

AVL Trees

AVL Trees

The **height** of a tree is length of the longest path from the root of a tree to one of its leaves. It can be calculated as follows:

```
height Leaf = 0
height (Node l v r) = (max (height l) (height r)) + 1
```

The **balance factor** of a binary tree node (Node l v r) is $height(l) - height(r)$.

A tree is **height-balanced** if at every node, the heights of the two subtrees differ by no more than one – i.e., if the balance factor at every node is in the set $\{-1, 0, 1\}$.

A binary search tree (BST) that is height-balanced is called an **AVL Tree** (named after its inventors, Adel'son-Vel'skii and Landis).

Maximum Height of an AVL Tree

What is the maximum height of a AVL tree with n nodes? To determine this, we first answer an equivalent question: what is the minimal number of nodes in a height-balanced tree with height h ? Let $N(h)$ denote this value.

$N(0) =$

$N(1) =$

Recurrence for general case of $N(h)$:

This is similar to Fibonacci numbers:

```
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

	0	1	2	3	4	5	6	7	8
N									
fib									

Note that $N(h) = fib(h + 2) - 1$.

It is possible to show¹ that $fib(n) = \frac{\phi^n - (1-\phi)^n}{\sqrt{5}}$, where $\phi = \frac{1+\sqrt{5}}{2}$ (the golden ratio). So $N(h) \approx \phi^{h+2}/\sqrt{5}$. This implies that the maximal height for n nodes is $\log_\phi(\sqrt{5}(n+1) - 2) = \Theta(\log_\phi n)$.

Conclusion: *the height of a height-balanced tree (and therefore an AVL tree) is logarithmic in the number of nodes.*

¹See <http://mathforum.org/dr.math/problems/bauer1.28.96.html> for a derivation.

What Happens to Height Balance on Insertion?

Suppose that T is an AVL tree with height h , and T' is the result of inserting v into T . Here we investigate whether T' is height-balanced.

Observe that $height(T')$ is either h or $h + 1$.

Suppose that T is a leaf. What is the height of T' ? Is T' height-balanced?

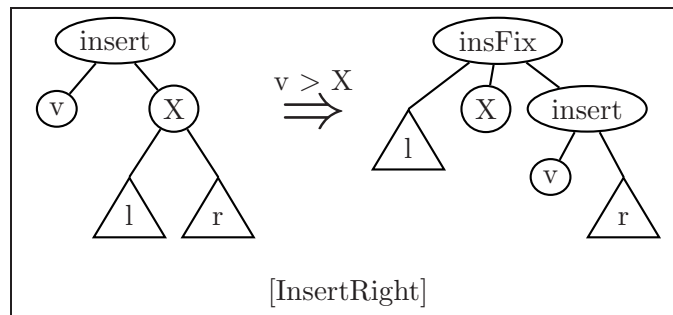
Suppose that T is a node (Node $l \ x \ r$):

- If $v = x$, what is the height of T' ? Is T' height-balanced?
- If $v > x$ ($v < x$ is symmetric), consider the following cases, where h , h_l , and h_r are the heights of T and its left and right subtrees and h' and h'_r are the heights of T' and its right subtree.

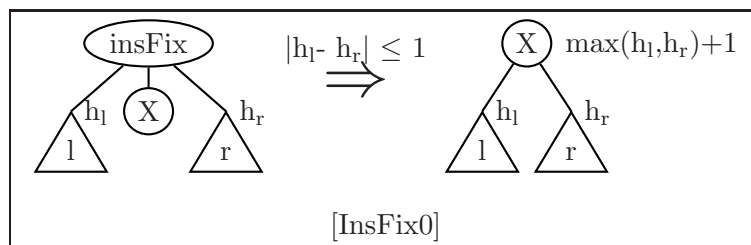
h_l	h_r	h'_r	h'	is T' balanced?
$h - 1$	$h - 1$			
$h - 1$	$h - 1$			
$h - 1$	$h - 2$			
$h - 1$	$h - 2$			
$h - 2$	$h - 1$			
$h - 2$	$h - 1$			

Insertion as Tree Rewriting

We can view insertion as a tree rewriting process that leaves behind a pending “fixup operation”, `insFix`, on the path from the root down to the insertion point. Here is the rewrite rule for the case where the element v being inserted is greater than the root node X of the AVL tree (the less than case is symmetric):

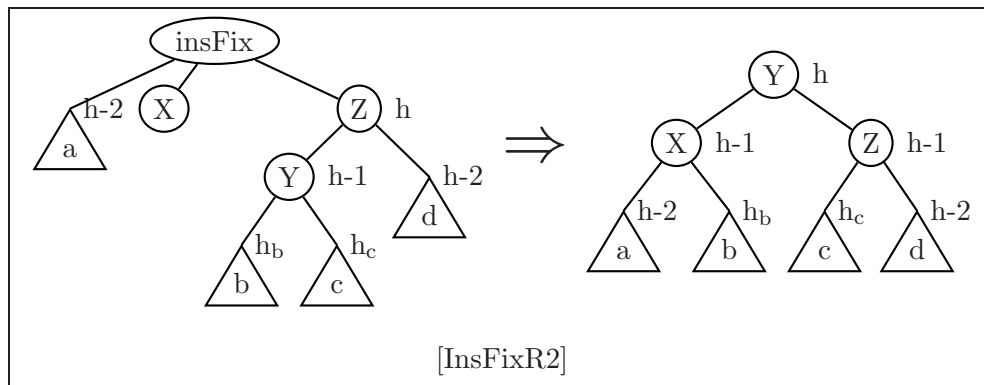
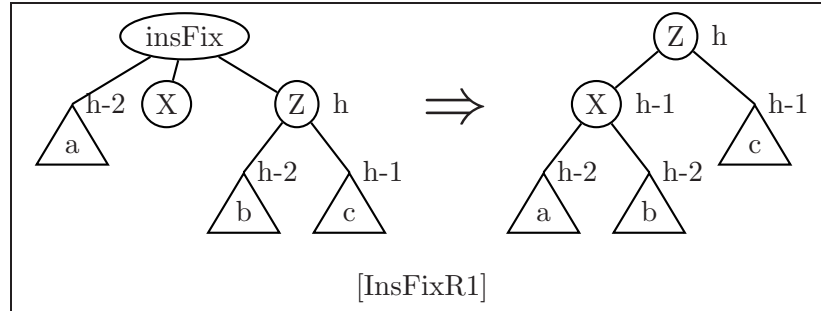


When the heights of the two tree arguments to `insFix` differ by no more than one, it just creates a height-balanced tree node:



Fixing the Bad Insertion Case

In the case where insertion introduces an imbalance, the imbalance can be repaired by **BST rotations**. Below, we show the rotations needed when $height(l) - height(r) = -2$. Symmetric rotations are needed for a difference of 2:

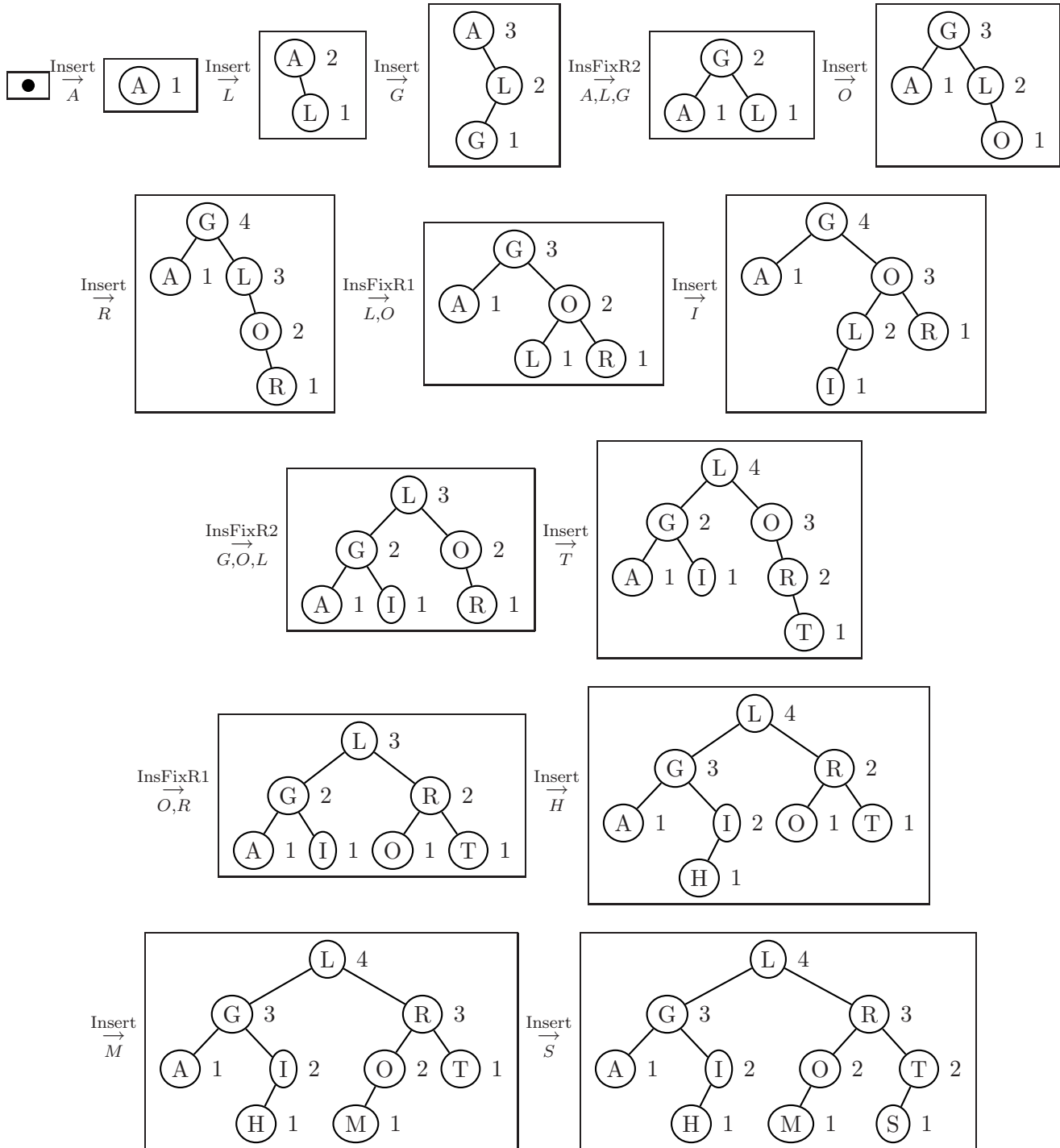


Notes:

- In [InsFixR2], we can have the following combinations of (h_b, h_c) : $(h-3, h-3), (h-3, h-4), (h-4, h-3)$. Regardless of the combination, the heights of the final X and Z nodes are $h-1$.
- Observe that $h' = height(T') = height(T) = h$ when a rotation is performed, so at most one rotation is required on insertion.
- The rules [InsFixR1] and [InsFixR2] handle the cases where the left and right subtrees of node Z have heights $(h-2, h-1)$ and $(h-1, h-2)$. The case $(h-2, h-2)$ is impossible. Why? Because the height of Z has increased to h from $h-1$ due to insertion, and before insertion, Z must, in the worst case, have had two subtrees of height $h-2$. But insertion was performed on only *one* of these subtrees to increase its height. No rotation could have been performed on the subtree rooted at Z, because its height would not have increased.

Insertion Example

Below are some of the intermediate AVL trees created by inserting the letters A L G O R I T H M S from left to right into an initially empty AVL tree. The results of naïve BST insertion (before any fixup operations) and all rotations are shown. Some liberties have been taken in the drawing of the trees; in particular, InsFix nodes are not distinguished from regular binary tree nodes.



What Happens to Height Balance on Deletion?

Suppose that T is an AVL tree with height h , and T' is the result of deleting v from T . Here we investigate whether T' is height-balanced.

Observe that $height(T')$ is either h or $h - 1$.

Suppose that T is a leaf. T' is too, and so is clearly height-balanced.

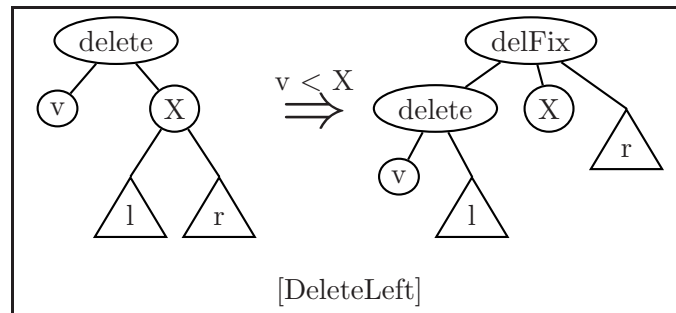
Suppose that T is a node (Node $l\ x\ r$):

- If $v = x$, what is the height of T' ? Is T' height-balanced?
- If $v < x$ ($v > x$ is symmetric), consider the following cases, where h , h_l , and h_r are the heights of T and its left and right subtrees and h' and h'_l are the heights of T' and its left subtree.

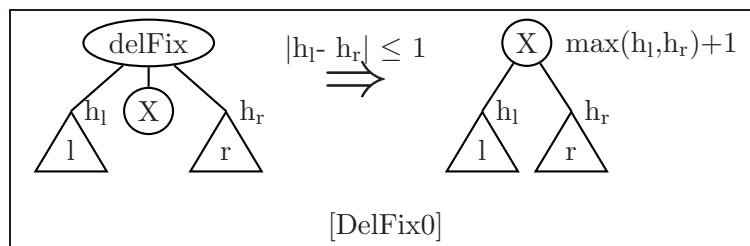
h_l	h_r	h'_l	h'	is T' balanced?
$h - 1$	$h - 1$			
$h - 1$	$h - 1$			
$h - 1$	$h - 2$			
$h - 1$	$h - 2$			
$h - 2$	$h - 1$			
$h - 2$	$h - 1$			

Deletion as Tree Rewriting

As with insertion, deleting v from a tree can be viewed as a tree rewriting process that leaves behind a pending “fixup operation”, `delFix`, on the path from the root down to the deletion point. Here, “deletion point” refers to the position of the fringe node that is actually deleted by the usual BST deletion algorithm, which may be the predecessor (or successor) of the node actually labeled v . Here is the rewrite rule for the case where the element v being inserted is less than the root node X of the AVL tree (the greater than case is symmetric):

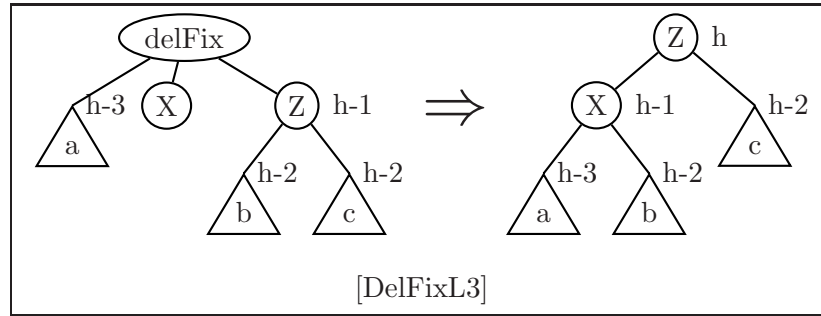


As in `insFix`, when the heights of the two tree arguments to `delFix` differ by no more than one, it just creates a height-balanced tree node:

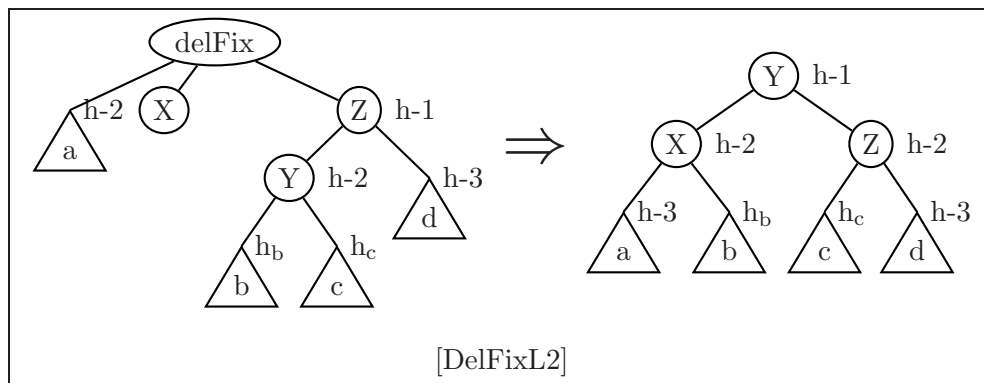
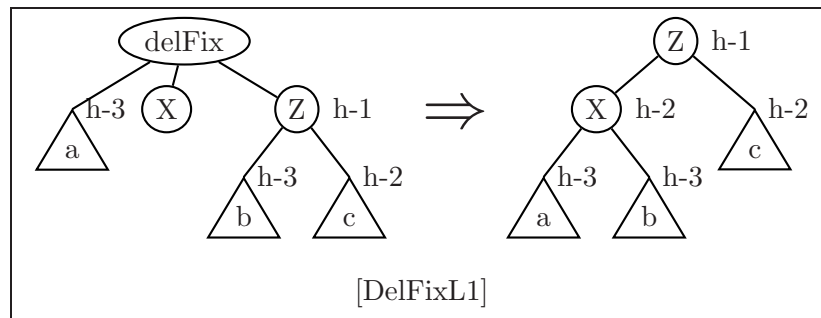


Fixing the Bad Deletion Case

The bad deletion case can be fixed by the same BST rotations as in insertion, *except that* there is one new case (where the two subtrees of Z have the same height):



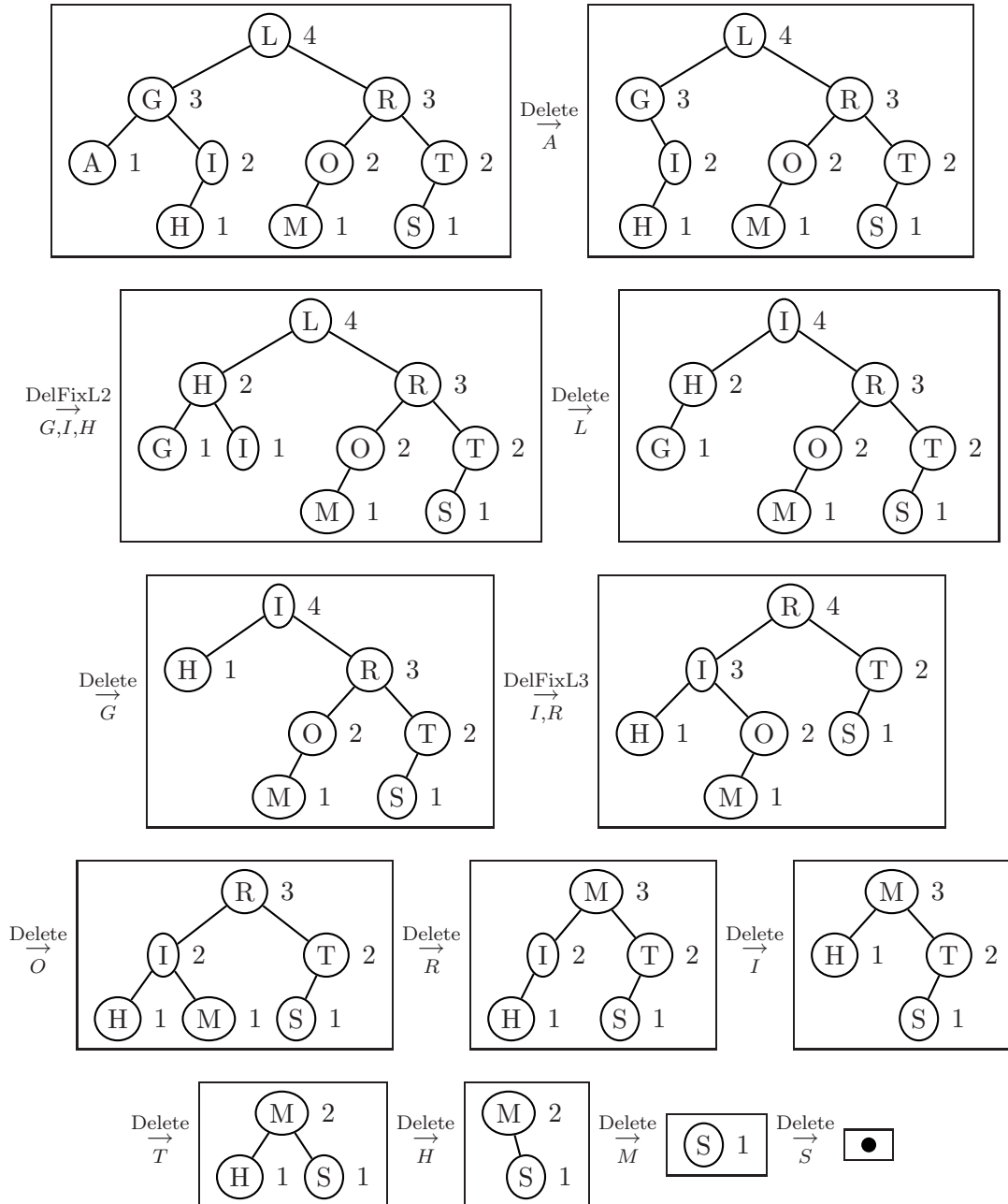
The other two rotations, [DelFixL1] and [DelFixL2], are similar to [InsFixL1] and [InsFixL2], except that the height labels differ (because we are decrementing the height of the left subtree rather than incrementing the height of the right subtree):



Unlike the insertion case, in the deletion case the height of T' after a rotation is not always the same height h as T before the deletion; after [DelFixL1] and [DelFixL2] it is $h - 1$. This implies that multiple rotations may be required for deletion. This will happen in the case where a decrement in height propagates up towards the root via a sequence of rotations involving the last two of the three rules above.

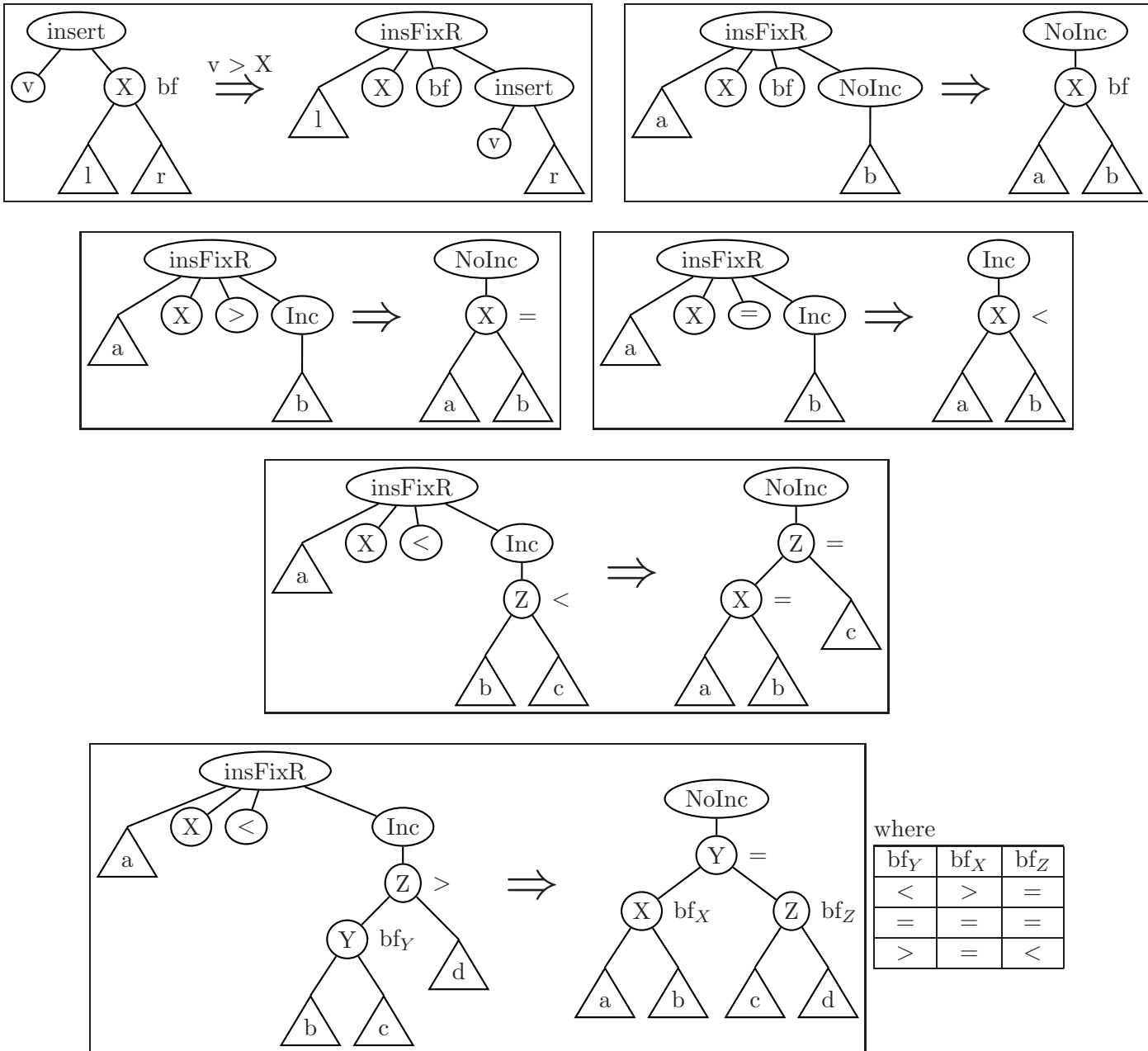
Deletion Example

Below are some of the intermediate AVL trees created by deleting the letters A L G O R I T H M S from left to right from the final tree of the insertion example. The results of naïve BST deletion (before any fixup operations) and all rotations are shown. As in the insertion drawings, some liberties have been taken in the drawing of the trees.



Optimizing AVL Insertion

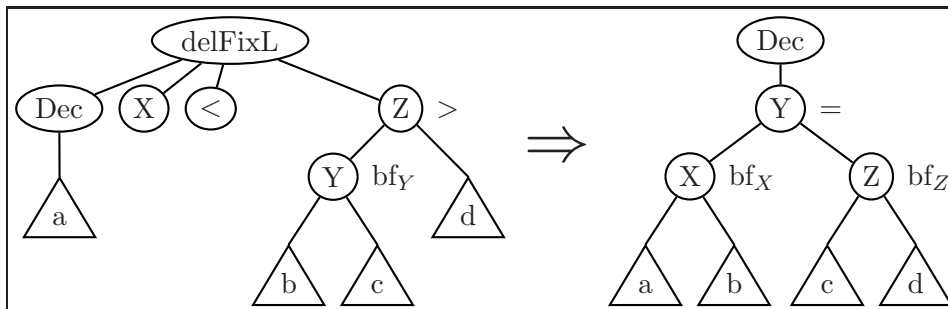
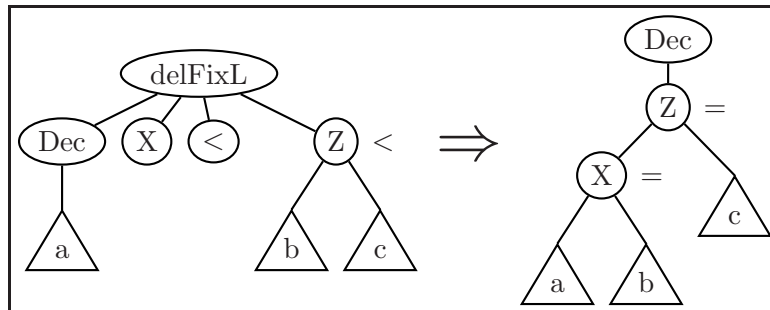
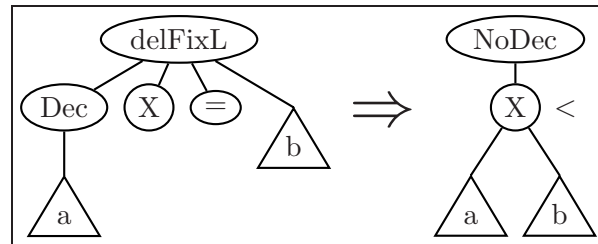
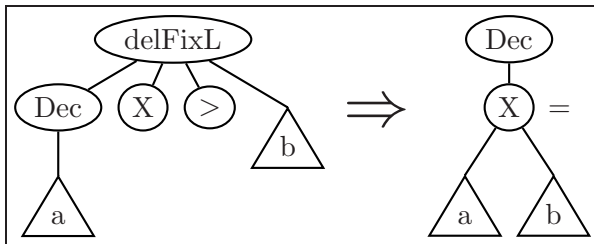
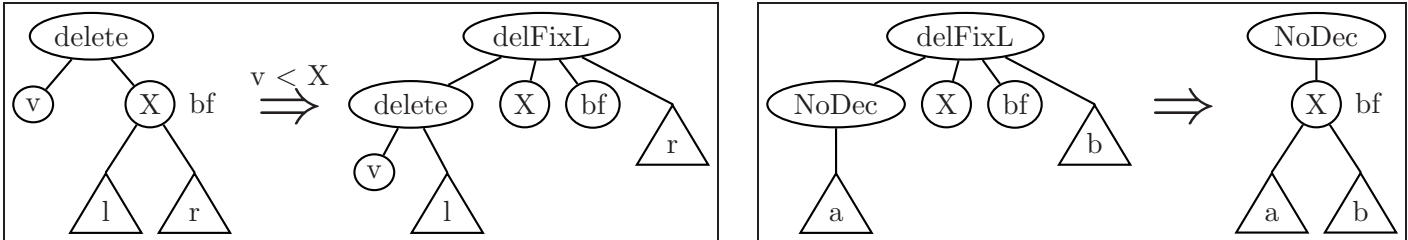
It is not necessary to maintain the heights of the nodes in an AVL tree. All that is needed is the balance factors on each node. We will denote the balance factors -1 , 0 , and 1 as $<$, $=$, and $>$, respectively. During insertion, we need to keep track of whether the height h' of result T' of an insertion into a tree T of height h is incremented ($h' = h + 1$, denoted by **Inc**) or not ($h' = h$, denoted by **NoInc**). Insertion into the right subtree can be described by the following rewrite rules (left subtree insertion is similar):



The balance-factor-based rules above perform the same BST tree rewriting operations as the height-balanced rules studied previously. The first three rules for **insFixR** correspond to three subcases of the height-based rule [InsFix0]. The last two **insFixR** rules correspond to [InsFixR1] and [InsFixR2], respectively.

Optimizing AVL Deletion

During deletion, we need to keep track of whether the height h' of result T' of an insertion into a tree T of height h is decremented ($h' = h - 1$, denoted by **Dec**) or not ($h' = h$, denoted by **NoDec**). Deletion from the left subtree can be described by the following rewrite rules (right subtree deletion is similar). Except for the handling of **Dec/NoDec** and the final rule (which corresponds to [DelFix3]), the rules for left deletion fixup are the same as those for right insertion fixup.



where

bf_Y	bf_X	bf_Z
<	>	=
=	=	=
>	=	<

