Hints and Reminders:

in x→A, x is a bound variable, regardless of what A looks like.
in x→y→A, x and y are both bound variables.

Any variable that appears even once when it is not bound, is free.

in ((x y) (y z)), x and y and z are all free variables.
in y→((x y) (y z)), y is bound, x and z are free.

in (y→(x y) x→(y x)), x and y are both bound and free.

in x→((y→(x y) x→(y x)), y is both bound and free,
but x is only bound.

The name "bound", as in "tied up" makes sense. Free variables are not connected to anything, and their meanings can change depending upon their contexts. Look at the (x x), in which x is free, when it is transported into these three examples:

in x→(x x) the x's become bound - mere parameter names.
in (x→(x x) k) the x's change into k's.
in (x→(x x) (z p)) the x's change into (z p)'s.

x, being free, can be grabbed by any → and turned into something else. On the other hand, a bound variable always represents "the parameter to this function", and no matter where abstractions are copied, their bound variables always retain that meaning.

The next statement a little bit important, but sort-of obvious:

When performing a β-reduction on (x→A B) by replacing every x with a copy of B, make sure you only replace *that particular* x.
If somewhere inside A, there is another x→Something, don't continue the replacement inside "Something", because that is covered by a new local version of x.

The next statement is much more important:

A β-reduction can be performed on (x→A B) by replacing it with a copy of A in which every x is replaced by a copy of B,

ONLY IF

None of the bound variables of A are free in B.

This rule is really just a technical statement of common sense. Suppose it were disobeyed:

in $x \to y \to z \to (x\ (y\ z))$ x and y and z represent that values of parameters that are yet to be supplied.

in $(a\ y)$ a and y are free variables that could yet be taken over

but put the two together and perform a β-reduction:

$$(x \to y \to z \to (x\ (y\ z))\ \ (a\ y))$$

becomes

$$y \to z \to ((a\ y)\ (y\ z))$$

and the y that lives next to the a has completely changed its nature.

Why does that matter?

Let's put that example in an even bigger context:

$$(y \to (x \to y \to z \to (x\ (y\ z))\ \ (a\ y))\ \ k)$$

There are two β-reductions that could be performed next - the outer one that replaces y with k, and the inner one which replaces x with (a y).

The first one yields

$$(x \to y \to z \to (x\ (y\ z))\ \ (a\ k))$$

which can then be reduced to

$$y \to z \to ((a\ k)\ (y\ z))$$

But the second one yields

$$(y \to y \to z \to ((a\ y)\ (y\ z))\ \ k)$$

which can then be reduces to

$$y \to z \to ((a\ y)\ (y\ z))$$

which is completely different.

Hence the rule. $(x \to A\ B)$ can only be β-reduced if none of the bound variables of A are free variables of B.

Fortunately this rule isn't a problem because you are always allowed to do α-conversions, which allow you to change the name of a bound variable to something else.

So you could have some complicated setup, where just before doing a β, you find all the bound variables of A, then search B to make sure they are not free, then change them if they are.

Or you could prevent the problem from ever arising in a very simple way:

Every time you replace any variable with anything, make a new copy of that anything, in which every single bound variable is automatically renamed to a new bound variable that has never been seen before. Even do this when reading the expressions in.

In your trees, an identifier should be represented not just by a letter, but by a letter and a generation number. Never re-use generation numbers, they just go up and up. Whenever you copy an abstraction $x \rightarrow A$ uniformly change the generation number of every appearance of x.