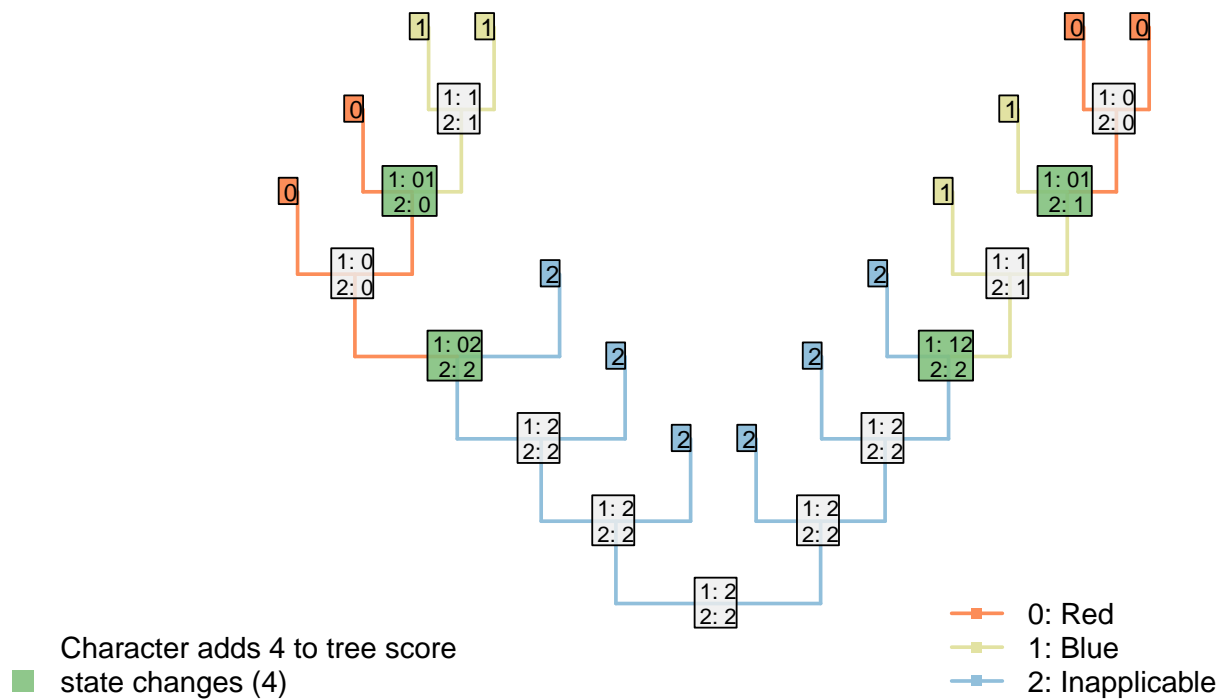
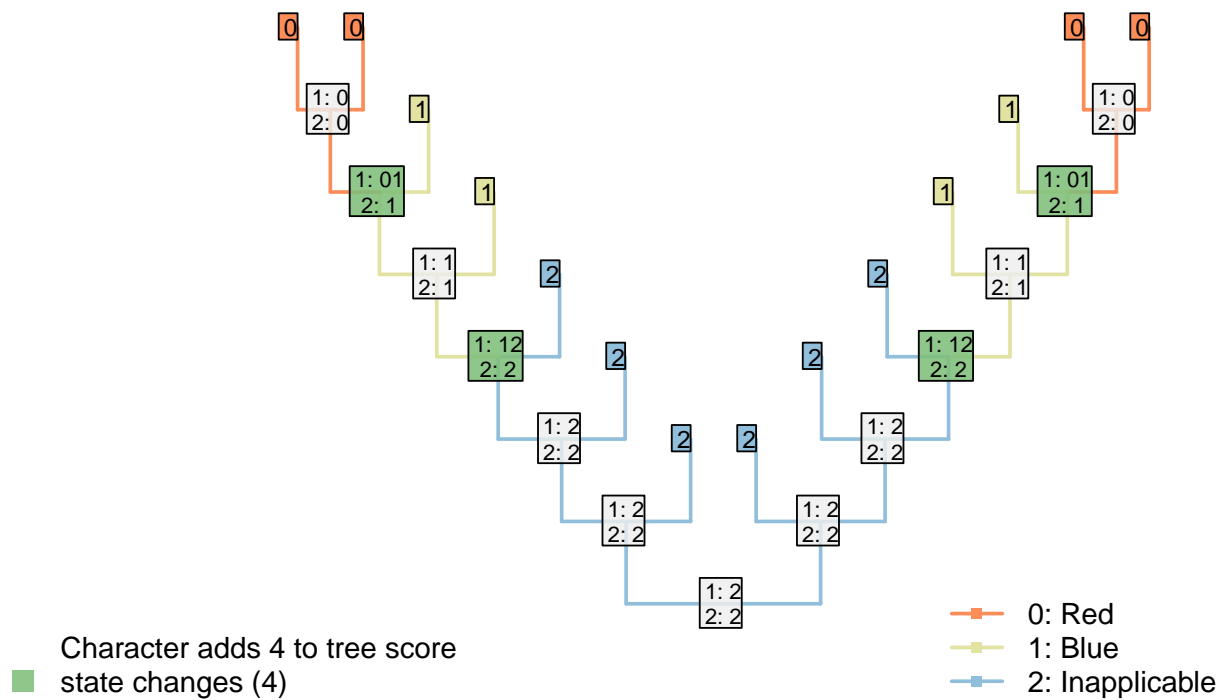
2.3 Inapplicable as an extra state

An alternative is to code the inapplicable token as an extra state:

Tail: (0), absent; (1), present.

Tail, colour: (0), red; (1), blue; (2), inapplicable.

This seems to resolve the problem case that we encountered with reductive coding:



Both trees now receive the same score for the ‘tail colour’ character, which contributes four steps. Two of these steps, however, correspond to steps that have already been counted in the parent character, reflecting the two gains of a tail.

Although this reconstruction is logically consistent, the gain (or loss) of the tail is now reflected in two characters – characters are not independent of one another.

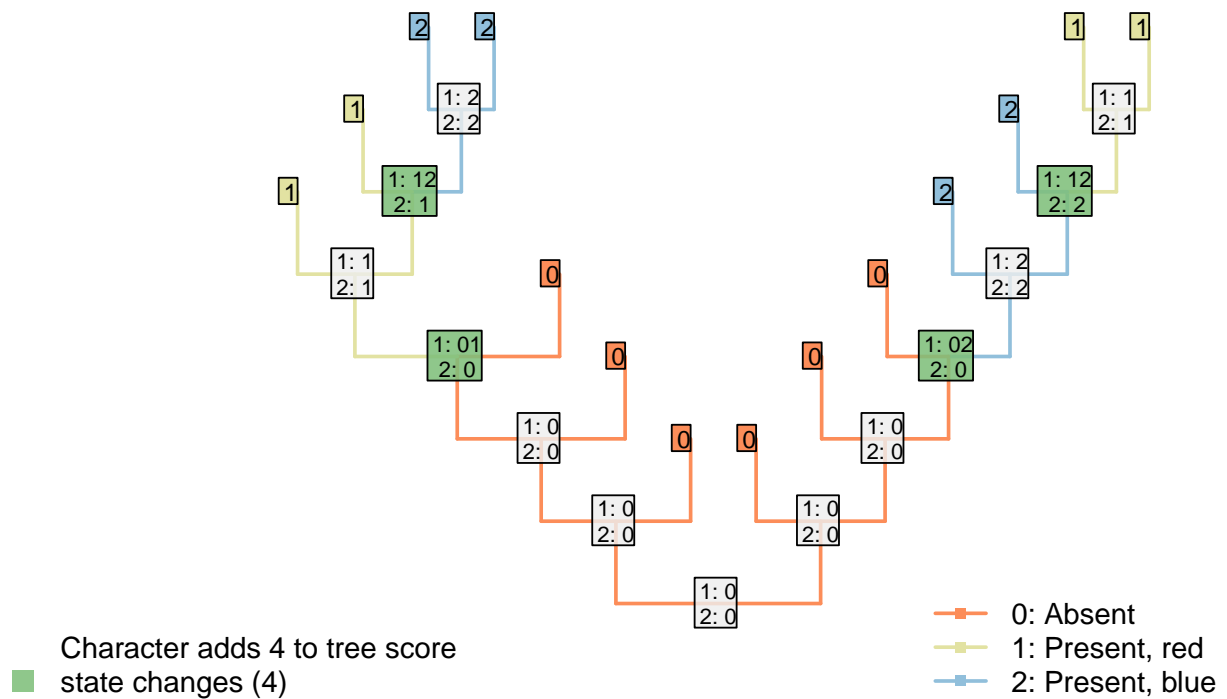
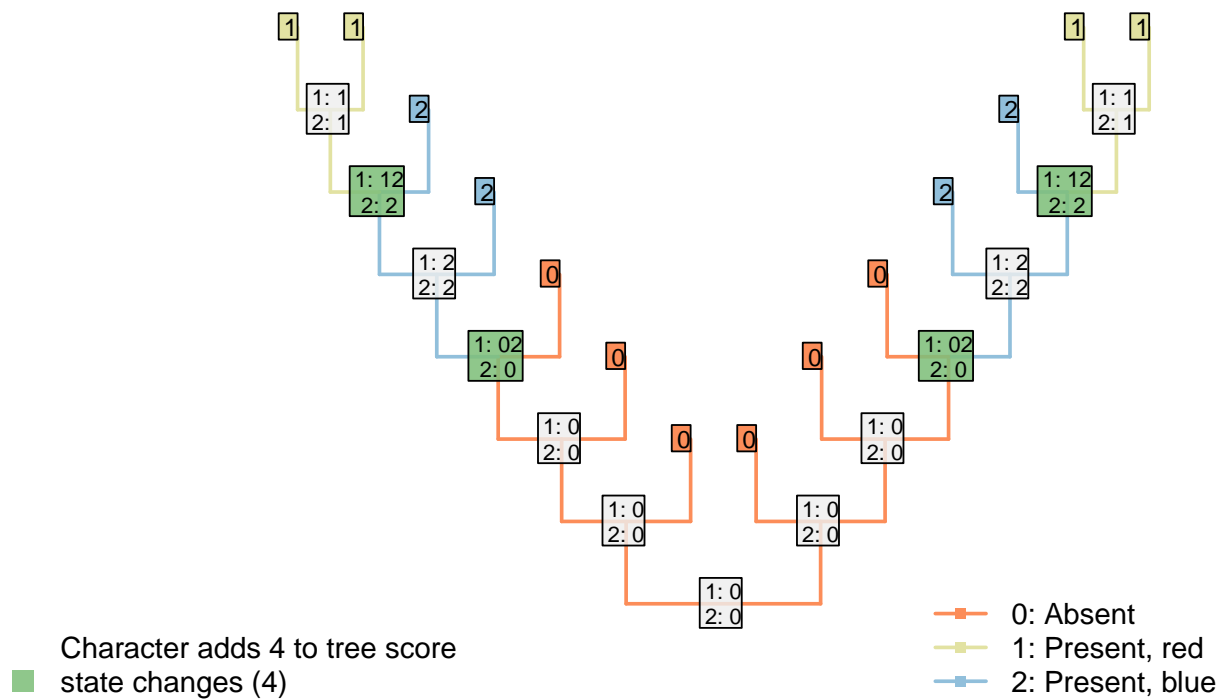
The outcome is that each ontologically dependent character serves to increase the weight of its parent character. The loss of a tail, for example, would incur a cost of one step in the tail character and one step in each ontologically dependent character, even though it represents a single evolutionary event.

2.4 A single multi-state character

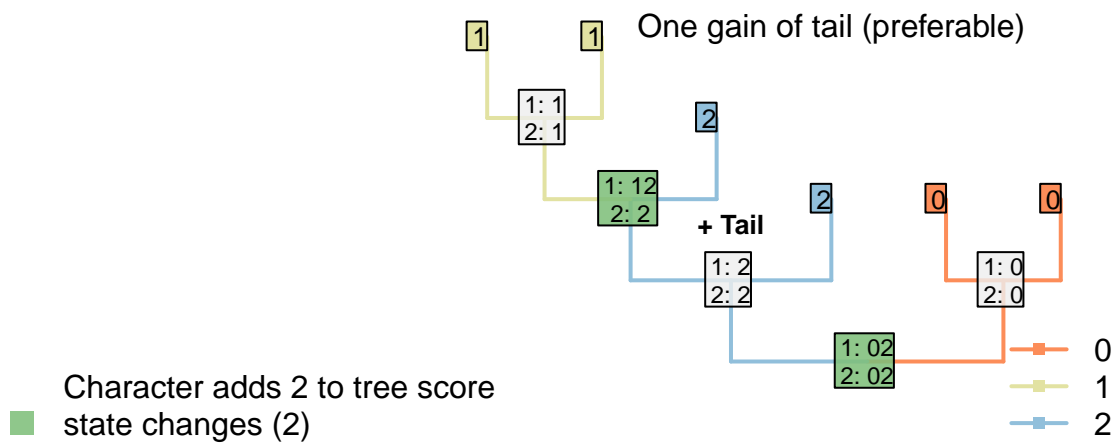
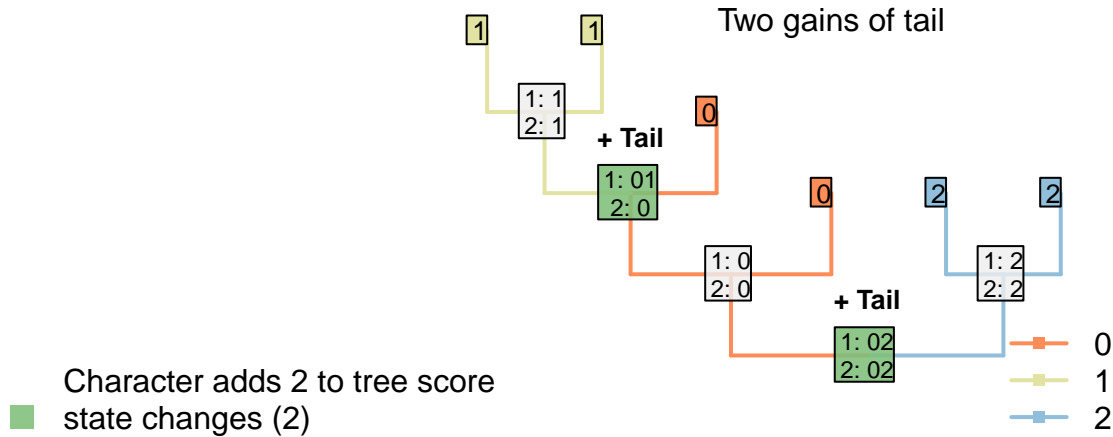
A different approach is to use a single character to denote both the presence and the colour of the tail:

Tail: (0), absent; (1), present, red; (2), present, blue.

This seems to resolve the problem case that we encountered with reductive coding:



However, we now have a situation where the gain/loss of a tail is afforded the same weight as a change in tail colour. We ought to prefer a tree where the tail evolved once (and changed colour) to one where it evolved twice (being a different colour each time).



2.5 Sankoff matrices

It would be possible to establish a Sankoff matrix (Sankoff and Rousseau, 1975; Sankoff and Cedergren, 1983) such that a change between absent and red or absent and blue cost more than a change between red and blue, but this effectively up-weights the tail character, and it's not clear that this is desirable – or how much this extra weight should be (Maddison, 1993).

2.5.1 Symmetric

Consider a character with three ontologically dependent characters:

Presence: Absent / present

Colour: Red / blue

Covering: Scaly / hairy

Shape: Straight / curly

This could be coded as a single transformation series using a Sankoff matrix:

The first thing to note is that each additional ontologically dependent character generates disproportionately more complexity in the Sankoff matrix.

Table 2.1: Tail: Cost to go from left state to top state:

	0	1	2	3	.	.	.	8
(0), absent	0	4	4	4	.	.	.	4
(1), present, red, scaly, straight	4	0	1	1	.	.	.	3
(2), present, red, scaly, curly	4	1	0	2	.	.	.	2
(3), present, red, hairy, straight	4	1	2	0	.	.	.	2
.	0	.	.	.
.	0	.	.
.	0	.
(8), present, blue, hairy, curly	4	3	2	2	.	.	.	0

Table 2.2: Tail: Cost to go from left state to top state:

	(0)	(1)	(2)	(3)	.	.	.	(8)
(0), absent	0	4	4	4	.	.	.	4
(1), present, red, scaly, straight	1	0	1	1	.	.	.	3
(2), present, red, scaly, curly	1	1	0	2	.	.	.	2
(3), present, red, hairy, straight	1	1	2	0	.	.	.	2
.	0	.	.	.
.	0	.	.
.	0	.
(8), present, blue, hairy, curly	1	3	2	2	.	.	.	0

Even if this additional complexity could be handled, the underlying issue remains that losing a tail, which arguably corresponds to a single evolutionary event, is allocated a large cost (here, 4) that grows in line with the number of ontogenetically dependant characters.

2.5.2 Gain and loss asymmetric

At the cost of symmetry, one could argue that the loss of a tail requires a single transformation, whereas the gain requires the addition of a tail and the “setting” of each ontologically dependent character, rendering an asymmetric Sankoff matrix that nevertheless respects triangular inequality (Wheeler, 1993):

Here, though, we encounter a new problem: reconstructions involving very many losses are preferred to those involving a single gain.

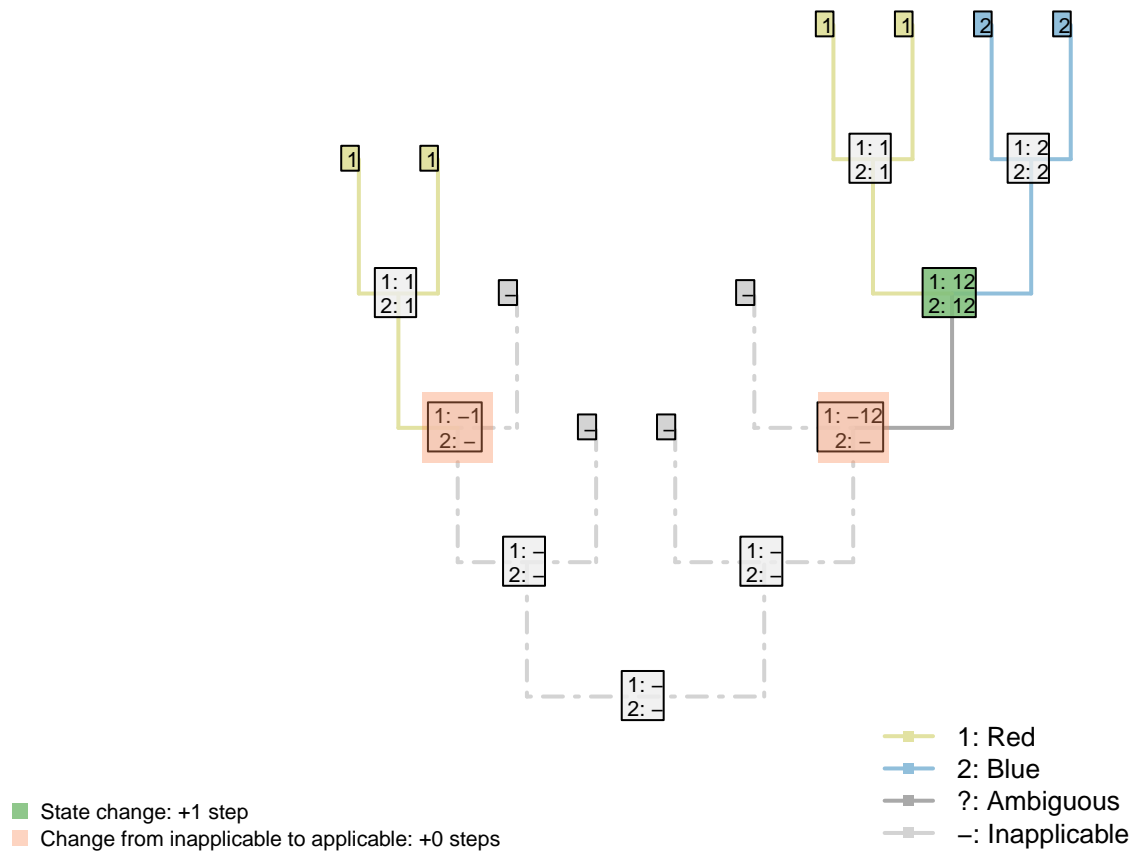
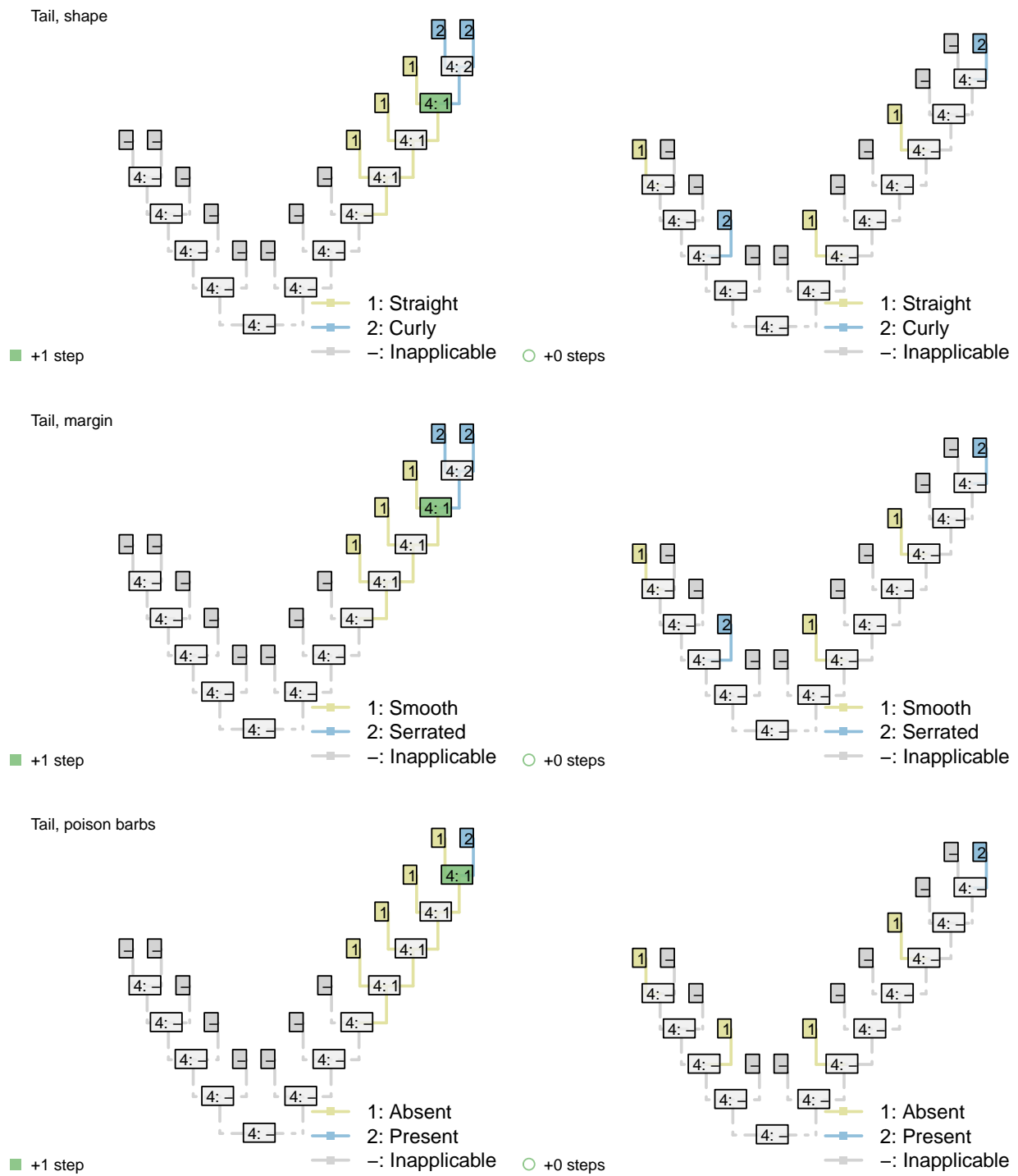


Figure 2.2: Tail colour optimization



2.7 Conclusion

No coding mechanism can generate consistent and logically meaningful tree scores when employing the Fitch algorithm. A new algorithm is needed: one that counts homoplasies instead of steps.

Chapter 3

A solution

3.1 Minimising homoplasy

A solution can be found if the goal of parsimony is recast not in terms of minimising the number of steps, but instead of minimising the amount of homoplasy in a tree.

De Laet has made this point before (De Laet, 2005; 2015), suggesting that a tree's score should be calculated as

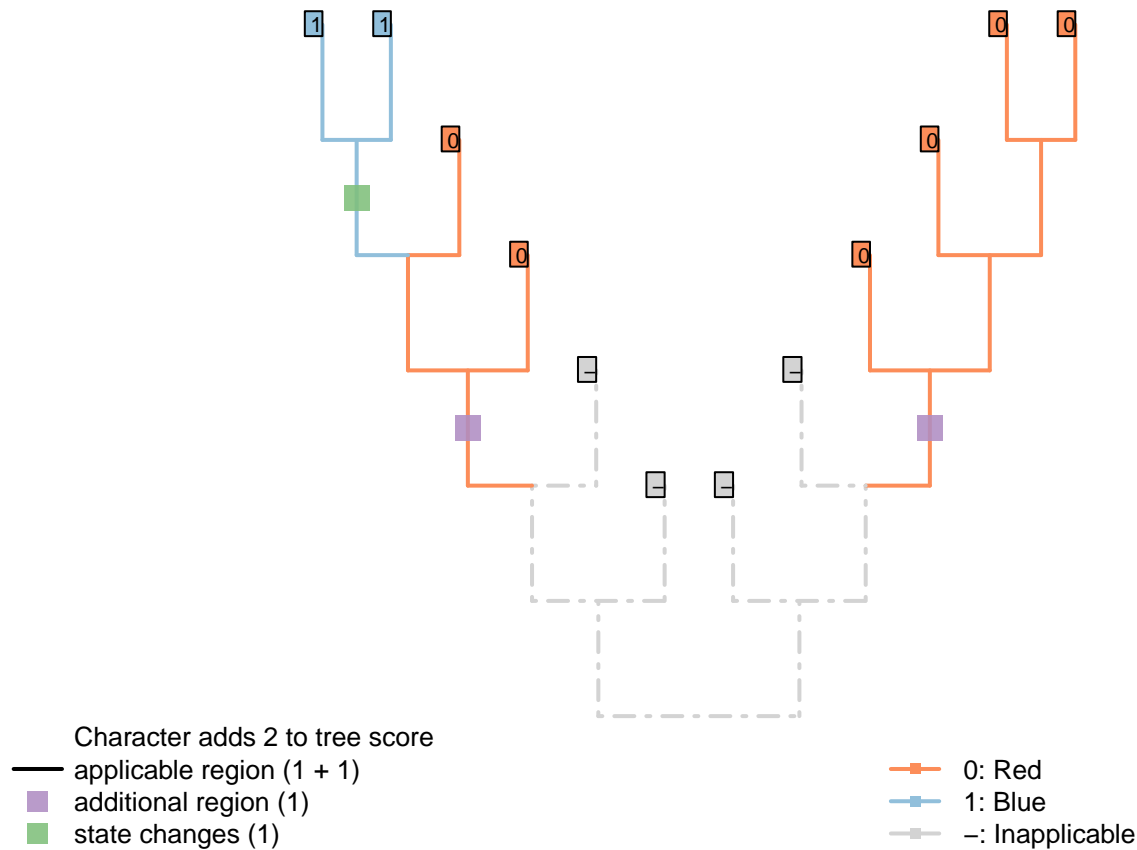
$$\text{Total score} = \text{Number of steps} + \text{Number of (additional) regions}.$$

Practically, because the number of unavoidable regions is a function of a dataset and not of a tree, one could alternatively count

$$\text{Total score} = \text{Number of steps} + \text{Number of regions}$$

which would be a constant number larger than the total score generated just counting additional regions; the absolute value of the score is not meaningful in itself and is not comparable between datasets, so the calculation method does not affect tree search.

The tree below gives an example of a tree in which a character is applicable in two regions (one more than the minimum possible, one) and one state change.



This score denotes two evolutionary observations that cannot be attributed to inheritance from a common ancestor: the blueness of tail in the blue tailed taxa (as the common ancestor inherited a red tail), and the redness of tail in the second region of the tree (as the common ancestor of all tail-bearing taxa did not itself have a tail, so tail colour could not be inherited).

3.1.1 What does it take to denote separate regions?

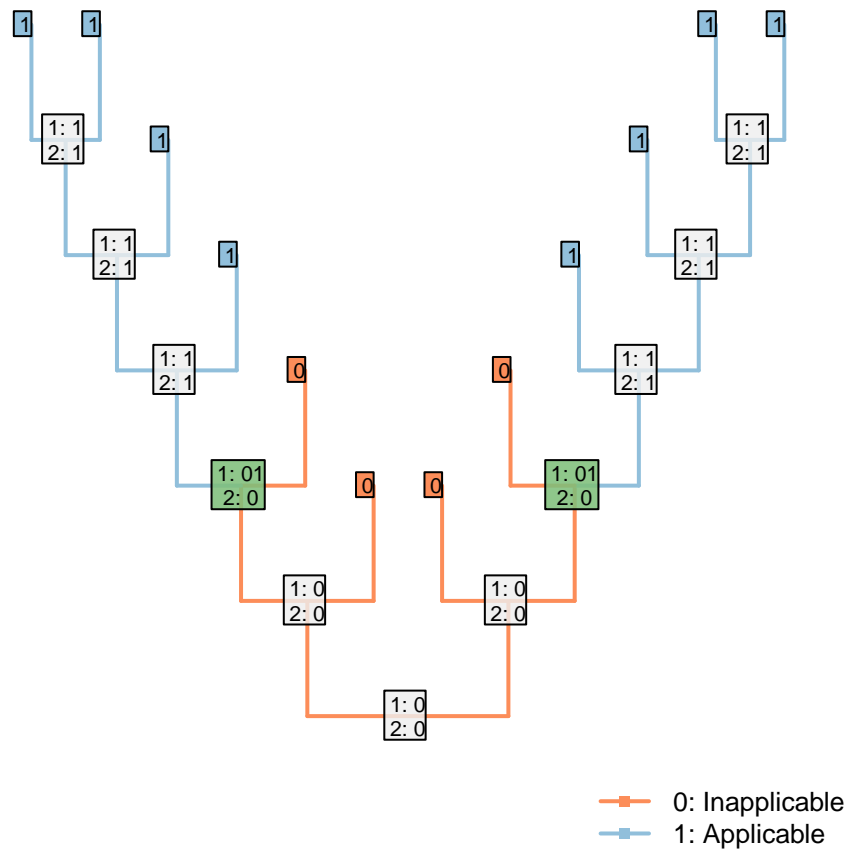
It takes three inapplicable nodes (including tips) to force two regions of the tree to be separated by an inapplicable region.

This can be established by imagining the Fitch optimisation of a separate character

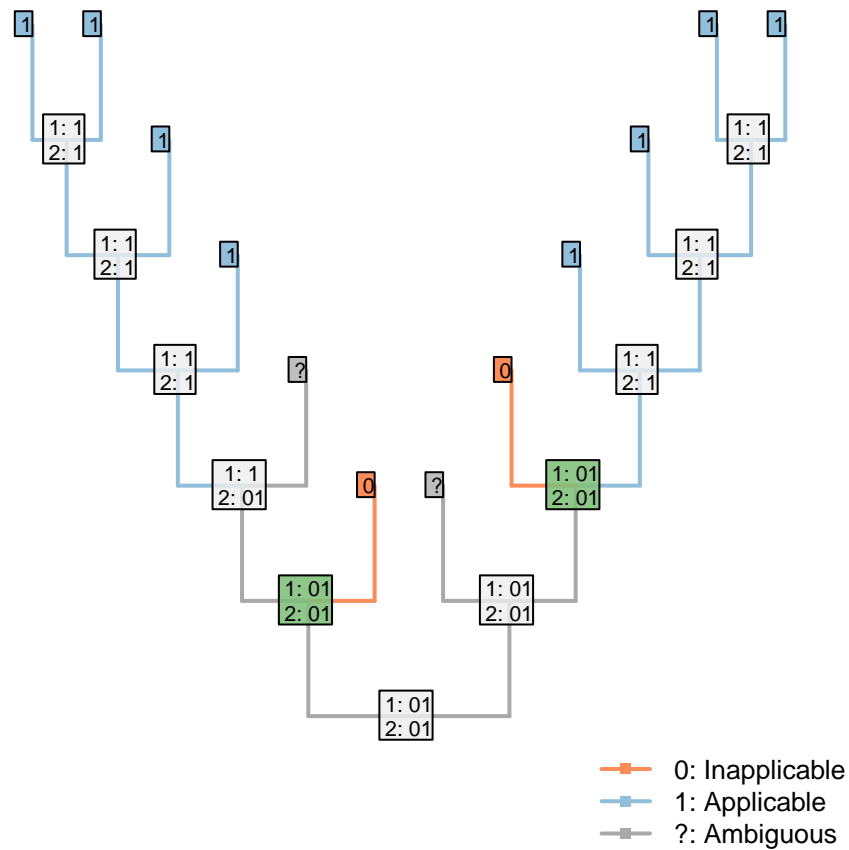
Applicability of the character of interest: (0), inapplicable; (1), applicable

In the case of tail colour, this applicability character has the same distribution as the presence / absence of the tail, but this is not necessarily the case (there may be a range of reasons to code a character as inapplicable).

In the tree shown above, the Fitch algorithm identifies two regions where the applicability character is unambiguously ‘applicable’:



If one of the inapplicable tips had instead been ambiguous, then the same distribution would arise:



This reconstruction maximises the inferred homology between tails, and so increases the opportunity to attribute shared colours in the tail to common ancestry. As such, our algorithm chooses to interpret this region as applicable wherever it parsimoniously can.

Note that the three inapplicable tips necessary to define an inapplicable region must be in a contiguous region of the tree, separated from one another only by taxa whose applicability is ambiguous, in order for two applicable regions to be reconstructed as separate.

3.1.2 How this fixes the problem

This overcomes the problem where steps could be avoided by inferring multiple innovations of a character:

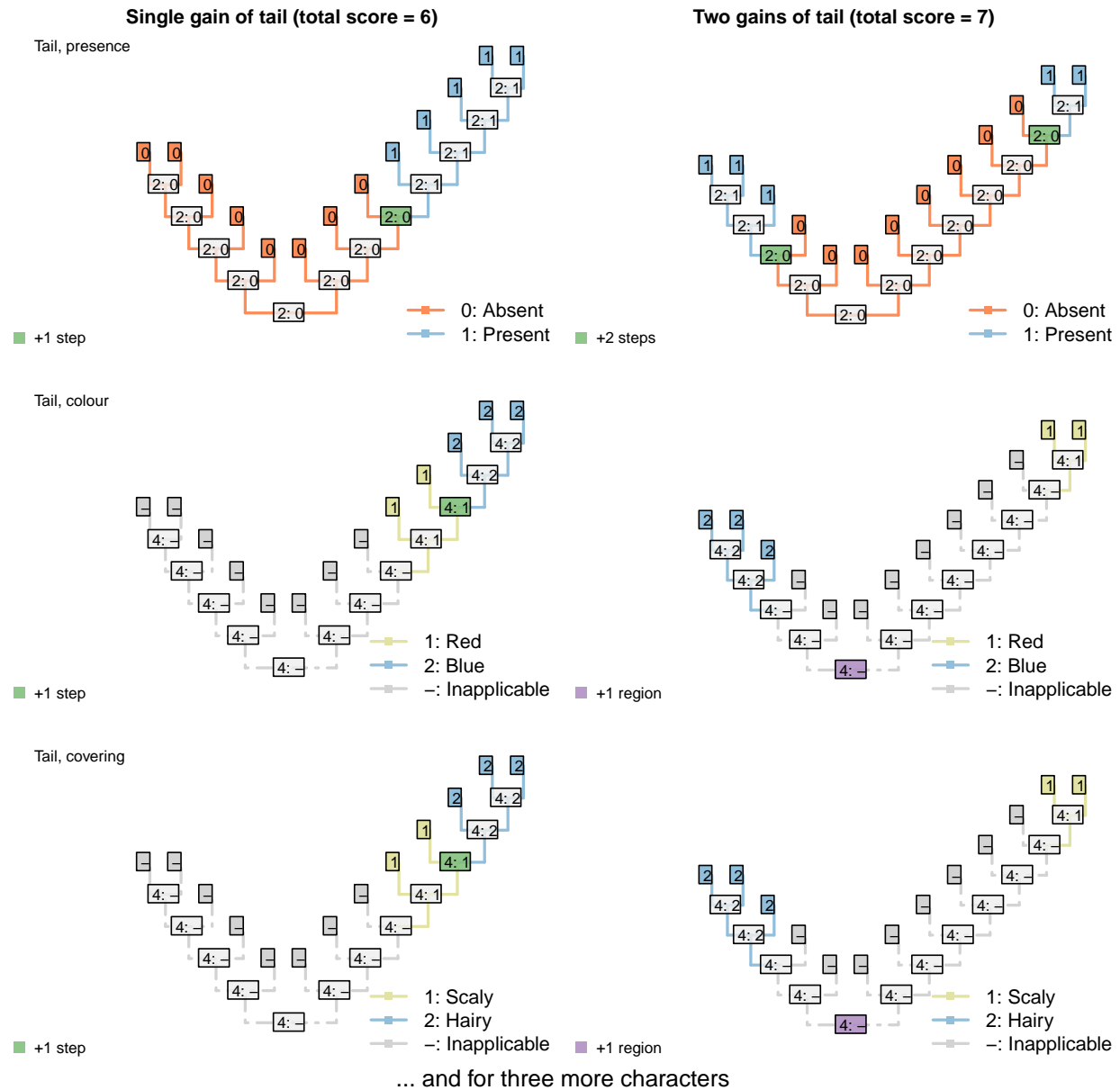


Figure 3.1: Reconstructions of tail presence and five contingent characters (only two shown)

3.1.3 Summary

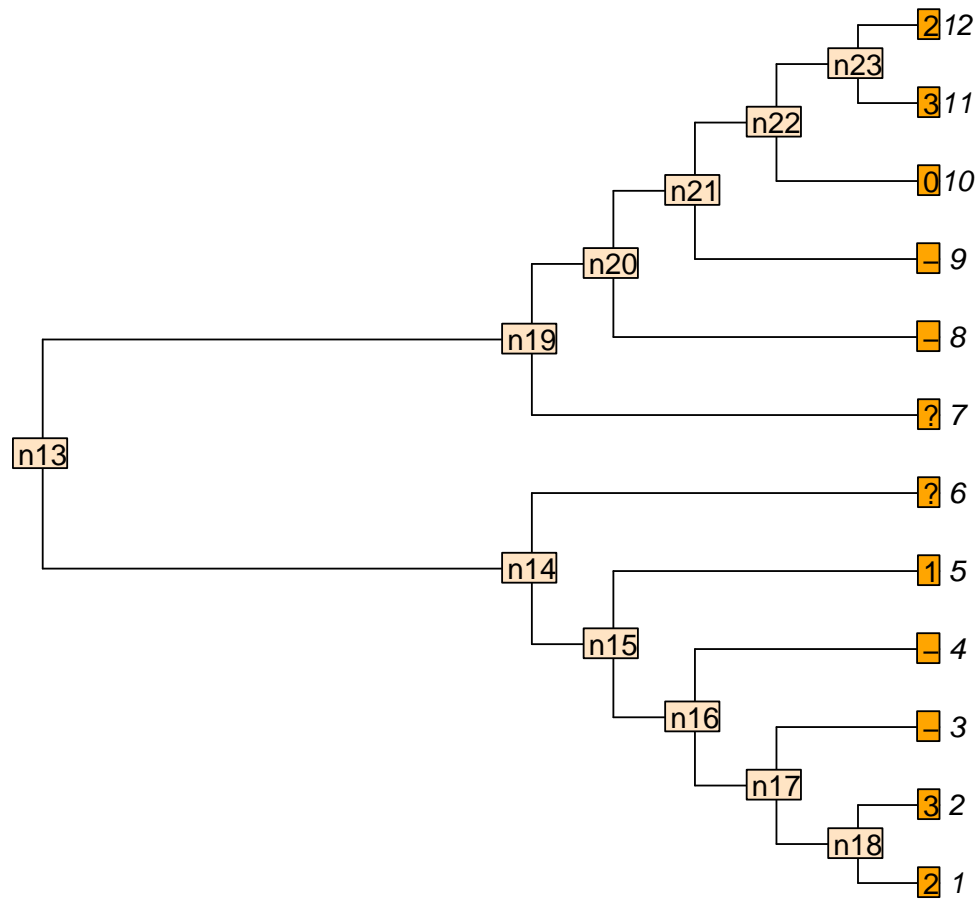
This is the desired behaviour. But how do we count this in practice?

In brief, we evaluate for each tip whether the character in question is applicable, inapplicable, or ambiguous (could be either), and use the standard Fitch algorithm on this applicability data to reconstruct the state of each internal node, reconstructing ambiguous nodes on the uppass as applicable.

This done, we conduct a second Fitch-like pass on the tree, in which we count transformations if they occur at nodes in which the character has been reconstructed as applicable. Additional regions are also counted on this downpass, by counting nodes that are ancestral to an inapplicable region of the tree that itself leads to an as-yet-uncounted applicable region.

3.2 Algorithmic implementation

Consider a tree with 12 taxa and the following multi-state characters with inapplicable data 23--1??--032; say the character is “colour of the tail” ranging from 0 to 3 (four colours). Four taxa in our example have no tail (hence the inapplicable data -) and for two taxa, the data is missing (?- we don’t know the colour of the tail or even whether the taxa have a tail or not).



We can use the `Inapp` package to apply our four-pass inapplicable algorithm to this character on this tree.

```
## Loading the Inapp package
library(Inapp)

## The tree
tree <- read.tree(text = "((((((1,2),3),4),5),6),(7,(8,(9,(10,(11,12))))))));")

## The character
character <- "23--1??--032"

## Applying the NA algorithm
matrix <- apply.reconstruction(tree, character, method = "NA")
```

Here is what is happening:

3.2.1 Passes 1 & 2

The first two passes are a standard Fitch algorithm applied to the parent character of the studied character (see Fitch algorithm) with a special rule for the inapplicable state (-).

For the first pass (first downpass):

- If state in common between the two descendants is the inapplicable state, but that both have also applicable states, set the node's state to be the union between the descendants states (rather than their state in common).
- If there is no state in common between the descendants and both descendants have applicable states, remove the inapplicable state from their union (rather than simply setting the nodal state to their union).

For the second pass (first uppass):

- If the focal node has both applicable and inapplicable states, set it to be the inapplicable state only if its ancestor has also only the inapplicable state, else remove the inapplicable state.
- If the focal node has only an inapplicable state and its ancestor has not only the inapplicable state, set it to be the union between its descendants states if their are both applicable, else, leave it as the inapplicable state.

```
## Plotting the NA two first passes
plot(matrix, passes = c(1,2), counts = 0,
      legend.pos = 'none', show.labels = c(1, 2))
```

The parent character can be considered as a binary character “presence (1) or absence (0) of a tail” that would be 11001??00111. The character would be reconstructed as:

```
## The parent character
parent_character <- "11001??00111"

## Applying the Fitch algorithm
matrix_parent <- apply.reconstruction(tree, parent_character, method = "Fitch")
plot(matrix_parent, passes = 1:2, legend.pos='none', show.labels = c(1, 2))
```

As you can see, both reconstructions are identical: nodes with no tail are denoted as 0 in the case of the “parent character” and as - for our current character. Note however that contrary to the Fitch algorithm, there is no tree score counting in our algorithm for the two first passes. Indeed, in the case of the Fitch reconstruction of the “parent character”, the gain or losses of a tail are counted but not the changes in states for the subtending character (the tree score is 3 in Fitch, 5 in our case).

3.2.2 Pass 3

The third pass further resolves ambiguities at nodal states. If the node is applicable, the standard Fitch downpass comparisons between the descendants are applied (see Fitch algorithm) but with the rules relative to the inapplicable state described for the first downpass above.

During this pass, we can also count the tree score. This score is composed of both:

- the change in states (e.g. the change in the colour of the tail)
- the change between applicable and inapplicable regions (e.g. the change in the parent character: a gain or a loss of the tail)

The changes of states are calculated the same way as Fitch **for the applicable states only**:

- If there is no state in common between both node's descendants and that the node, and its descendants have a least one applicable state, increment the tree score.

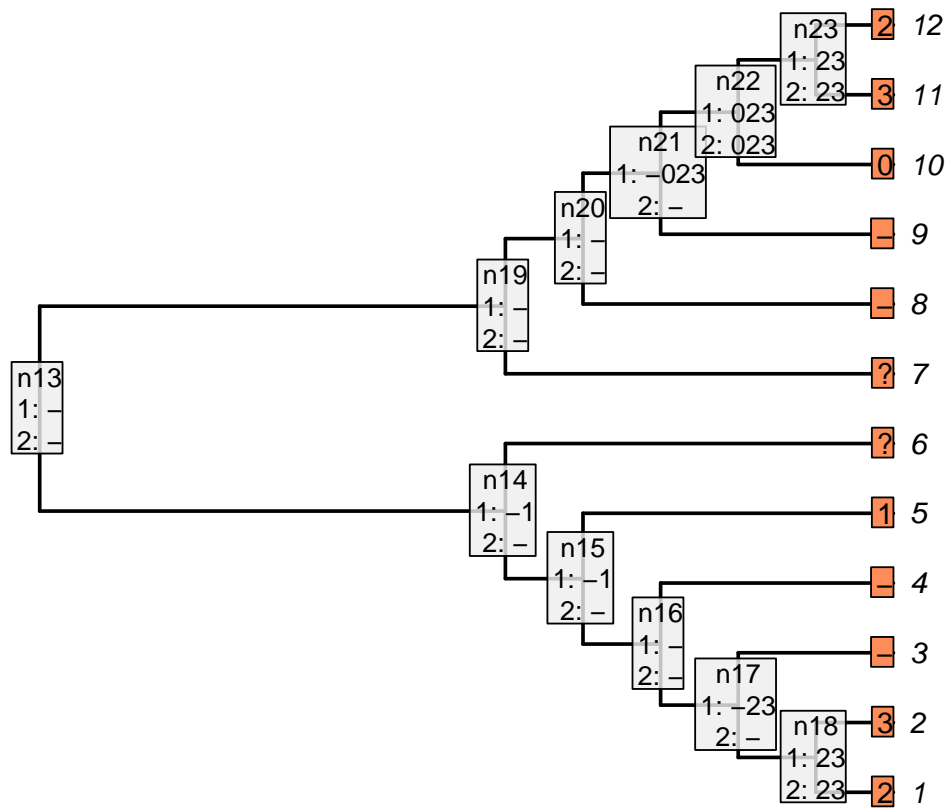


Figure 3.2: Inapplicable reconstruction after two passes

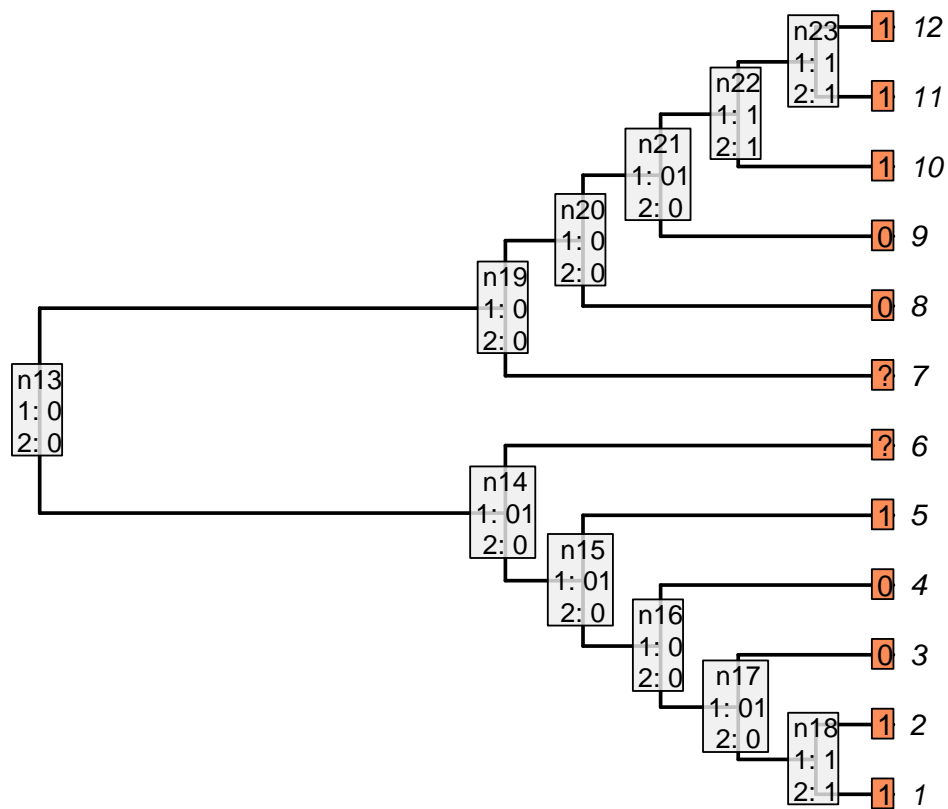


Figure 3.3: Fitch reconstruction of the parent character

```
plot(matrix, passes = c(1,2,3), counts = 2, show.labels = c(1, 2))
```

For example, for node **n23**, there is no state in common between the tip 12 (2) and 11 (3), the tree score is incremented at this node (case 1 above). Note, however, that for node **n21**, there is no state in common between node **n22** (023) and tip 9 (-) but the score is not incremented since it does not concern applicable states only. In other words, there is no change in state at the node **n21** from the tail having a colour 0, 2 or 3 to the tail not being present (-) but rather a change in the parent character between presence and absence of the tail (present is 023 and absent is -).

3.2.2.1 Tracking applicable regions

To know whether any node leads to a region of applicable states we can use a “tracker” for each node that tells us at any moment whether descendants of a node contain applicable data or not. When a node is inapplicable and has a descendant whose lineage leads to applicable regions, an extra applicable region is implied by the tree. In other words, following our “colour of the tail” character, extra applicable regions imply independent appearances of the tail somewhere in the node’s descendants.

The tracker is initialised during the *second pass* (first uppass) and is updated during the *third pass* (second downpass). The tracker works as follows for each node’s left and right descendants:

- If the descendant state is applicable or leads to an applicable region, then the node leads to an applicable region; else, it does not.

The trackers are initialised for each node during the first uppass and then propagated back down the tree during the second downpass.

Using these trackers, we can then increment the tree score for all changes that imply a new applicable region. The switch to or from an inapplicable and applicable region are counted as follows:

- If the node is inapplicable and both descendants lead to regions of applicable states, increment the region count.
- If the node is applicable, but has an inapplicable descendant that leads to a region of applicable states, increment the region count.

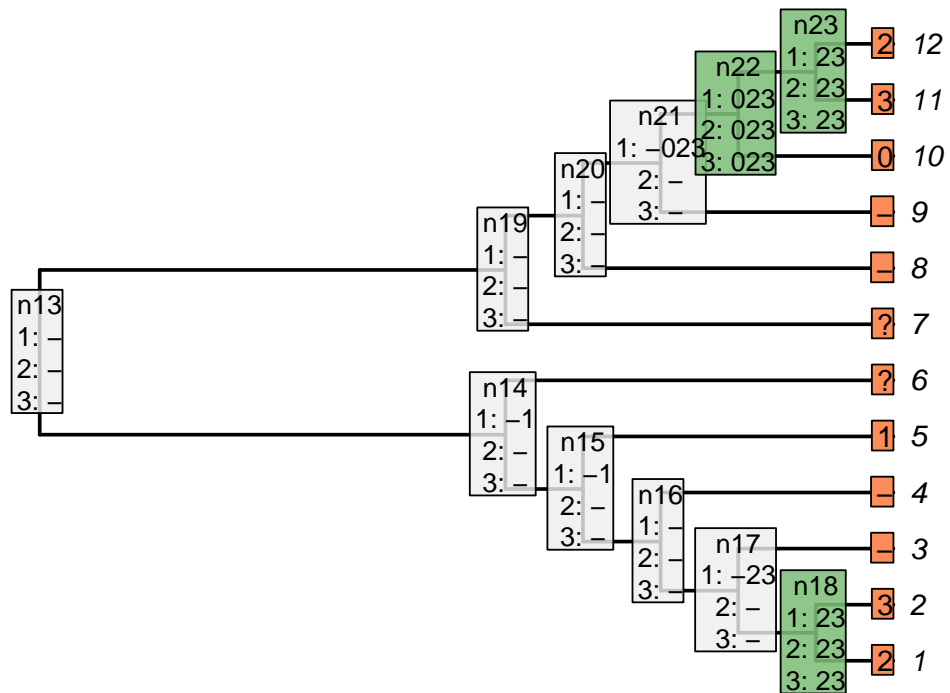
```
plot(matrix, passes = c(1,2,3), counts = 1, show.labels = c(1, 2))
```

For example, node **n15** is solved as inapplicable but both his descendants lead to two independent applicable regions (tip 5 with the state 1 and node **n18** with the states 1 and 2). This implies an independent change in the parent character (in our example, tail is absent at node **n15** but evolves independently at tip 5 and node **n18**). Conversely, node **n21** is solved as inapplicable but not both his descendants lead to independent applicable regions. This node does thus not imply an independent change in the parent character.

Note that the number of applicable regions for a character is always at least 1 (unless every taxa has the inapplicable state) and therefore, we only count the *additional* regions.

Combining both scores – the number of changes in character states and the number of additional applicable regions – we get indeed a total tree score of 5 for this character on this tree.

```
## Plotting the NA two first passes
plot(matrix, passes = c(1,2,3), counts = c(1,2), show.labels = c(1,2))
```



Character adds 5 to tree score
 state changes (3)

Figure 3.4: State changes

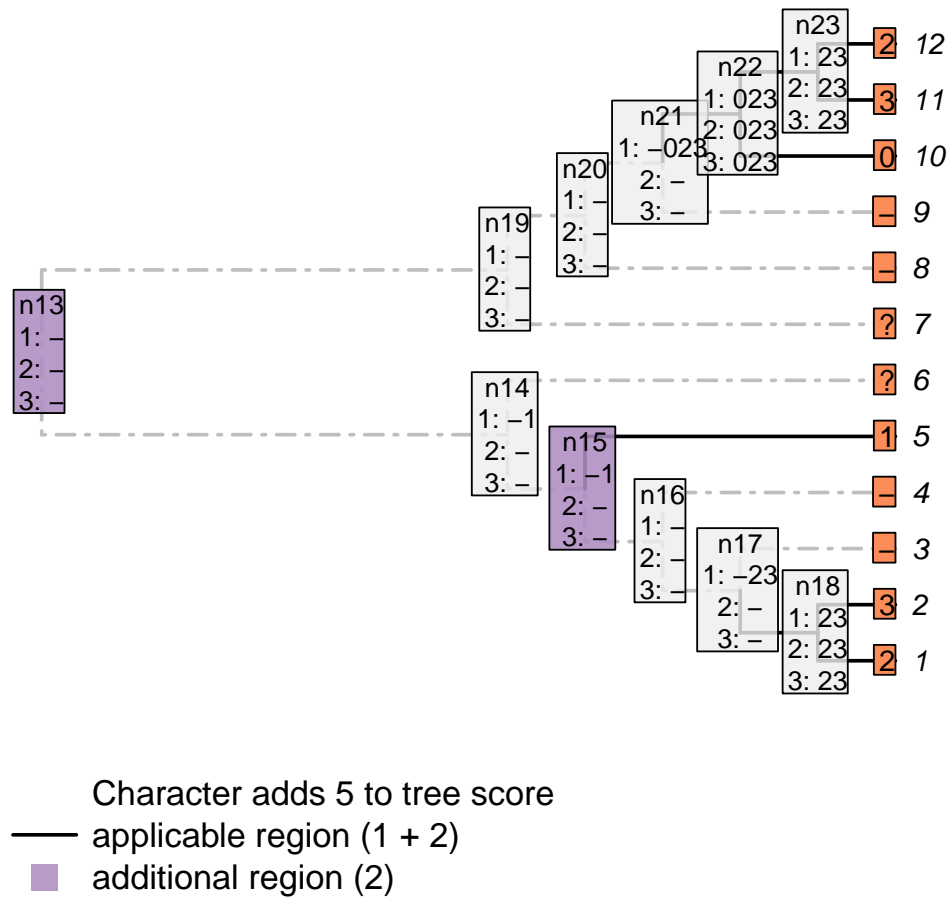
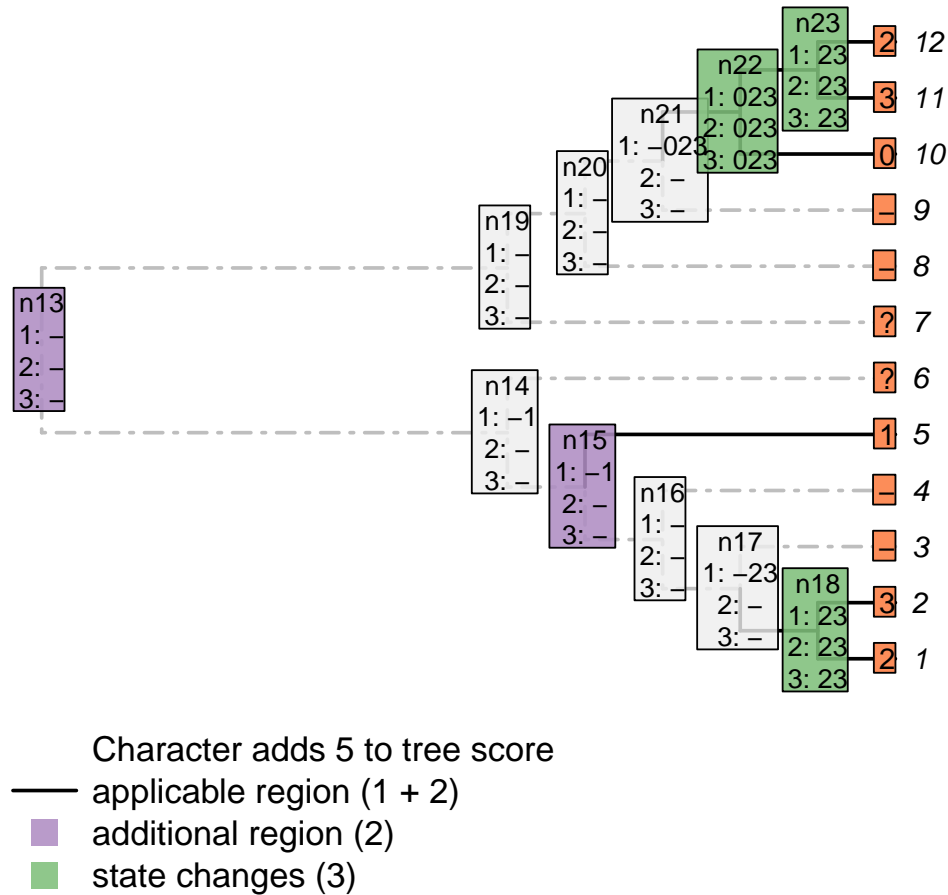


Figure 3.5: Counting applicable regions



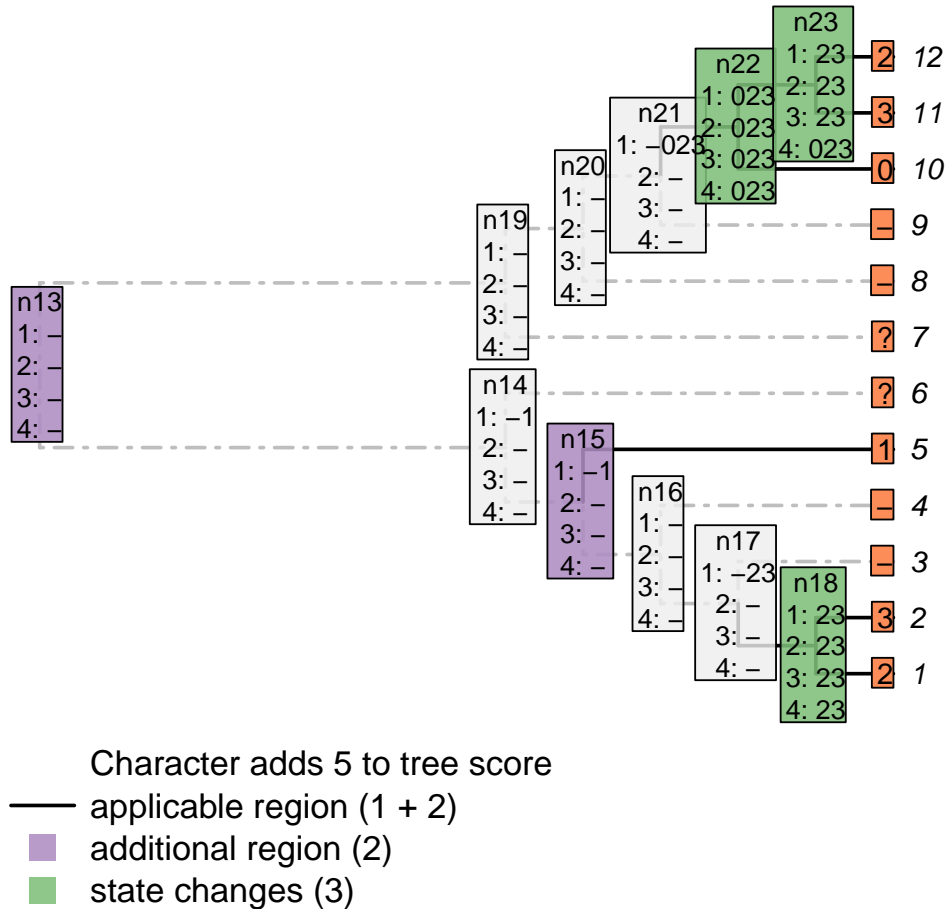
Using the first three passes is enough to get the tree score (while taking into account inapplicable data!) but does not solve all ancestral reconstructions. A fourth pass (second uppass) might be necessary to finalise the node states reconstructions.

3.2.3 Pass 4

In the example above, the node **n23** is still not correctly solved after the third pass. It could conceivably be state 0 (with transformations to states 2 and 3 occurring on the branches leading to tips 11 and 12 respectively). As such, its final state reconstruction should be 023. To reach the correct final reconstructions, we apply a final pass of the algorithm. This algorithm, similarly to the second pass of the Fitch algorithm is used to solve ambiguities in the ancestral nodes reconstructions (although the score of the tree is already known). It follows these rules and only applies to nodes and ancestors that have at least one applicable token for themselves and their ancestor (nodes that are inapplicable are already solved):

- If there is a state *in common* between the node and its ancestor or between the ancestor and the states *in common* of its descendants, resolve the node to be this state in common.
- If there is nothing in common between the node and its ancestor or between its descendants, solve the node as either:
 1. being the ancestors state if the any of the descendants' have at least one inapplicable state but no state in common with the ancestor.
 2. being the union of the ancestor's and the descendants' states if the any of the descendants have at least one inapplicable and have at least one state in common with the ancestor.
 3. being the union of the ancestor's and the current node states the descendants have no inapplicable state.

```
plot(matrix, passes = c(1,2,3,4), counts = c(1,2), show.labels = c(1,2))#, col.states=TRUE)
```



3.3 Software implementation

This algorithm has been implemented in two **R** packages. **Inapp** provides an interactive visualization of how the score of a user-specified tree is calculated for any character under different approaches to inapplicable data. This package was used to generate many of the figures in this document.

TreeSearch allows for parsimony tree searches with the inapplicable algorithm (Brazeau et al., 2018). It includes heuristic search options that make it possible to search reasonable-sized matrices, and includes an option for equal or implied weighting.

TreeSearch is a front-end to the **morpholib C** library, which will eventually be implemented in the standalone **Morphy** program for rapid phylogenetic searches.

Chapter 4

Coding data

The availability of our algorithm has some implications for how investigators might choose to code characters.

4.1 Multiple dependencies

It's not a problem to have characters dependent on characters that are dependent on characters. Consider the following characters, whose descriptions are written in order to emphasize their hierarchical nature (following the recommendations of Sereno, 2007):

1. Appendages: (0), absent; (1), present.
2. Appendages, termination: (0), blunt; (1), sucker; (2), claw.
3. Appendages, suckers, morphology: (0), round; (1), polygonal.
4. Appendages, claws, morphology: (0), smooth; (1), serrated.

The included taxa may or may not bear appendages; if they do, then the appendages may end either with either claws or suckers, or neither (but not both). Claws come in two flavours, smooth and serrated; suckers come in two shapes, rounded and polygonal.

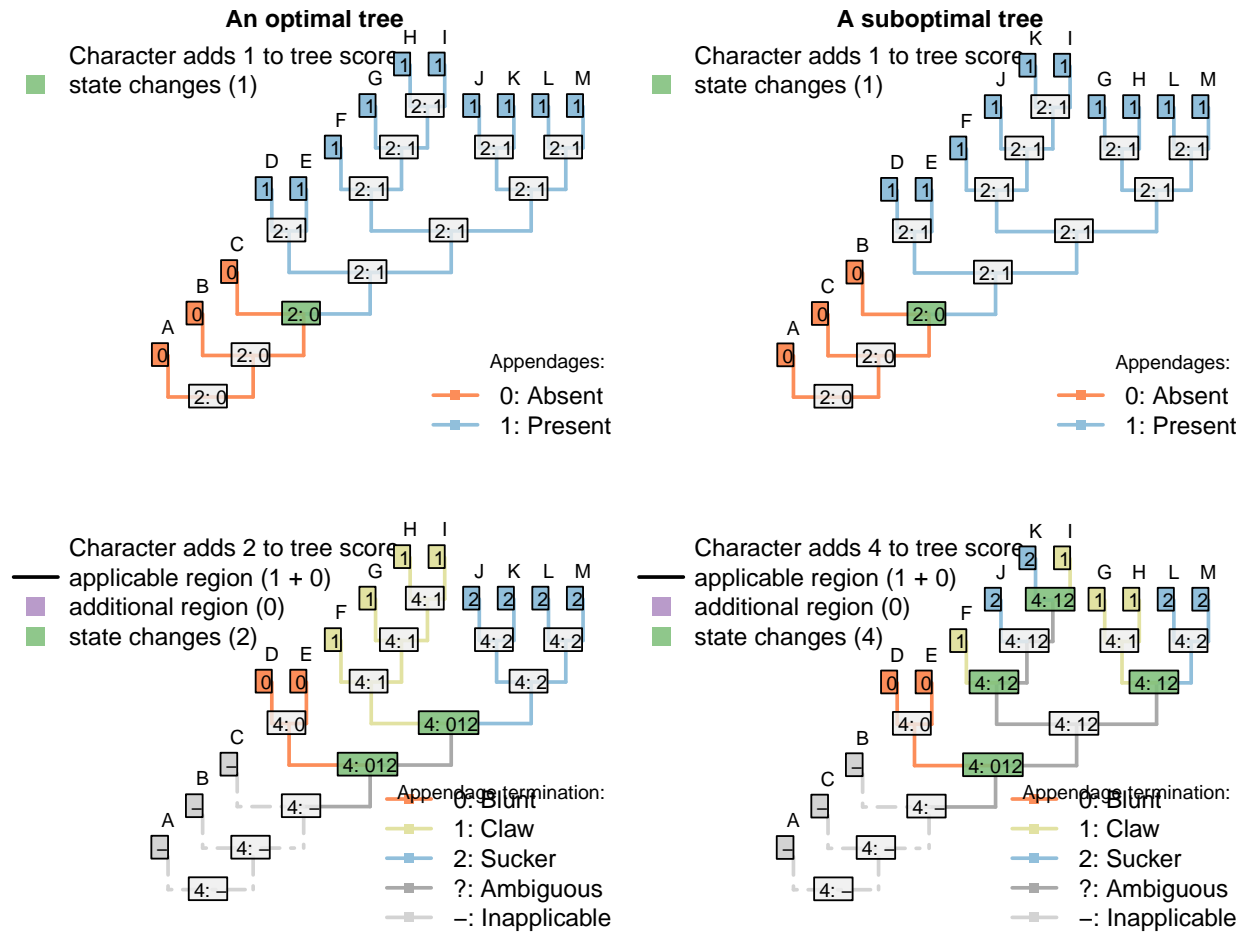
- If character 1 (appendages) is absent, then characters 2–4 are inapplicable. Otherwise, character 2 (appendage termination) must take one of the three applicable values.
- If character 2 (termination) has state 0 (blunt), then characters 3 and 4 (morphology of sucker / claw) are inapplicable.
- If character 2 (termination) has state 1 (sucker), then character 3 (sucker morphology) is applicable and character 4 (claw morphology) is inapplicable.
- If character 2 (termination) has state 2 (claw), then character 3 (sucker morphology) is inapplicable and character 4 (claw morphology) is applicable.

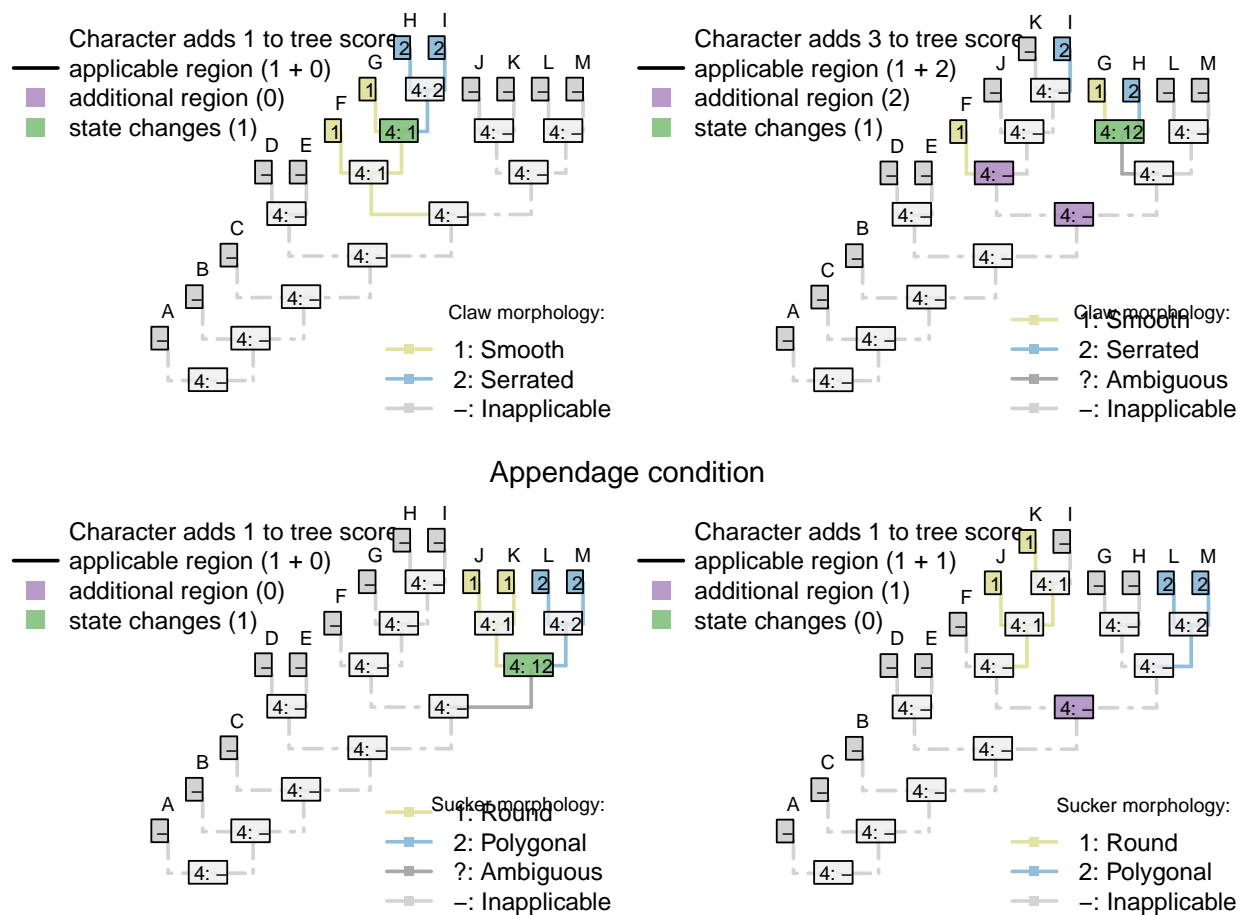
A sample character matrix might look like this:

Which would plot on a tree thus:

Table 4.1: Heirarchichal characters

	A	B	C	D	E	F	G	H	I	J	K	L	M
Appendages: (0), absent; (1), present.	0	0	0	1	1	1	1	1	1	1	1	1	1
Appendage termination: (1), blunt; (2), sucker; (3), claw.	-	-	-	1	1	2	2	2	2	3	3	3	3
Sucker morphology: (1), smooth; (2), serrated.	-	-	-	-	-	1	1	2	2	-	-	-	-
Claw morphology: (1), round; (2), polygonal.	-	-	-	-	-	-	-	-	-	1	1	2	2





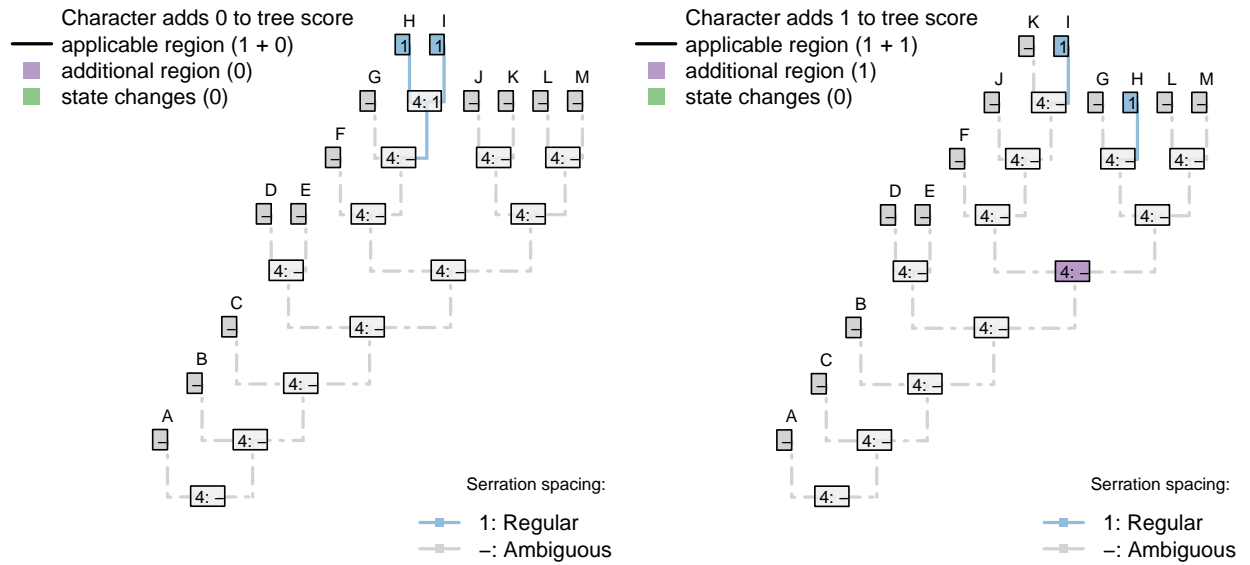
There's no limit to the depth of recursion: one could add a further character

5. Appendages, claws, serrations, spacing: (1), regular; (2), irregular.

that would be inapplicable in all taxa that lacked serrated claws.

Table 4.2: An invariant character, tail colour, contributes as much to tree score as a variable one, body colour.

	A	B	C	D	E	F	G	H
Tail: (0), absent; (1), present	0	0	0	0	1	1	1	1
Tail colour: (1), blue; (-), inapplicable	-	-	-	-	1	1	1	1
Body colour: (1), black; (2), white	1	1	2	2	2	2	1	1



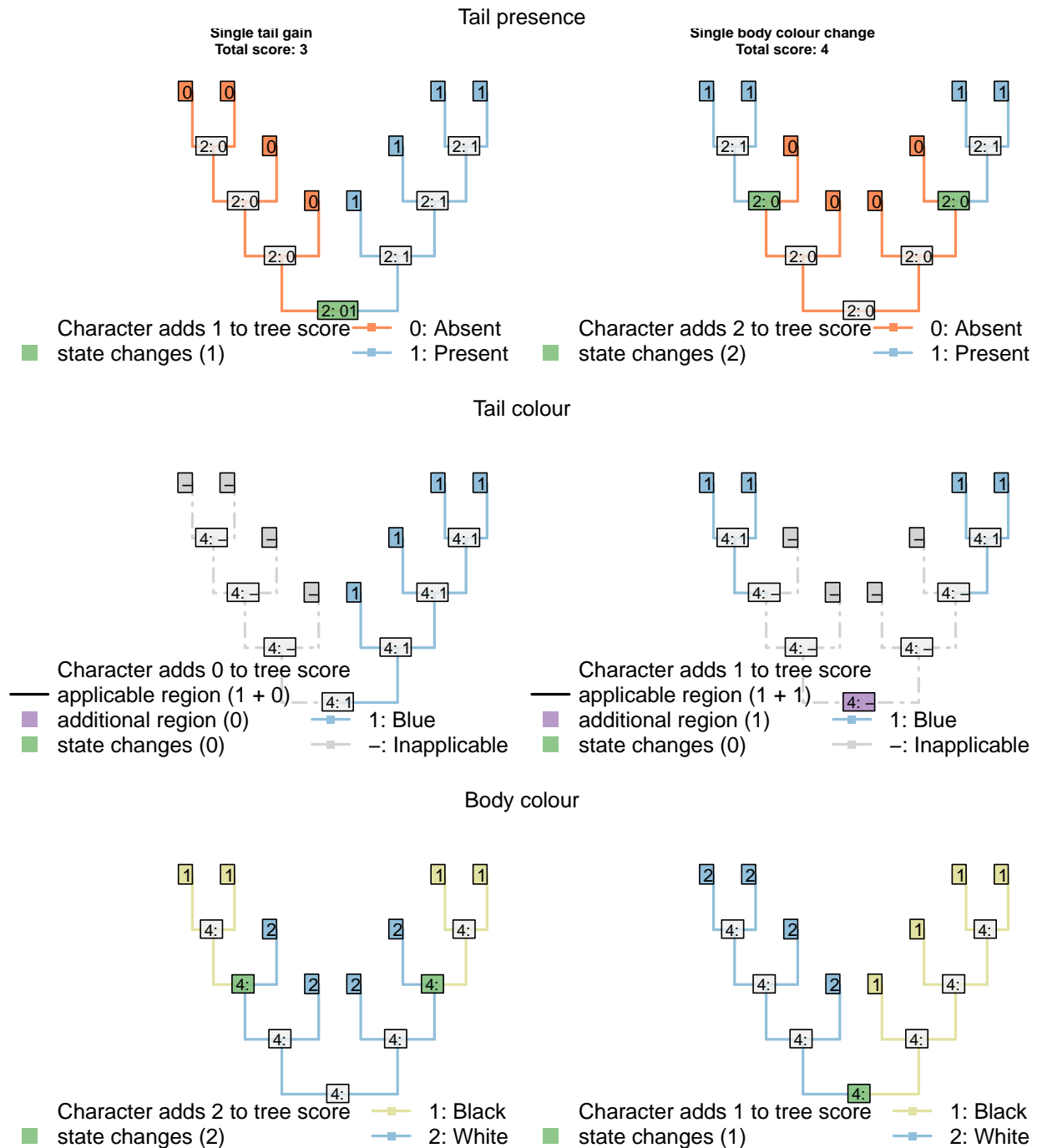
To readers familiar with standard Fitch parsimony, it will be surprising to notice that the two trees receive a different score for this invariant character. When our algorithm is employed, invariant characters that contain inapplicable tokens can inform parsimony.

4.2 Invariant characters can inform parsimony

Consider a situation in which every tail in the observed taxa is blue – but the same complex molecular machinery is responsible for this blue colouration in every taxon.

If its underlying mechanism is considered biologically and evolutionarily meaningful, then a systematist might opt to include tail colour as an additional character, even though it is invariant in the taxa of interest. Reconstructions that attribute this common colouration to common ancestry will be more parsimonious than those that do not.

Let's compare two trees. The first groups taxa based on the presence of tails; the other groups taxa based on body colour.



Where the tail has a single origin (one step), blue colouration also evolves once (zero steps), but body colour must change twice (two steps; total score = three). But where body colour changes only once (one step), the tail necessarily arises twice (two steps), meaning two independent origins of its distinctive blue colouration (one extra homoplasy; total score = four)

If the invariant tail colour character had not been included, both trees would have the same score, and there would be nothing to choose between them. As such, the inclusion or exclusion of invariant characters must be carefully evaluated: if there is a case that an invariant (ontologically dependent) character implies an exclusive common ancestry between those taxa that share it, then it should be included; if not, then it should be excluded.

Table 4.3: Tail colour is variable but ‘parsimony uninformative’

	A	B	C	D	E	F	G	H	I
Tail: (0), absent; (1), present	0	0	0	0	1	1	1	1	1
Tail colour: (1), red; (2), blue; (-), inapplicable	-	-	-	-	1	1	1	1	**2**
Body colour: (1), black; (2), white	1	1	2	2	2	2	1	1	1

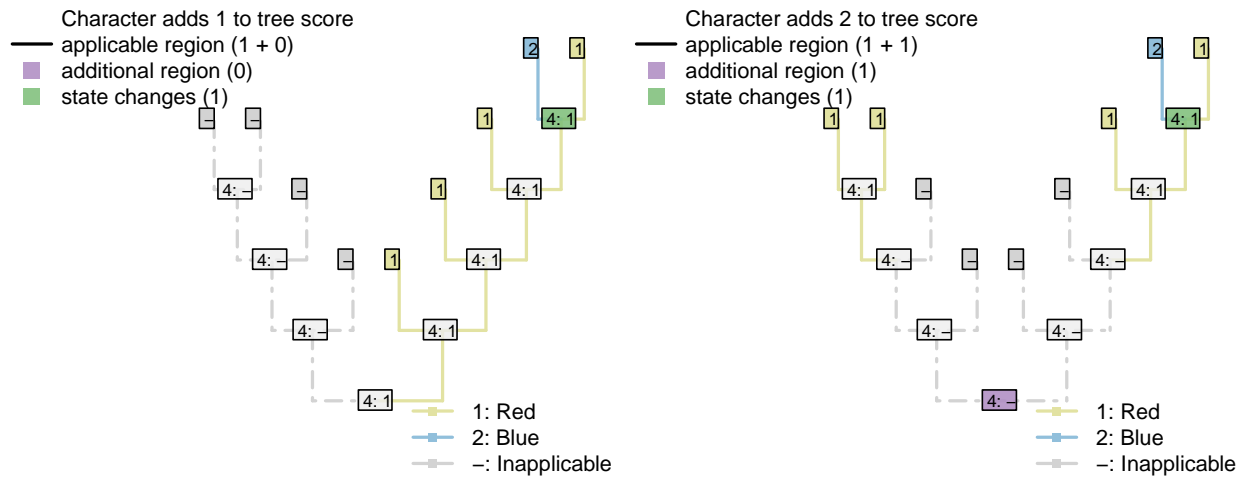


Figure 4.1: Tail colour

4.3 Variable but ‘parsimony uninformative’ characters can inform parsimony

The same effect of course follows if a character has an additional state that is only observed in one taxon. Any tree that implies that blueness evolves multiple times will incur an additional penalty that would not have been encountered had the tail colour character been omitted.

4.4 This may not be desirable in neomorphic characters

The more general rule is that any tree that reconstructs the same state arising twice, independently, in an ontologically dependent character will incur a penalty relative to one that reconstructs that same state arising once. With transformational characters, this is often a desideratum – as discussed above. In certain neomorphic characters, however, it may not be desirable to penalise trees in which the *absence* of a character arises multiple times. Let us imagine that there is a biological reason to believe that tails in a particular group lacked poisoned barbs when they first evolved: that is, poisoned barbs are an evolutionary innovation that can only be added to a tail once a tail is already present.

Table 4.4: A neomorphic character, poison barbs, present in some but not all tails

	A	B	C	D	E	F	G	H	I
Tail: (0), absent; (1), present	0	0	0	1	1	1	1	1	1
Tail, poison barbs: (-), inapplicable; (0), absent; (1), present	-	-	-	0	0	0	0	1	1

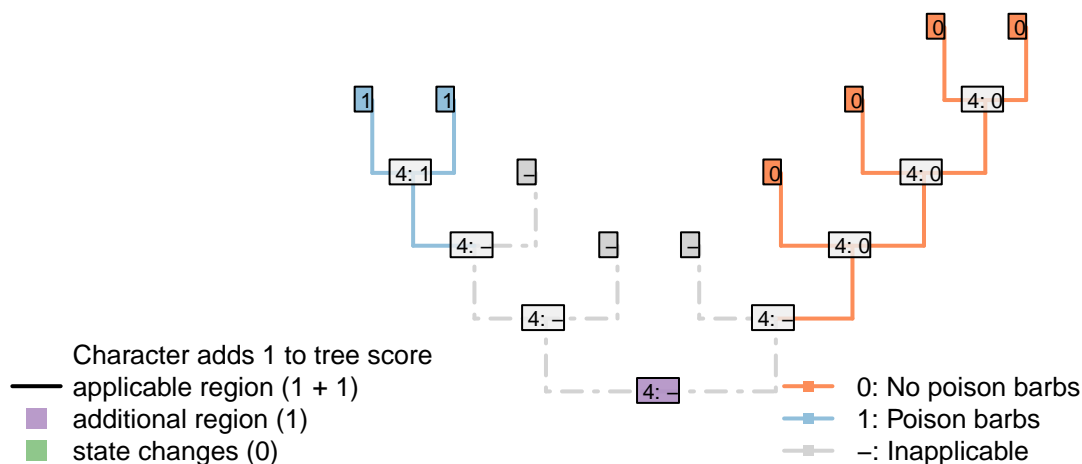


Figure 4.2: One tail with barbs, one without

4.4.1 Three scenarios

The presence of poison barbs obviously contains grouping information – a reconstruction that attribute the presence of poison barbs to a single evolutionary gain in a common ancestor is parsimonious with respect to that character (even if it is less parsimonious with respect to another – e.g. the presence or absence of a tail).

Consider a reconstruction in which a tail evolved twice, and barbs evolved twice. Here, the duplicate origin of barbs (as well as the duplicate origin of the tail) makes this reconstruction less parsimonious.

But what about a situation in which a tail evolved twice, and lacked barbs each time it evolved? Coding this character as transformational penalises the duplicate origin of the state “no poison barbs”, making this reconstruction less parsimonious.

If we expect a tail, when it evolves, to lack barbs, then the second origin of “no barbs” does not represent a homoplasy: it’s not a feature that has evolved twice, but rather an observation that something has *not* evolved twice.

The absence of poison barbs in the two ancestral tail-bearers has been inherited from a common ancestor that did not itself bear tail barbs (by virtue, in this instance, of not bearing a tail). This second non-origination should not, therefore, be penalized in this situation.

This problem has arisen because the inapplicable token has been used in a character that is, in fact, applicable.

The statement “A tail is absent; the tail is red” is not logically consistent, which is why the inapplicable token is necessary. In contrast, the statement “A tail is absent; tail barbs are absent” *is* logically consistent, and the inapplicable token is not necessary. Instead, the ‘absence’ token should be employed instead of the inapplicable:

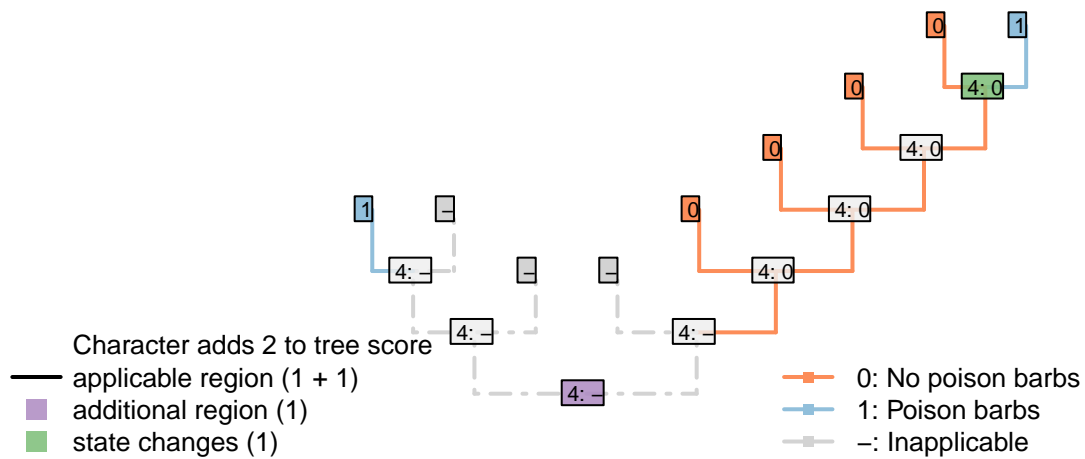


Figure 4.3: Two barb appearances

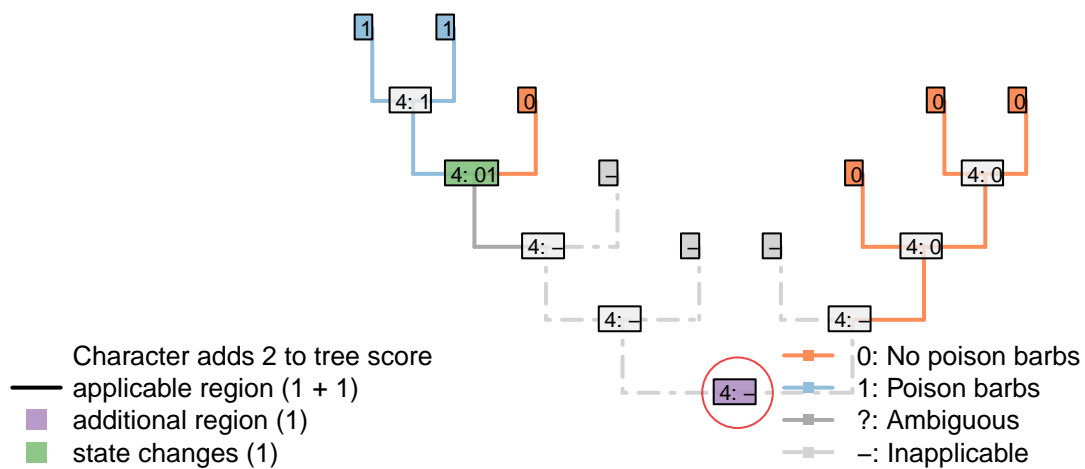


Figure 4.4: Two barbless appearances: second absence is penalized

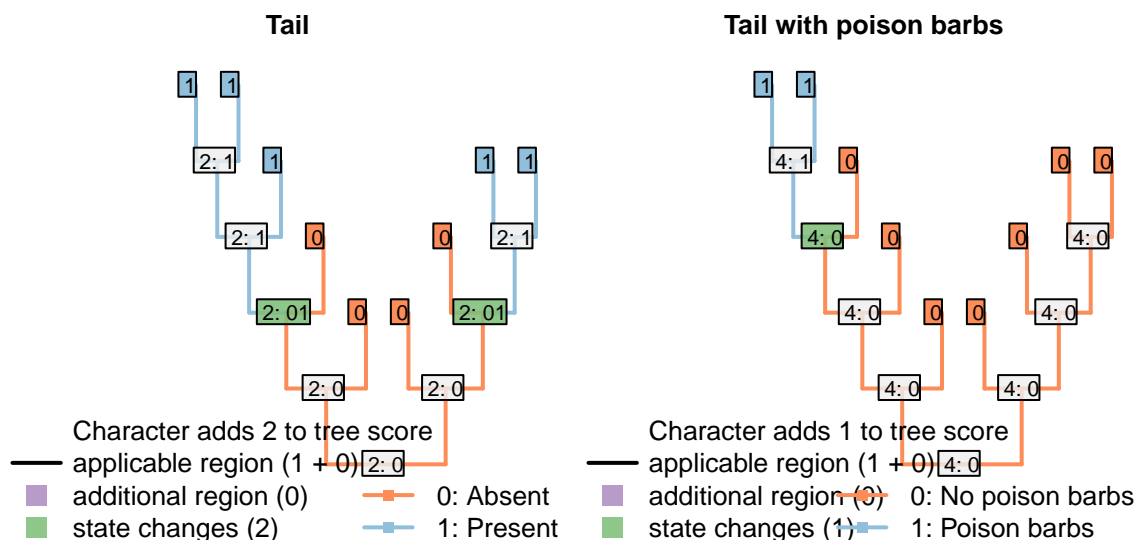


Figure 4.5: Two barbless appearances

Table 4.5: Recommended coding: state 0 reserved for absence; states 1 and 2 used for (transformational) tail colour character.

	A	B	C	D	E	F	G	H	I
Tail: (0), absent; (1), present	0	0	0	1	1	1	1	1	1
Tail, poison barbs: (0), absent; (1), present	0	0	0	0	0	0	0	1	1
Tail, colour: (-), inapplicable; (1), red; (2), blue	-	-	-	1	1	1	2	2	2

The point here is that the inapplicable token ought only to be used in tips where a character description literally does not apply. As an example, De Laet (2017) contends that the character “Tail: absent/present” is inapplicable in an angiosperm. We disagree. Angiosperms do not have tails. “Tail” should be coded as absent in angiosperms.

One way to emphasize this distinction in character matrices is to reserve the 0 token to denote absence, and denoting states of transformational characters using the positive integers:

One implication of this coding strategy is that the loss of a tail (a single evolutionary event) causes the loss of all contingent characters – characters are not independent.

If a poisoned tail was present in a lineage, then lost, then re-gained, would one expect the re-gained tail to also re-gain its poisoned barbs? One could spend some time evaluating whether this behaviour has a biological underpinning, or whether it is desirable – is a reconstruction that invokes the loss of a complex tail more parsimonious than one that invokes the loss of a simple tail?

Indeed, it would be straightforward to construct an algorithm that does not penalise losses where the loss corresponds to the inferred loss of a parent character.

The underlying issue, however, is that both parsimony and the Mk model assume character independence; it is perhaps more fruitful to focus effort on developing models of evolution that take proper account of character non-independence.

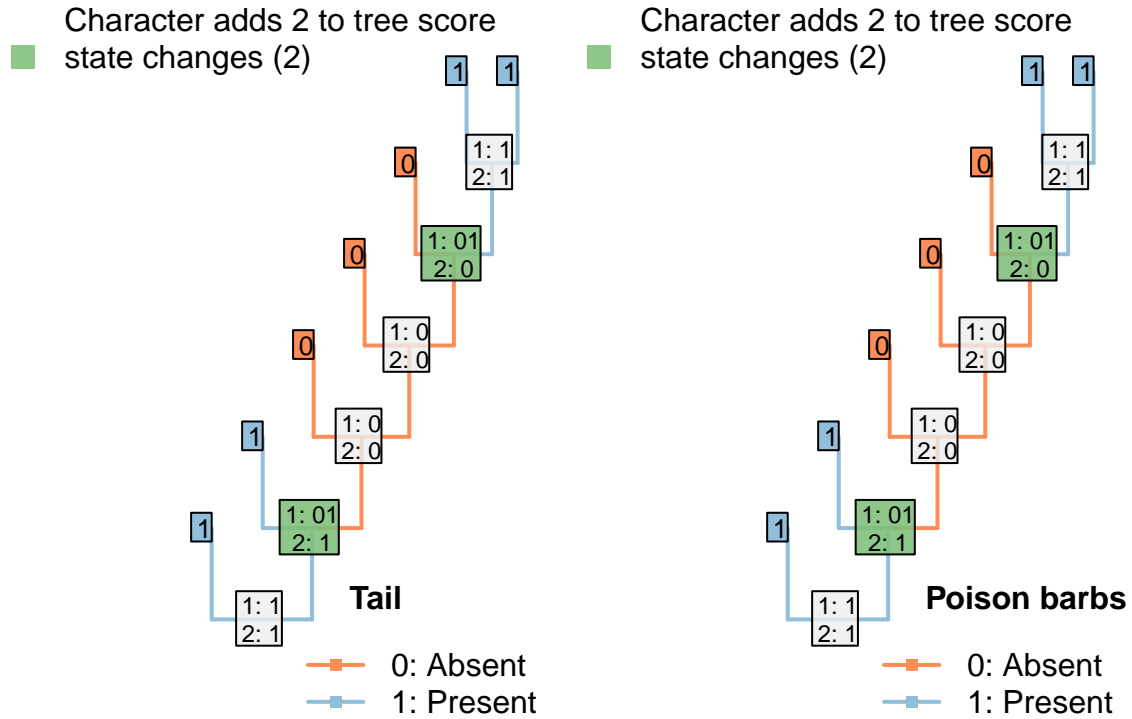


Figure 4.6: Presence of a tail and presence of poison barbs will have the same distribution if all tails have poison barbs. Loss and subsequent re-gain of a tail implies the same loss and re-gain of barbs.

Table 4.6: Absences treated as transformational characters

	A	B	C	D	E	F	G
Tail: (0), absent; (1), present	0	0	0	1	1	1	1
Tail, margin: (-), inapplicable; (1), smooth; (2), serrated	-	-	-	{12}	{12}	1	1
Tail, glow-in-the-dark pigment: (-), inapplicable; (1), absent; (2), present	-	-	-	{12}	{12}	1	1
Tail, ability to generate electricity: (-), inapplicable; (1), absent; (2), present	-	-	-	{12}	{12}	1	1

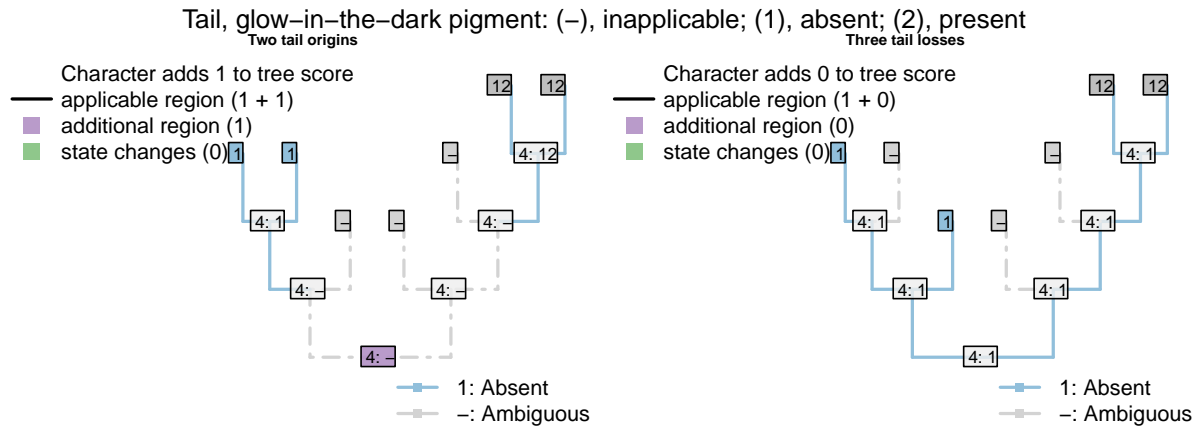
4.4.2 Does absence contain phylogenetic information?

In some cases, the absence of a feature (e.g. serrations) may represent a transformational character and should thus be coded as such. But this decision is significant, and merits careful thought. A researcher may or may not be justified in including properties of a tail that occur in only one, or even in none, of the taxa of interest, for if absence is informative for parsimony, then such characters will influence tree topology: parsimony uninformative characters inform parsimony.

Note that each of the unobserved (i.e. always-absent) characters provides evidence against independent origins of the tail, in favour of independent losses:

Table 4.7: Recommended coding: Absences treated as neomorphic characters were appropriate

	A	B	C	D	E	F	G
Tail: (0), absent; (1), present	0	0	0	1	1	1	1
Tail, margin: (1), smooth; (2), serrated	-	-	-	{12}	{12}	1	1
Tail, glow-in-the-dark pigment: (-), inapplicable; (0), absent; (1), present	0	0	0	{01}	{01}	0	0
Tail, ability to generate electricity: (-), inapplicable; (0), absent; (1), present	0	0	0	{01}	{01}	0	0



Under the simple matrix presented above, the left-hand tree receives a score of five (two independent gains of the tail, plus the three ontologically dependent characters with an additional step each), whereas the right-hand tree scores but three (three independent losses of the tail; no steps in the ontologically dependent characters), making it more parsimonious.

If the three ontologically-dependent characters were coded as ‘absent’ (instead of inapplicable) when the tail was absent, then the left-hand tree would be preferred (with a score of 2 vs. 3).

The two trees are equally parsimonious (both scoring three) if tail margin is treated as a transformational character (inapplicable when tail absent) and the other characters are treated as neomorphic (absent when tail absent).

Chapter 5

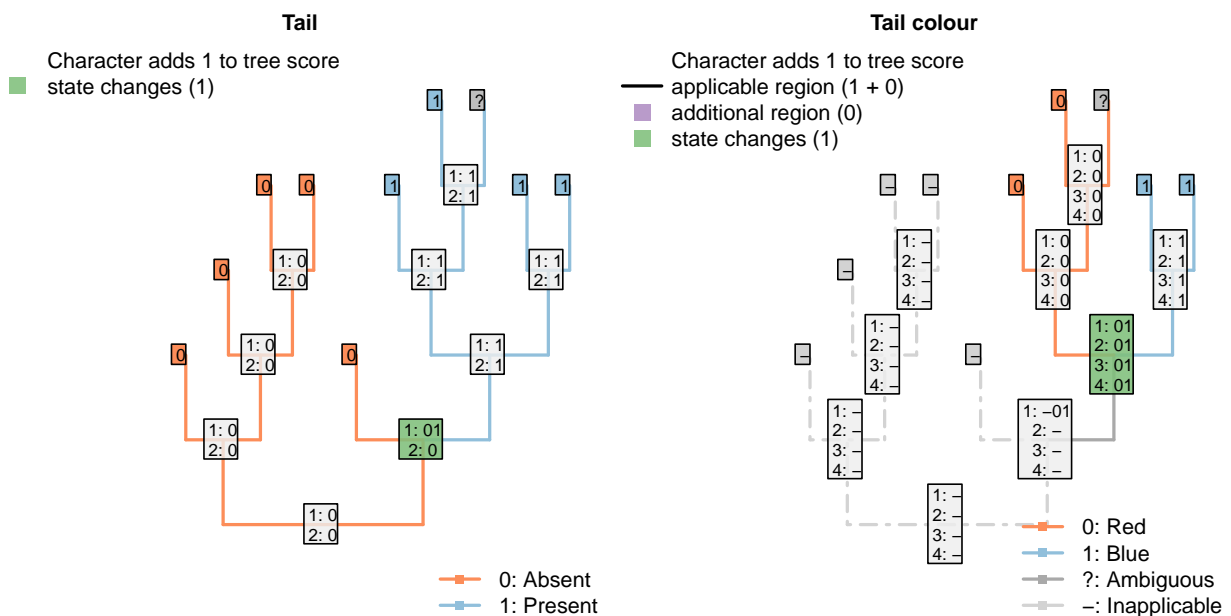
Coding ambiguity

Ambiguous data does not pose a problem for the algorithm, but the nature of the ambiguity must be considered when scoring a character.

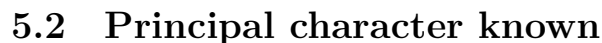
5.1 Principal character ambiguous

If it's not clear whether or not a taxon has a tail, then tail colour should be coded as ?, denoting that any possible token (including the inapplicable token) may be the most parsimonious for the tail.

In trees in which the tail can be reconstructed as present, the ambiguous tip will be reconstructed as having a tail of the appropriate colour:



In trees in which the tail cannot be reconstructed as present without inferring a homoplasious origin, the tail colour will be reconstructed as inapplicable:



5.2.1 Subordinate character has finite states

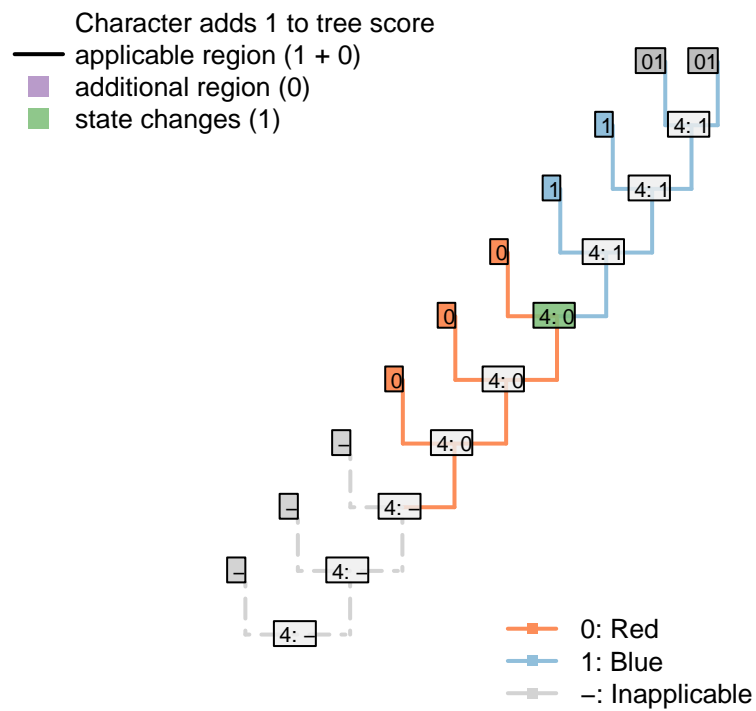
For example:

Tail margin: (0), smooth; (1), serrated.

5.2.2 Subordinate character may have unobserved states

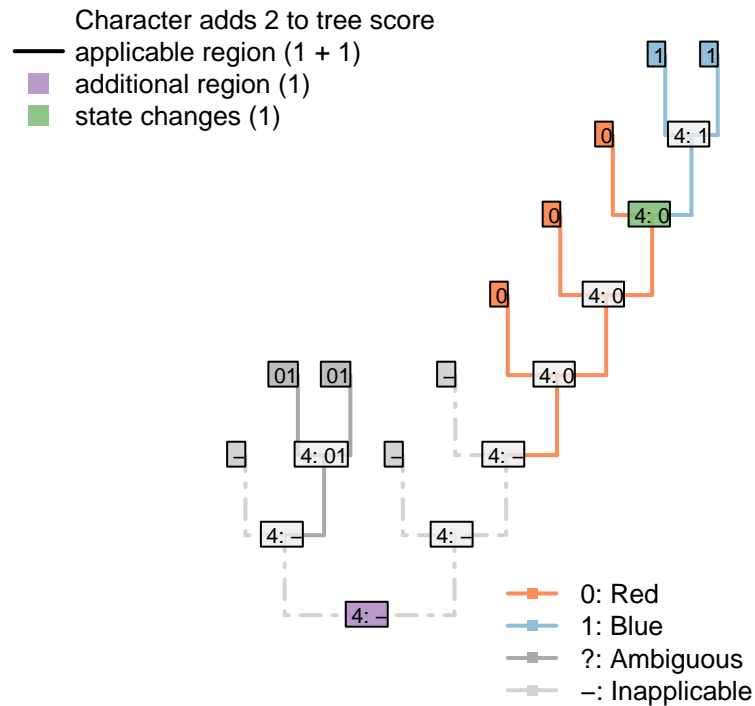
Tail colour: (0), red; (1), blue.

Coding it as {01} would be appropriate if the tail was known to certainly be homologous with other tails in the dataset, in which case it would be most parsimonious to assume that the tail colour is the same colour as the ancestor of the tip, which was necessarily either red or blue.



But if, as will more often be the case, homology of the tails is not known *a priori*, then it is possible that this taxon has a tail that is not homologous with any other tail whose colour has been observed.

In this case, coding the tail colour as {01} denotes that the tail is the same colour as a tail that has already been observed. This means that the independent origin of the tail also represents an independent origin of this particular colour – and hence an instance of homoplasy.

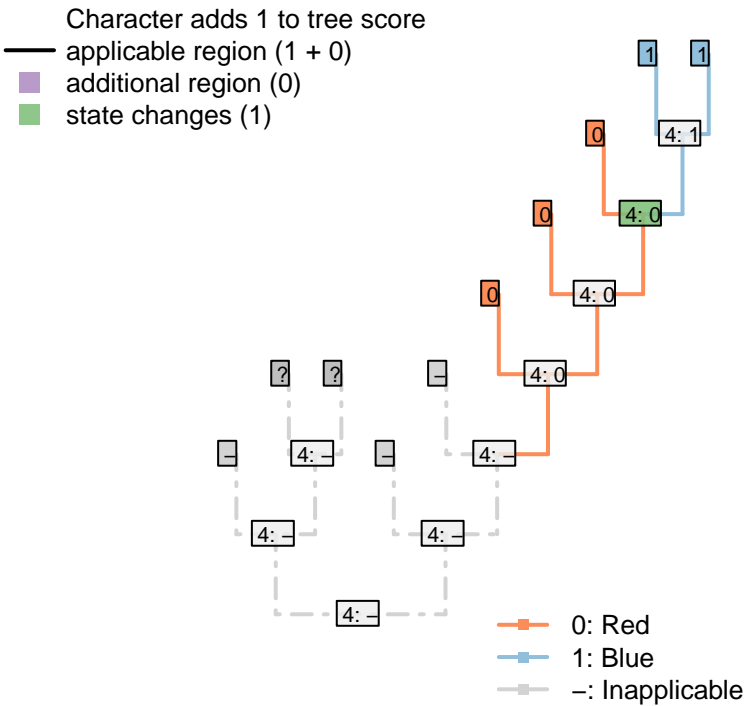


Coding the tail colour as ? allows the possibility that the independently-evolved tail has a different colour to the tails already observed – green, perhaps. Reconstructing the tail colour as a colour that has not already been observed avoids an instance of homoplasy, and is therefore more parsimonious.

In the case that the unknown tail evolved independently and was green, the original character formulation – which only provides tokens for red and blue tails – cannot be applied and is thus inapplicable. Our algorithm will thus reconstruct tail colour as being inapplicable in such a taxon.

Table 5.1: Recommended coding for unknown contingent characters

	Present	Unknown	Absent
Tail: (0), absent; (1), present.	1	?	0
Tail margin: (0), smooth; (1), serrated.	{01}	?	-
Tail colour: (0), red; (1), blue.	?	?	-



5.3 Recommendation

We therefore recommend the following coding schema for ambiguous tips where the tail is known to be present, ambiguous, or known to be absent:

“Tail margin” represents a character that can only take the states observed (smooth or serrated), whereas tail colour represents a character that may take an unobserved state (e.g. green).

Chapter 6

Examples

This vignette describes how the algorithm approaches some example trees. We follow the example of a tail coded using two characters:

Tail: (0), absent; (1), present;

Tail colour: (0), red; (1), blue.

6.1 Some caterpillars

First we'll address some pectinate "caterpillar" trees, in which eight taxa have tails (and eight do not), four of which are red, four of which are blue.

An optimal tree with this character invokes a single origin of the tail, and a single change in tail colour, thus incurring a score of two. Here is one example:

If we insist that the tail evolves twice, then the best score is accomplished by reconstructing a different colour of tail in each of the two regions in which the tail is present. On a caterpillar tree, this means the loss of a tail that has one colour, and an independent innovation in a tail-less taxon of a tail that has a different colour:

Under the parsimony criterion, it is considered less optimal if a tail, when it re-evolves, happens to independently re-evolve a colour that has already been observed – "blueness" has evolved twice on the following tree, meaning that the second innovation of "blueness" represents an instance of homoplasy.

6.2 Three equally suboptimal alternatives

The following three trees differ in the number of innovations of the tail that are implied, and the number of changes in tail colour. All are equally parsimonious.

Under the first, our algorithm reconstructs the tail as ancestrally present, being lost on edge 2, gained on edge 5, lost in tips H and I, lost on edge 11, and gained on edge 14 (a total of six homoplasies). It further reconstructs independent, homoplastic origins of tail redness on edge 5, tail blueness on edge 14, and a change in tail colour from red to blue somewhere between edges 7 and 9 (three homoplasies).

In the second, our algorithm reconstructs the tail as ancestrally present, being lost in tips B, D, E, H, and I, and on edge 11, before being independently gained on edge 14 (a total of seven homoplasies). It further reconstructs an independent, homoplastic origins of tail blueness on edge 14, and a change in tail colour from red to blue somewhere between edges 7 and 9 (two homoplasies).



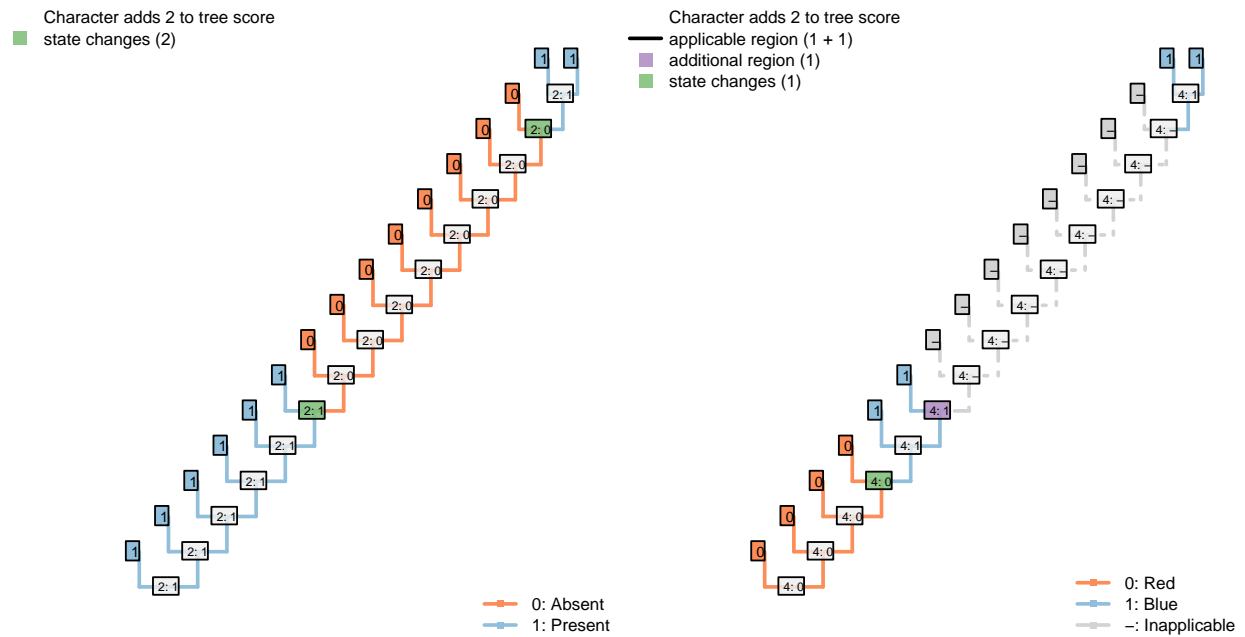


Figure 6.3: Tree A: Total score 4

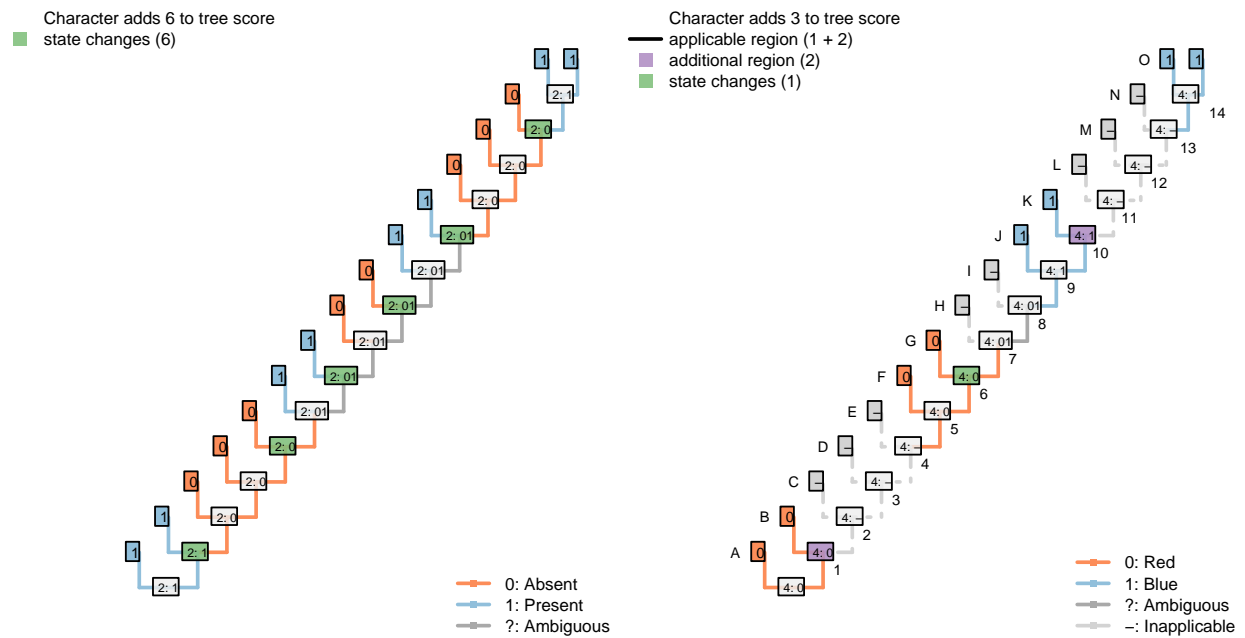


Figure 6.4: Tree B: Total score 9

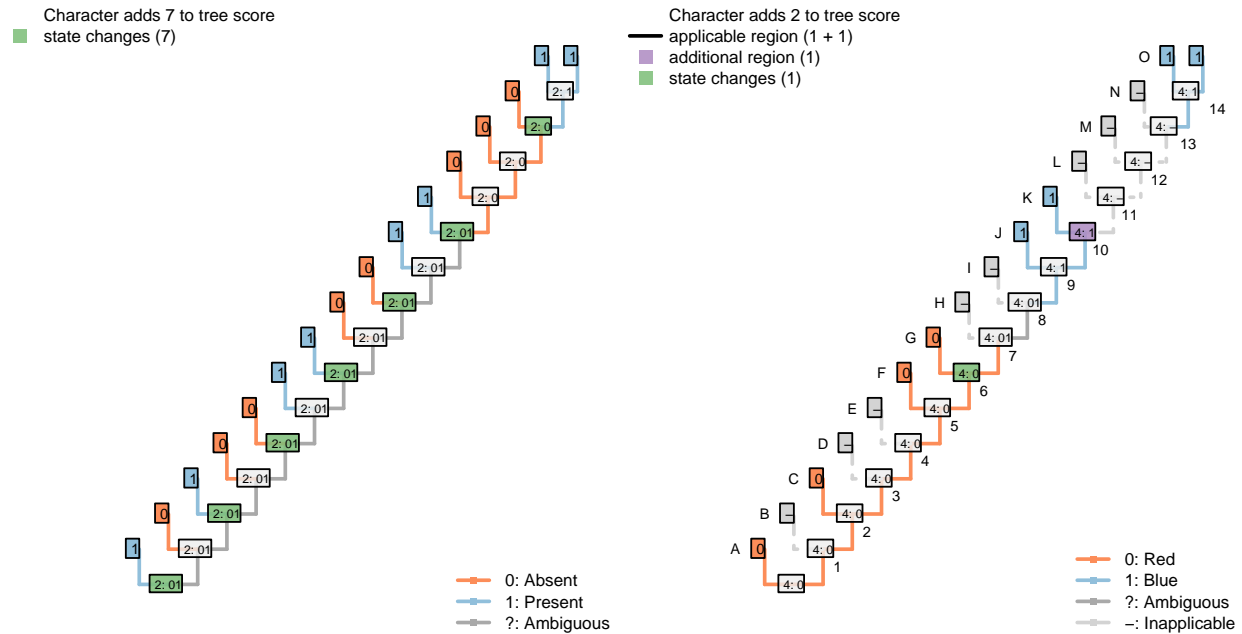


Figure 6.5: Tree C: Total score 9

Table 6.1: Coding

	A	B	C	D	E	F	G	H
Tail: (0), absent; (1), present.	0	1	1	?	0	0	1	1
Tail, colour: (1), red; (2), blue.	-	1	1	?	-	-	2	2

The third configuration reconstructs the tail as ancestrally present, being lost in tips B, D, F, H, J, L, N and P (a total of eight homoplastic losses). It further reconstructs a single change in tail colour from red to blue on edge 8.

6.3 A better caterpillar tree

The tree below obtains a better score than any of the previous three: it implies a loss of the tail at edge 2, a gain at edge 6, a loss at edge 10, and a gain at edge 14; it invokes a homoplastic origin of redness at edge 6, one of blueness at edge 14, and a change in colour at edge 8, for a combined score of 7.

6.4 De Laet's caterpillars

De Laet (2017) identifies a corner case in which our algorithm (Brazeau et al., 2018) will not reconstruct every equally-parsimonious character reconstruction. Below is a simplified version of his example:

When optimising tail colour, we reconstruct the tail as present at all internal nodes, with independent losses of the tail in each of the three tailless taxa (i.e. edges 1, 9, 11).

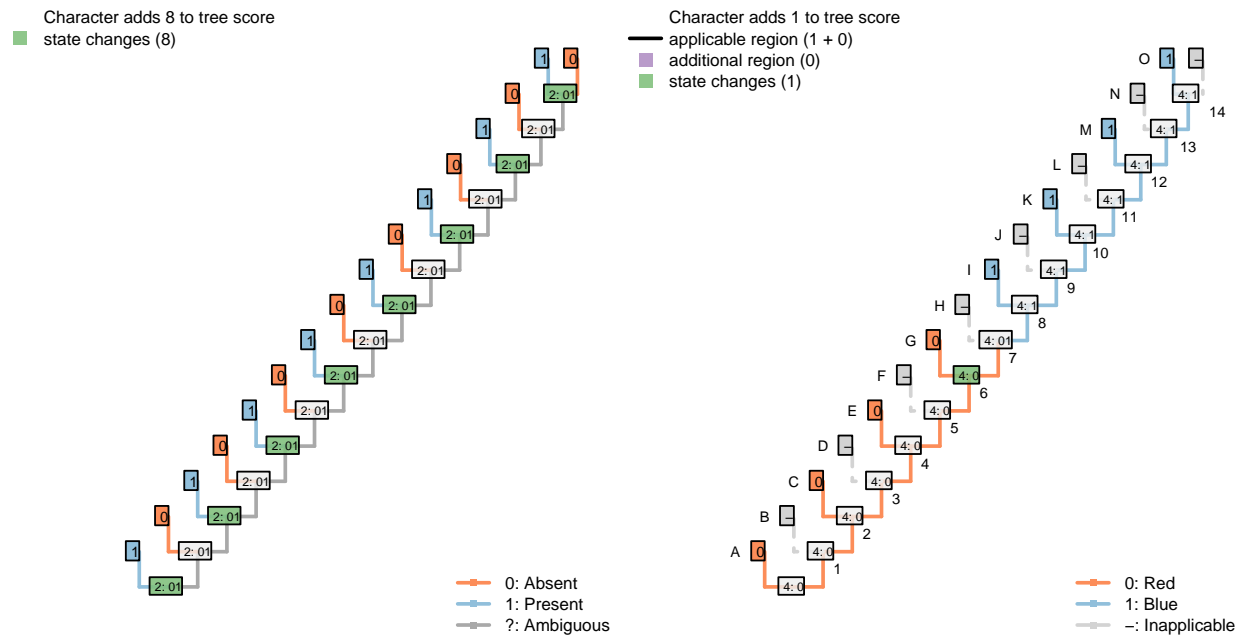


Figure 6.6: Tree D: Total score 9

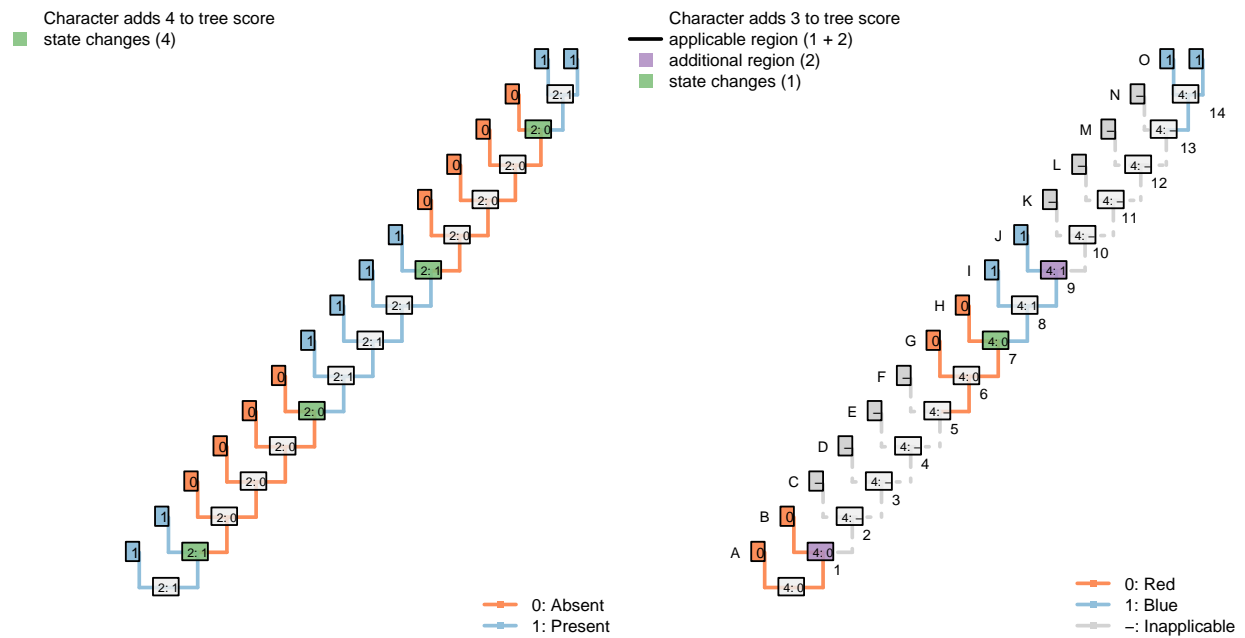


Figure 6.7: Tree E: Total score 7

Bibliography

- Ingi Agnarsson and Jeremy A. Miller. Is ACCTRAN better than DELTRAN? *Cladistics*, 24(6):1032–1038, 2008. doi:10.1111/j.1096-0031.2008.00229.x.
- Martin D. Brazeau, Thomas Guillerme, and Martin Ross Smith. An algorithm for morphological phylogenetic analysis with inapplicable data. *Systematic Biology*, 2018. doi:10.1101/209775. URL <https://www.biorxiv.org/content/early/2017/10/26/209775>.
- Jan De Laet. Parsimony and the problem of inapplicables in sequence data. In V.A Albert, editor, *Parsimony, Phylogeny and Genomics*, pages 81–116. Oxford University Press, 2005.
- Jan De Laet. Parsimony analysis of unaligned sequence data: maximization of homology and minimization of homoplasy, not minimization of operationally defined total cost or minimization of equally weighted transformations. *Cladistics*, 31:550–567, 2015. doi:10.1111/cla.12098.
- Jan De Laet. A note on Brazeau et al.’s (2017) algorithm for characters with inapplicable data, illustrated with an analysis of their Fig. 3d using anagallis, a program for parsimony analysis of character hierarchies. 2017. doi:10.13140/RG.2.2.31309.54245.
- James S. Farris, Arnold G. Kluge, and M J Eckardt. A numerical approach to phylogenetic systematics. *Systematic Biology*, 19(2):172–189, 1970. doi:10.2307/2412452.
- Walter M. Fitch. Toward defining the course of evolution: minimum change for a specific tree topology. *Systematic Biology*, 20(4):406–416, dec 1971. ISSN 1063-5157. doi:10.1093/sysbio/20.4.406. URL <http://sysbio.oxfordjournals.org/content/20/4/406.short>.
- Pablo A Goloboff and Santiago A Catalano. TNT version 1.5, including a full implementation of phylogenetic morphometrics. *Cladistics*, 32(3):221–238, 2016. doi:10.1111/cla.12160. URL <http://doi.wiley.com/10.1111/cla.12160>.
- Wayne P. Maddison. Missing data versus missing characters in phylogenetic analysis. *Systematic Biology*, 42(4):576–581, 1993. doi:10.1093/sysbio/42.4.576.
- F. Ronquist, M. Teslenko, P. van der Mark, D. L. Ayres, A. Darling, S. Hohna, B. Larget, L. Liu, M. A. Suchard, and J. P. Huelsenbeck. MrBayes 3.2: efficient Bayesian phylogenetic inference and model choice across a large model space. *Systematic Biology*, 61(3):539–42, 2012.
- David Sankoff and R. J. Cedergren. Simultaneous comparison of three or more sequences related by a tree. In David Sankoff and J. B. Kruskal, editors, *Time warps, string edits and macromolecules: the theory and practice of sequence comparison*, pages 253–263. Addison-Wesley, Reading, Massachusetts, 1983.
- David Sankoff and Pascale Rousseau. Locating the vertices of a steiner tree in an arbitrary metric space. *Mathematical Programming*, 9(1):240–246, 1975. doi:10.1007/BF01681346.
- Paul C. Sereno. Logical basis for morphological characters in phylogenetics. *Cladistics*, 23(6):565–587, 2007. doi:10.1111/j.1096-0031.2007.00161.x.

- Alexandros Stamatakis. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, 30(9):1312–1313, 2014. doi:10.1093/bioinformatics/btu033. URL <http://bioinformatics.oxfordjournals.org/content/30/9/1312.abstract>.
- David L Swofford. PAUP*: Phylogenetic analysis using parsimony (and other methods) 4.0. B5. 2001.
- David L. Swofford and Wayne P. Maddison. Reconstructing ancestral character states under Wagner parsimony. *Mathematical Biosciences*, 87(2):199–229, dec 1987. doi:10.1016/0025-5564(87)90074-5.
- Ward C. Wheeler. The triangle inequality and character analysis. *Molecular Biology and Evolution*, 10(3): 707–712, 1993. ISSN 07374038.