Bottom-Up Construction of Sublist Trees

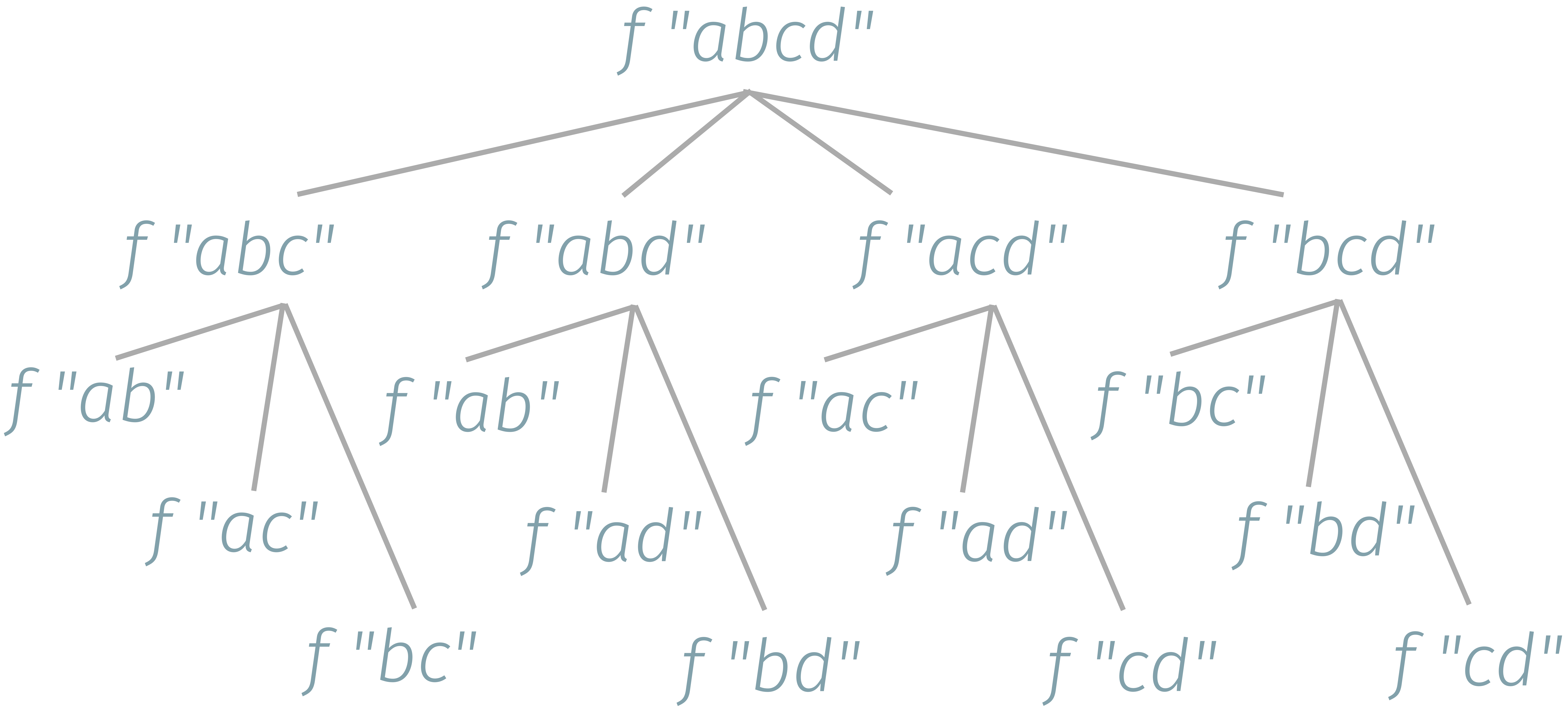
Shin-Cheng Mu

IFIP WG 2.1 Meeting #80 @ Oxford, UK. July 2023

Task

Compute $f :: [X] \to Y$, where $f\ xs$ is defined in terms of values of f on *immediate sublists* of xs.

An *immediate sublist* of xs is one where exactly one element is missing.



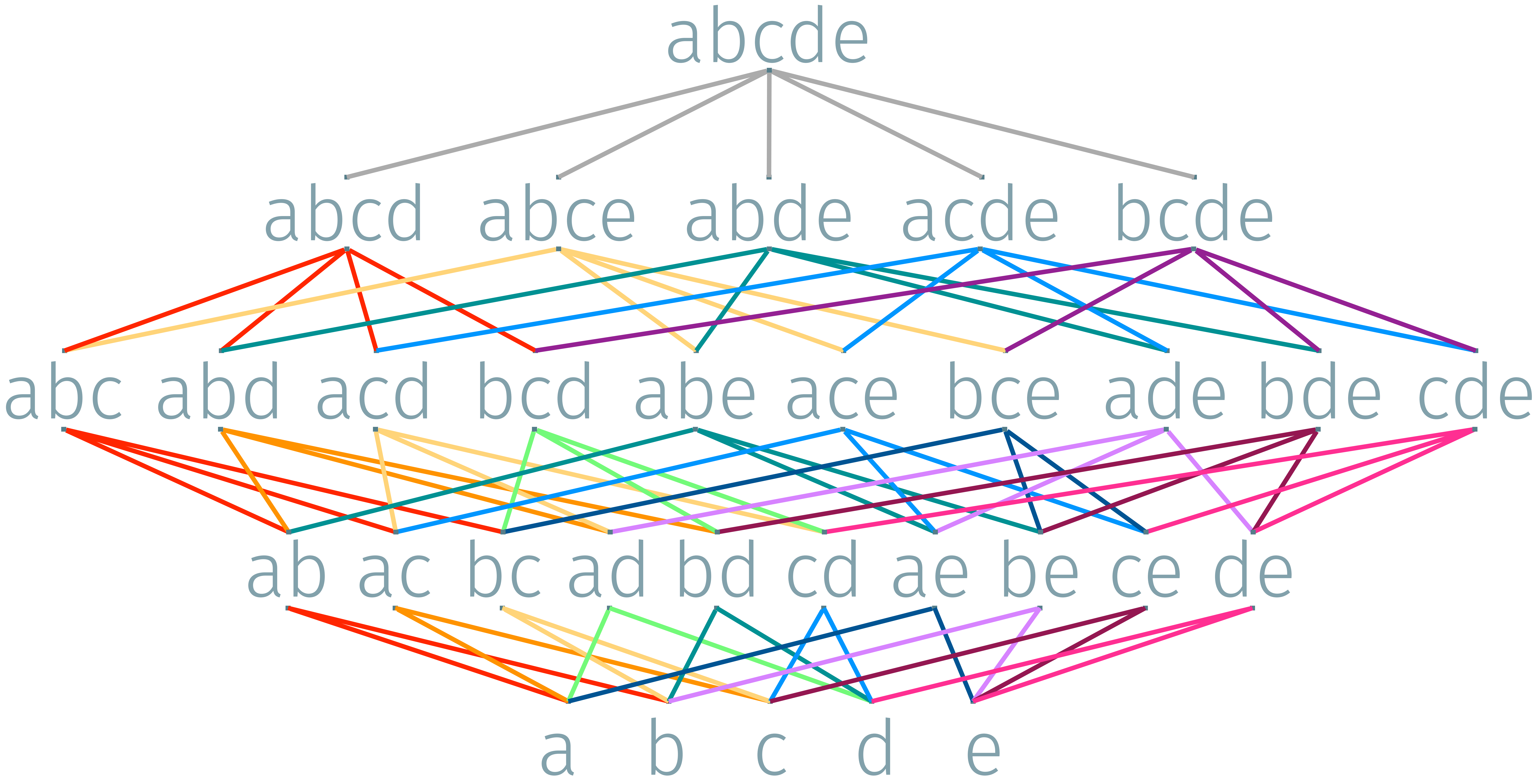
f "abcd"
f "abc"
f "abd"
f "acd"
f "bcd"
f "ab"
f "ac"
f "bc"
f "ab"
f "ad"
f "bd"
f "ac"
f "ad"
f "cd"
f "bc"
f "bd"
f "cd"

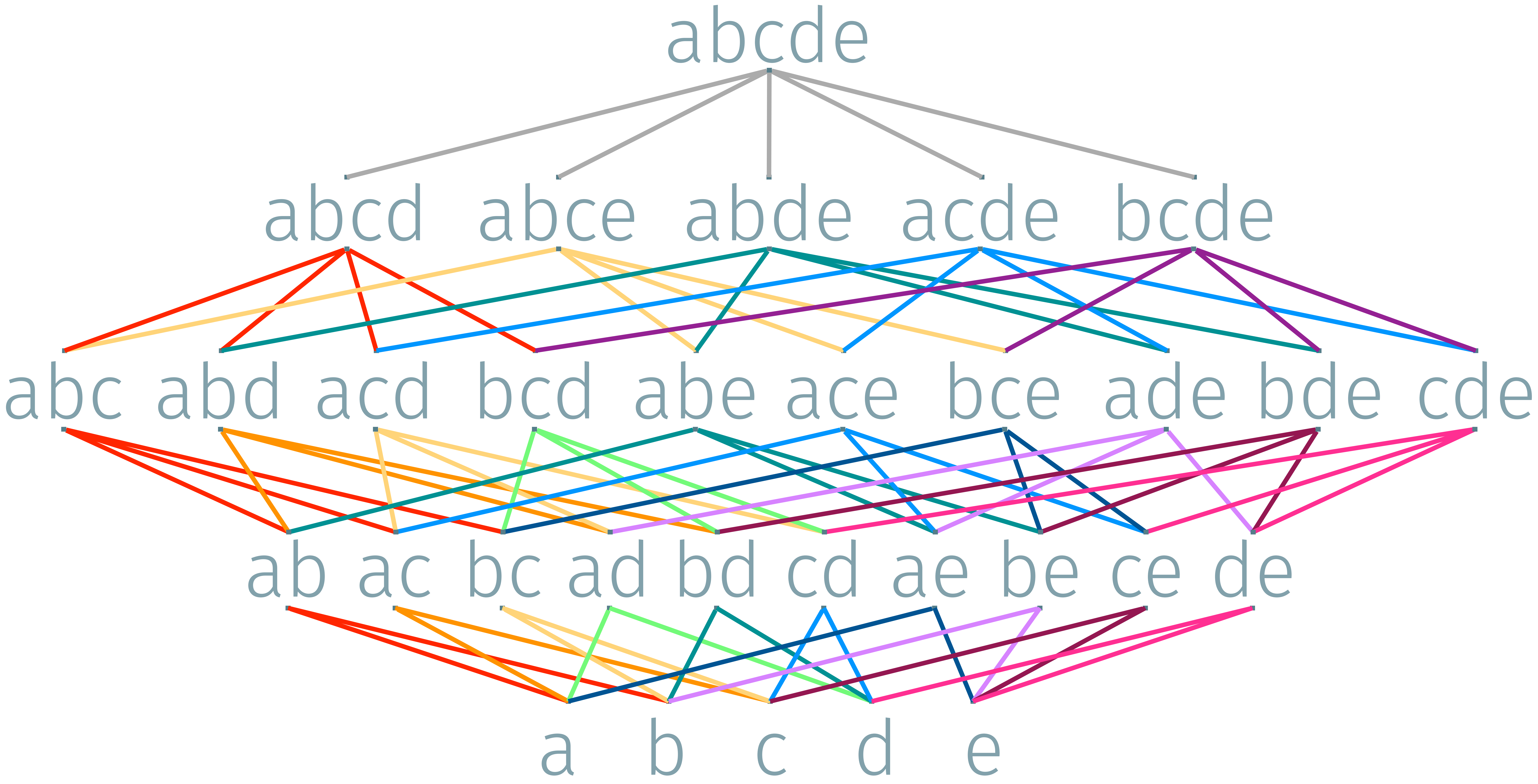
Richard Bird,
Zippy tabulations of recursive functions.
Mathematics of Program Construction, 2008.

$$f\ xs = f\ (\text{init}\ xs) \oplus f\ (\text{tail}\ xs)$$

$$f\ xs = f\ (\text{take}\ (n/2)\ xs) \oplus f\ (\text{drop}\ (n/2)\ xs)$$
$$\quad \text{where}\ n = \#xs$$

$f\ xs =$ … immediate sublists of xs …


abcde
abcd abce abde acde bcde
abc abd acd bcd abe ace bce ade bde cde
ab ac bc ad bd cd ae be ce de
a b c d e


abcde
abcd abce abde acde bcde
abc abd acd bcd abe ace bce ade bde cde
ab ac bc ad bd cd ae be ce de
a b c d e

[f abc, f abd, f acd, f bcd, f abe, f ace ...]

map f_body :: [[Y]] → [Y]

[[f ab, f ac, f bc], [f ab, f ad, f bd],
[f ac, f ad, f cd], [f bc, f bd, f cd], ...]

upgrade :: [b] → [[b]]

[f ab, f ac, f bc, f ad, f bd, f cd, f ae, f be ...]

Goal

To construct *upgrade*...

... or some equivalent function that uses other representations of layers.

But what is its specification?

Immediate Sublists

```
sub :: [a] → [[a]]

sub abcde = [abcd, abce, abde, acde, bcde]
```

We will use this function later.

```
choose :: [a] → Nat → [[a]]
choose _          0       = [[]]
choose xs         k       |  #xs == k = [xs]
choose (xs⧺[x]) (k+1)  =
     choose xs (k+1) ⧺ map (⧺[x]) (choose xs k)
```

```
choose abcde 2 = [ab,ac,bc,ad,bd,cd,ae...]
choose abcde 3 = [abc,abd,acd,bcd,abe,ace...]
```

```
data B a = T a | N (B a) (B a)

choose :: [a] → Nat → B [a]
choose _          0       = T []
choose xs         k       |  #xs == k = T xs
choose (xs⧺[x]) (k+1)  =
     N (choose xs (k+1)) (mapB (⧺[x]) (choose xs k))

mapB :: (a → b) → B a → B b
zipBW :: (a → b → c) → B a → B b → B c
```

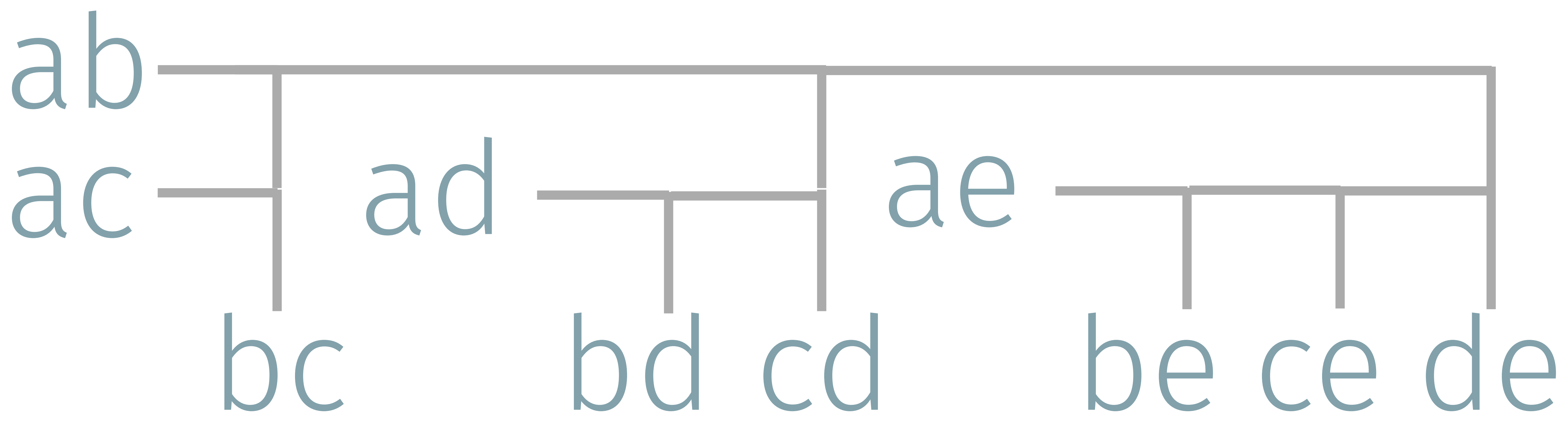
```haskell
data B a = T a | N (B a) (B a)
```

T xs drawn as xs

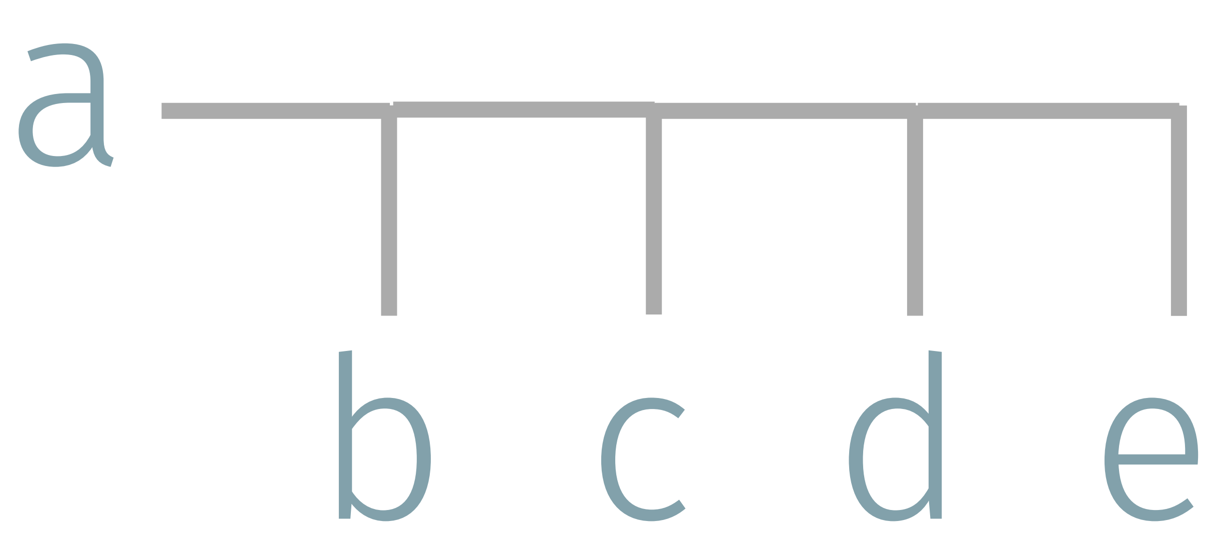
N t u drawn as t

u

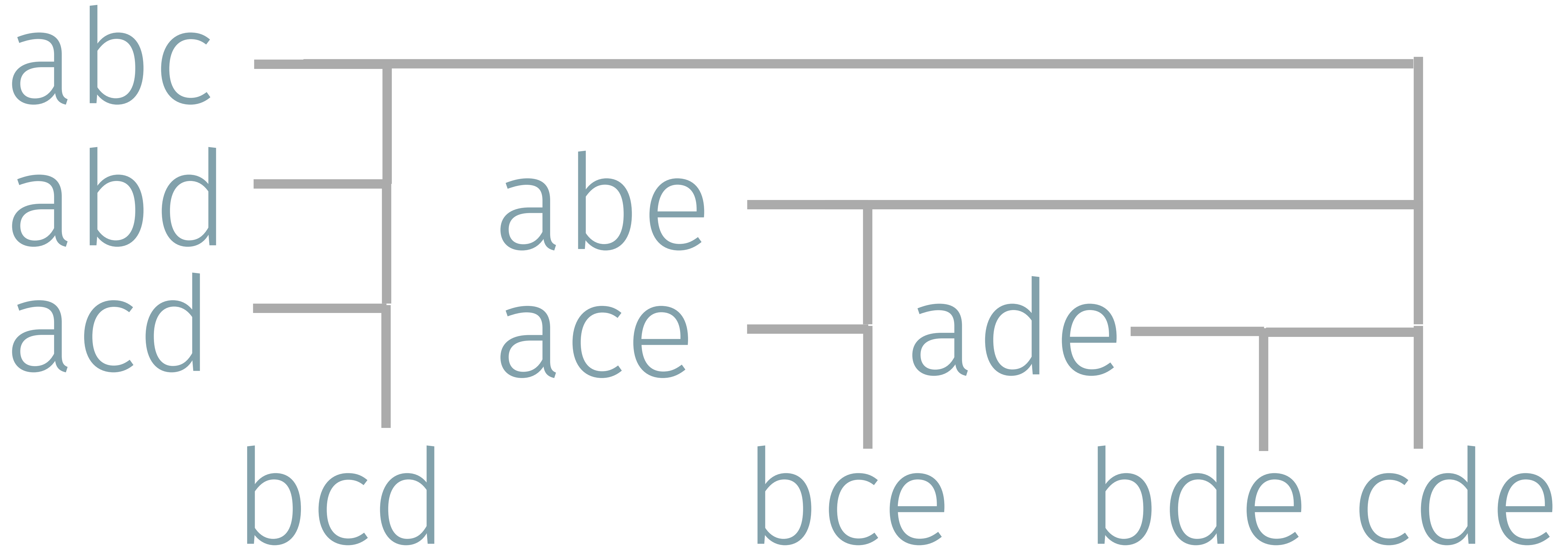
choose abcde 2 =



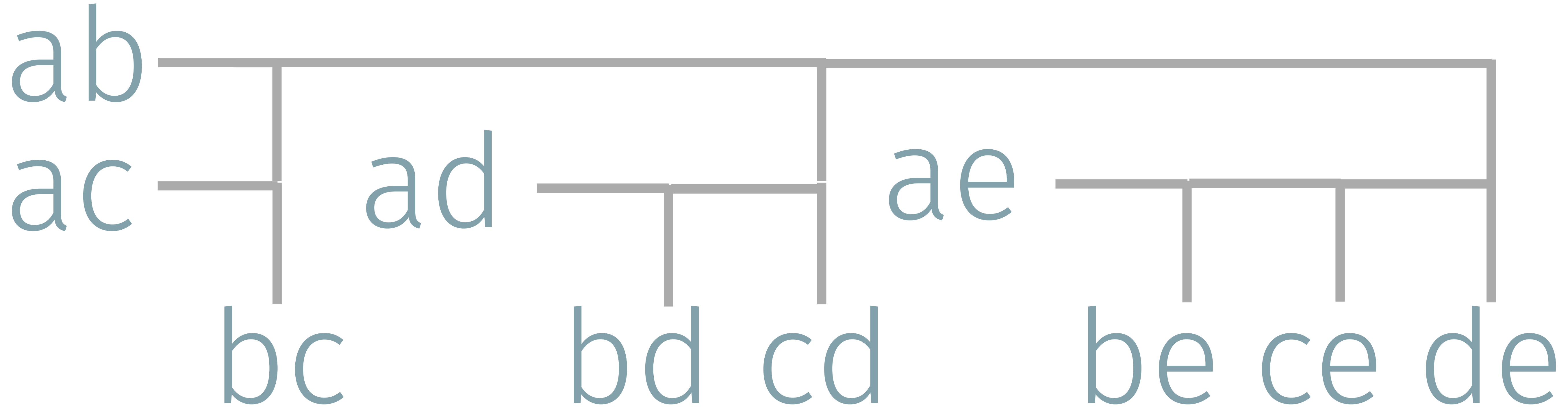
choose abcde 1 =

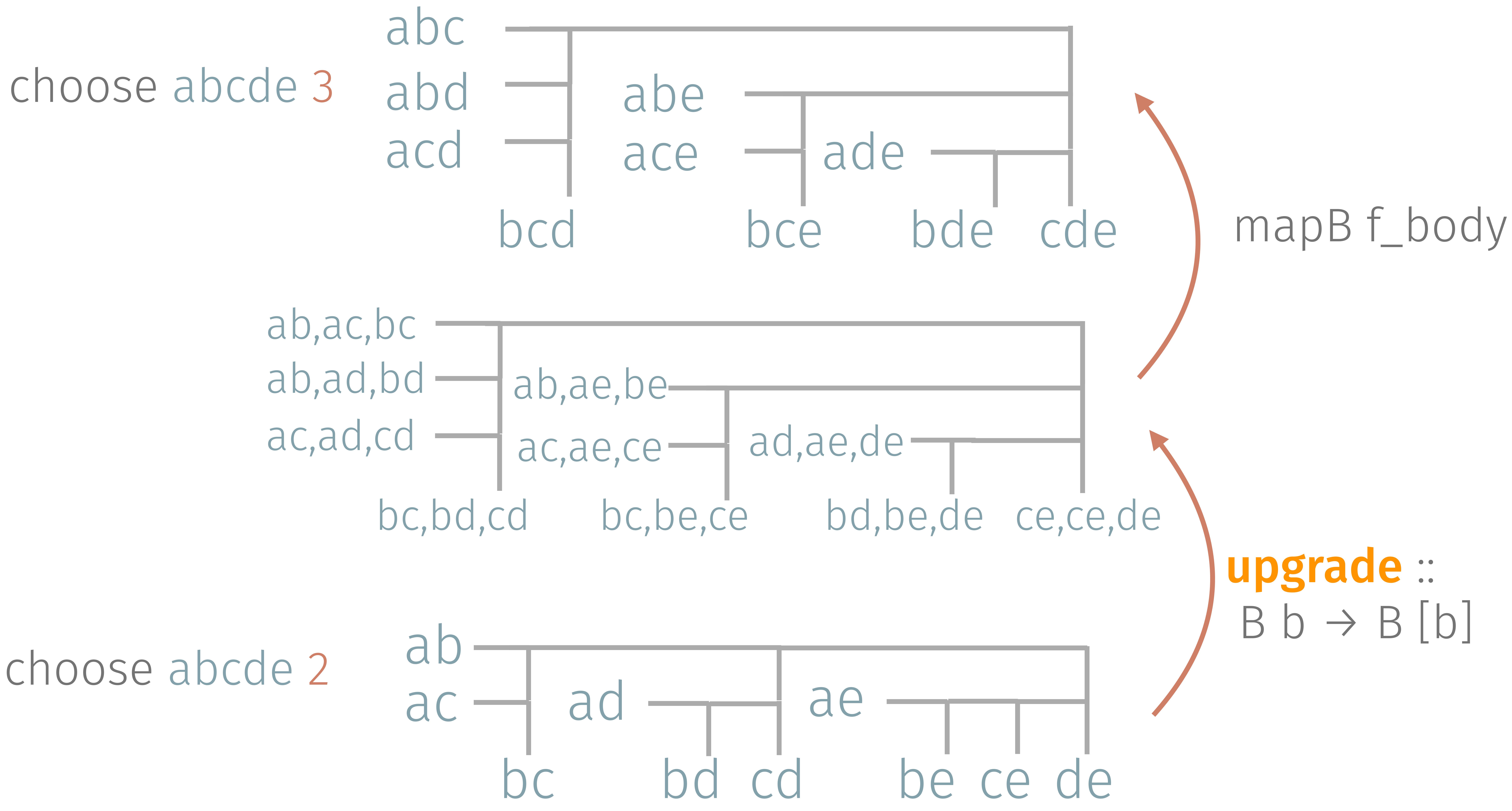


choose abcde 3 =



choose abcde 2 =




choose abcde 3
abc
abd
acd
abe
ace
ade
bcd
bce
bde
cde
mapB f_body
ab,ac,bc
ab,ad,bd
ac,ad,cd
ab,ae,be
ac,ae,ce
ad,ae,de
bc,bd,cd
bc,be,ce
bd,be,de
ce,ce,de
upgrade ::
B b → B [b]
choose abcde 2
ab
ac
ad
ae
bc
bd
cd
be
ce
de

Specification

$$\forall\ xs :: [a],\ k :: Nat\ .\ k > 0 \wedge \#xs \geq k{+}1\ .$$
$$upgrade\ (choose\ xs\ k) = mapB\ sub\ (choose\ xs\ (k{+}1))$$

Recall:

```
choose  :: [a] → Nat → B [a]
sub     :: [a] → [[a]]
upgrade :: B b → B [b]
```

Case $xs := xs +\!\!+ [z]$, $k := k+1$, $\# \, xs > k$

$$
\begin{aligned}
& \mathit{up}\ (\mathit{choose}\ (xs +\!\!+ [z])\ (k+1)) \\
=\ & \mathit{up}\ (N\ (\mathit{choose}\ xs\ (k+1)) \\
& \qquad\qquad (\mathit{mapB}\ (+\!\!+ [z])\ (\mathit{choose}\ xs\ k)) \\
=\ & ??\ \ \mathit{up}\ (\mathit{choose}\ xs\ (k+1))\ ?? \\
& ??\ \ \mathit{up}\ (\mathit{mapB}\ (+\!\!+ [z])\ (\mathit{choose}\ xs\ k))\ ?? \\
& \qquad\qquad \vdots \\
=\ & \mathit{mapB}\ \mathit{sub}\ (\mathit{choose}\ (xs +\!\!+ [z])\ (k+2))
\end{aligned}
$$

```
  up (N (choose xs (k+1))
            (mapB (⧺[z]) (choose xs k))

= ??  up (choose xs (k+1)) ??
  ??  up (mapB (⧺[z]) (choose xs k)) ??

                  ⋮

=  mapB sub (choose (xs⧺[z]) (k+2))
```

we may then pick

```
up (N t u) = ?? up t ??
             ?? up u ??
```

To Establish: up (choose xs k) = mapB sub (choose xs (k+1))

mapB sub (choose (xs++[z]) (k+2))

= { def. of choose }

mapB sub (N (choose xs (k+2))
(mapB (++[z]) (choose xs (k+1))))

Goal: ..??.. up (choose xs (k+1)) ..??..
..??.. up (mapB (++[z]) (choose xs k))

To Establish: up (choose xs k) = mapB sub (choose xs (k+1))

mapB sub (N (choose xs (k+2))
(mapB (++[z]) (choose xs (k+1))))

= { def. of mapB }

N (mapB sub (choose xs (k+2))
(mapB (sub · (++[z])) (choose xs (k+1))))

Goal: ..??.. up (choose xs (k+1)) ..??..
..??.. up (mapB (++[z]) (choose xs k))

To Establish: up (choose xs k) = mapB sub (choose xs (k+1))

$$
\begin{aligned}
& \mathrm{N}\ (\mathrm{mapB\ sub\ (choose\ xs\ (k{+}2))}) \\
& \quad (\mathrm{mapB\ (sub} \cdot (\mathbin{+\!\!+}[z]))\ (\mathrm{choose\ xs\ (k{+}1)))}) \\
= & \quad \{\ \text{induction}\ \} \\
& \mathrm{N}\ (\mathrm{up\ (choose\ xs\ (k{+}1))}) \\
& \quad (\mathrm{mapB\ (sub} \cdot (\mathbin{+\!\!+}[z]))\ (\mathrm{choose\ xs\ (k{+}1)))})
\end{aligned}
$$

Goal: ..??.. up (choose xs (k+1)) ..??..
..??.. up (mapB (++[z]) (choose xs k))

To Establish: up (choose xs k) = mapB sub (choose xs (k+1))

N (up (choose xs (k+1))
(mapB (sub · (++[z])) (choose xs (k+1))))

Lemma: sub (zs++[z]) = zs : map (++[z]) (sub zs)

E.g: sub abcdz = [abcd, abcz, abdz, acdz, bcdz]

Goal: ..??.. up (choose xs (k+1)) ..??..
..??.. up (mapB (++[z]) (choose xs k))

To Establish: up (choose xs k) = mapB sub (choose xs (k+1))

N (up (choose xs (k+1))
(mapB (sub · (++[z])) (choose xs (k+1))))

Lemma: sub (zs++[z]) = zs : map (++[z]) (sub zs)

Lemma: mapB (sub · (++[z])) t =
zipBW (:) t (mapB (map (++[z]) · sub) t)

Goal: ..??.. up (choose xs (k+1)) ..??..
..??.. up (mapB (++[z]) (choose xs k))

To Establish: up (choose xs k) = mapB sub (choose xs (k+1))

$$
\begin{aligned}
& \mathsf{N}\ (\mathsf{up}\ (\mathsf{choose}\ xs\ (k{+}1)) \\
& \quad (\mathsf{mapB}\ (\mathsf{sub} \cdot ({+\!\!+}[z]))\ (\mathsf{choose}\ xs\ (k{+}1)))) \\
= \quad & \{\ \text{lemma below}\ \} \\
& \mathsf{N}\ (\mathsf{up}\ (\mathsf{choose}\ xs\ (k{+}1)) \\
& \quad (\mathsf{zipBW}\ (:)\ (\mathsf{choose}\ xs\ (k{+}1)) \\
& \qquad (\mathsf{mapB}\ (\mathsf{map}\ ({+\!\!+}[z]) \cdot \mathsf{sub})\ (\mathsf{choose}\ xs\ (k{+}1))))
\end{aligned}
$$

Lemma:

$$
\begin{aligned}
& \mathsf{mapB}\ (\mathsf{sub} \cdot ({+\!\!+}[z]))\ t = \\
& \quad \mathsf{zipBW}\ (:)\ t\ (\mathsf{mapB}\ (\mathsf{map}\ ({+\!\!+}[z]) \cdot \mathsf{sub})\ t)
\end{aligned}
$$

To Establish: up (choose xs k) = mapB sub (choose xs (k+1))

```
    N (up (choose xs (k+1))
       (zipBW (:) (choose xs (k+1))
           (mapB (map (++[z]) · sub) (choose xs (k+1))))
=      {  induction  }
    N (up (choose xs (k+1))
       (zipBW (:) (choose xs (k+1))
           (mapB (map (++[z])) (up (choose xs k))))
```

Goal: ..??.. up (choose xs (k+1)) ..??..
..??.. up (mapB (++[z]) (choose xs k))

To Establish: up (choose xs k) = mapB sub (choose xs (k+1))

N (up (choose xs (k+1))
  (zipBW (:) (choose xs (k+1))
    (mapB (map (++[z])) (up (choose xs k))))

= { naturalty: up :: B b → B [b] }

N (up (choose xs (k+1))
  (zipBW (:) (choose xs (k+1))
    (up (mapB (++[z]) (choose xs k))))

Goal: ..??.. up (choose xs (k+1)) ..??..
  ..??.. up (mapB (++[z]) (choose xs k))

To Establish: up (choose xs k) = mapB sub (choose xs (k+1))

```
N (up (choose xs (k+1))
   (zipBW (:) (choose xs (k+1))
      (up (mapB (++[z]) (choose xs k))))
```

Goal: ..??.. up (choose xs (k+1)) ..??..
..??.. up (mapB (++[z]) (choose xs k))

Choose:

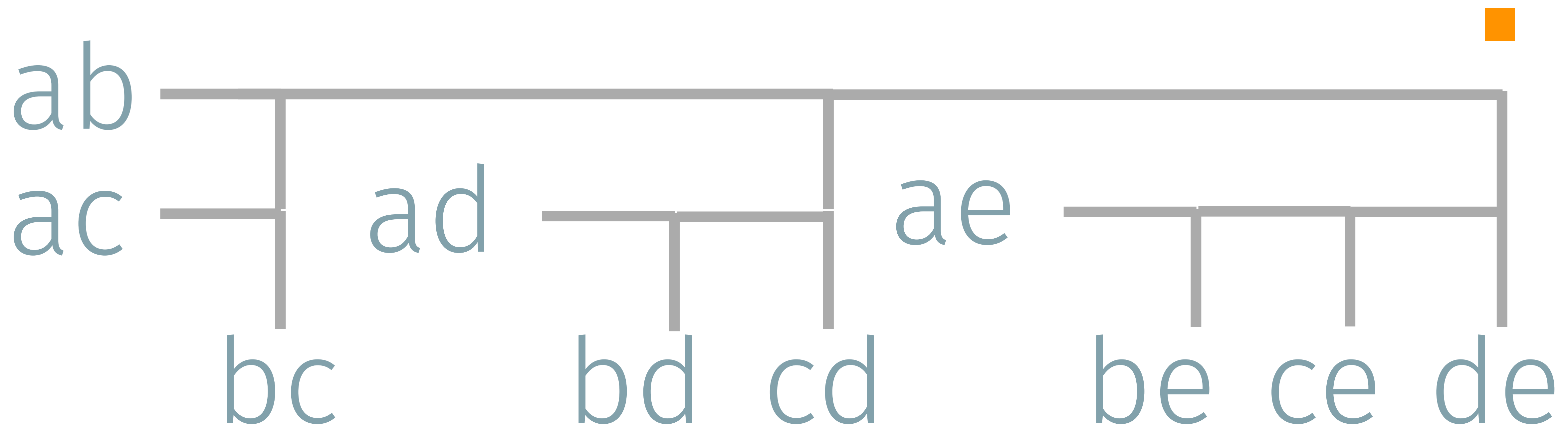
```
up (N t u) = N (up t)
               (zipBW (:) t (up u))
```

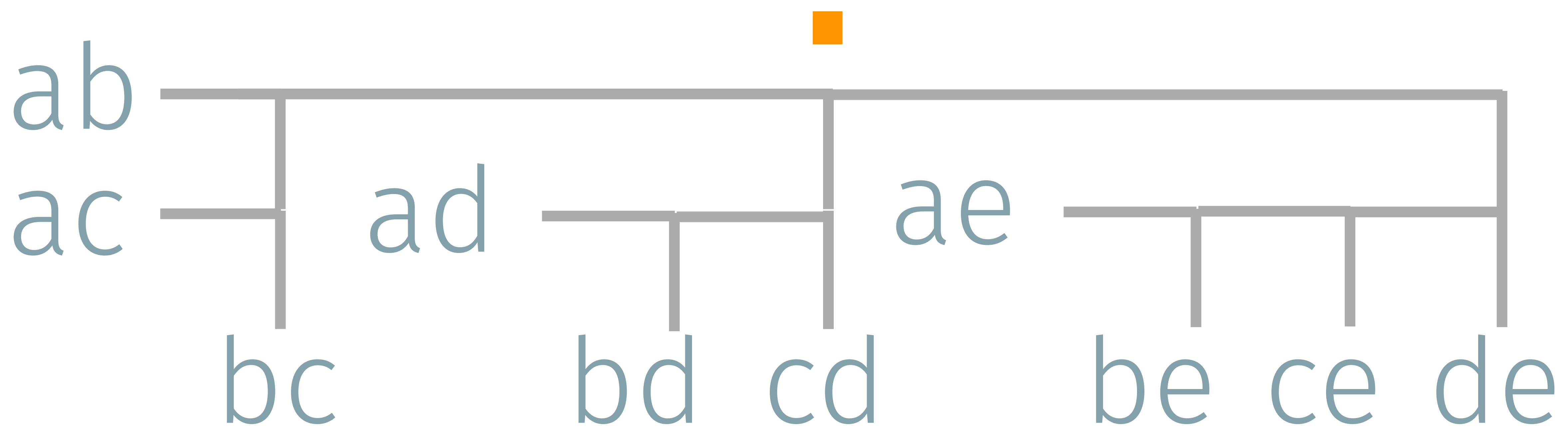
The Program

```
up :: B b → B [b]
up (N (T x) (T y)) = T [x, y]
up (N (T x) u    ) = T (x : unT (up u))
up (N t     (T y)) = N (up t) (map (++[y]) (up u))
up (N t      u   ) = N (up t) (zipBW (:) t (up u))
```

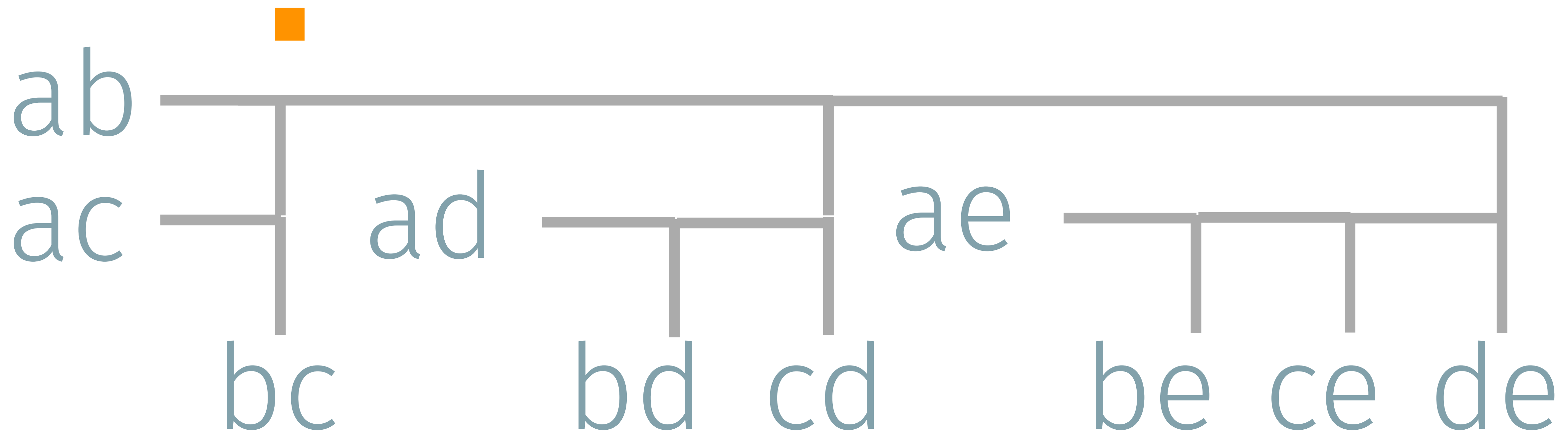
End

unless we have time for an example

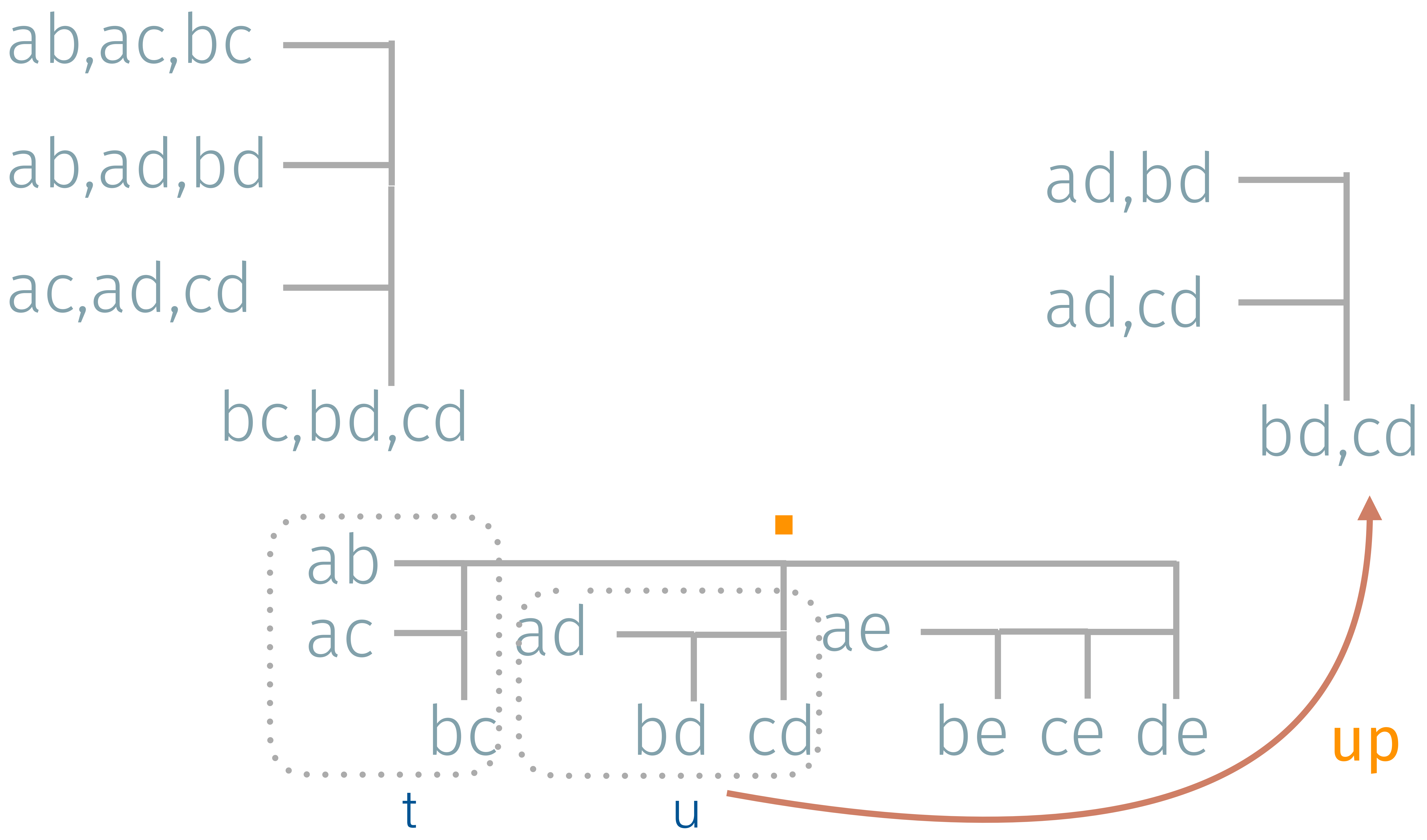

ab
ac
ad
ae
bc
bd
cd
be
ce
de


ab
ac
ad
ae
bc
bd
cd
be
ce
de

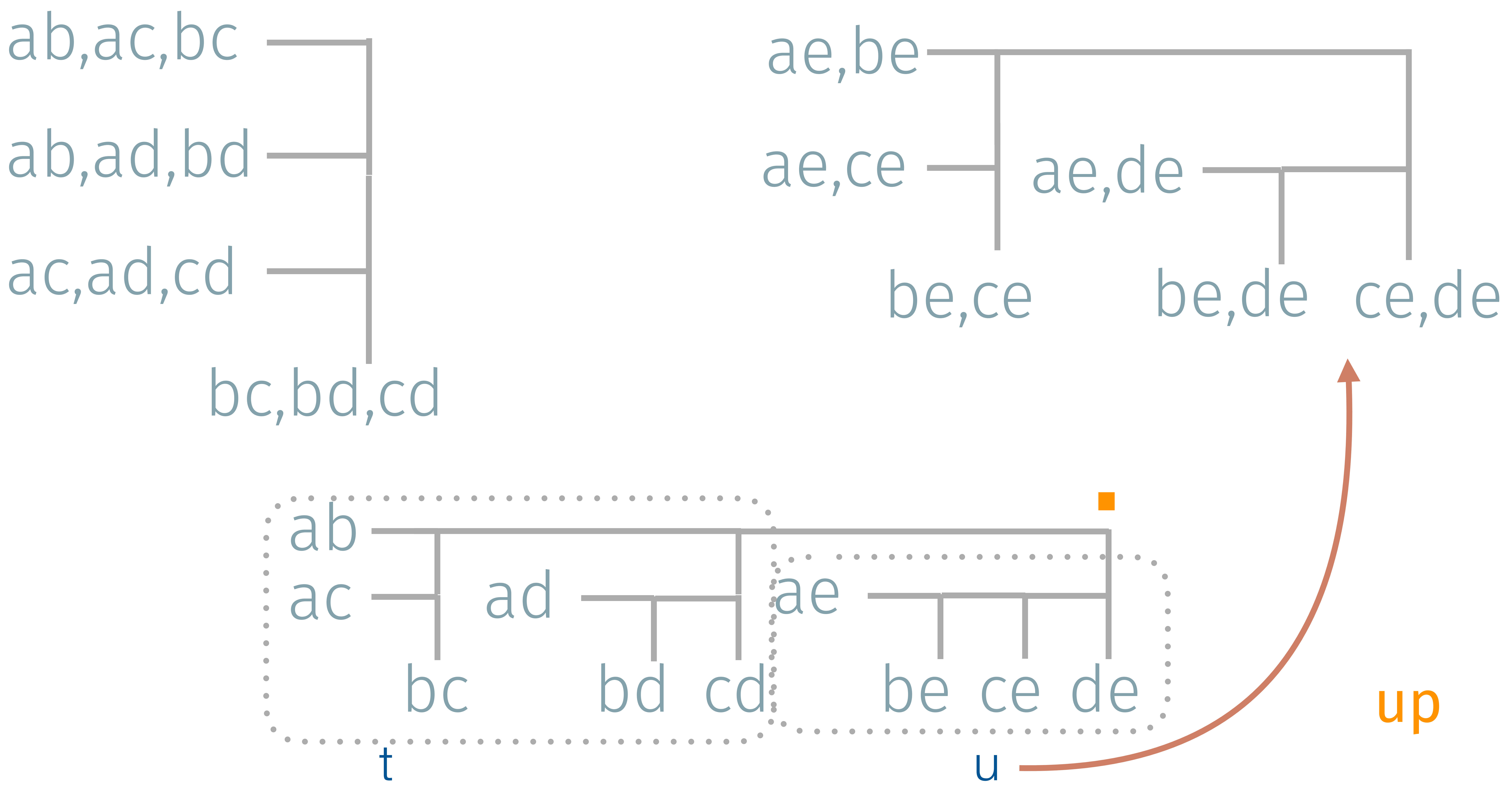
ab,ac,bc

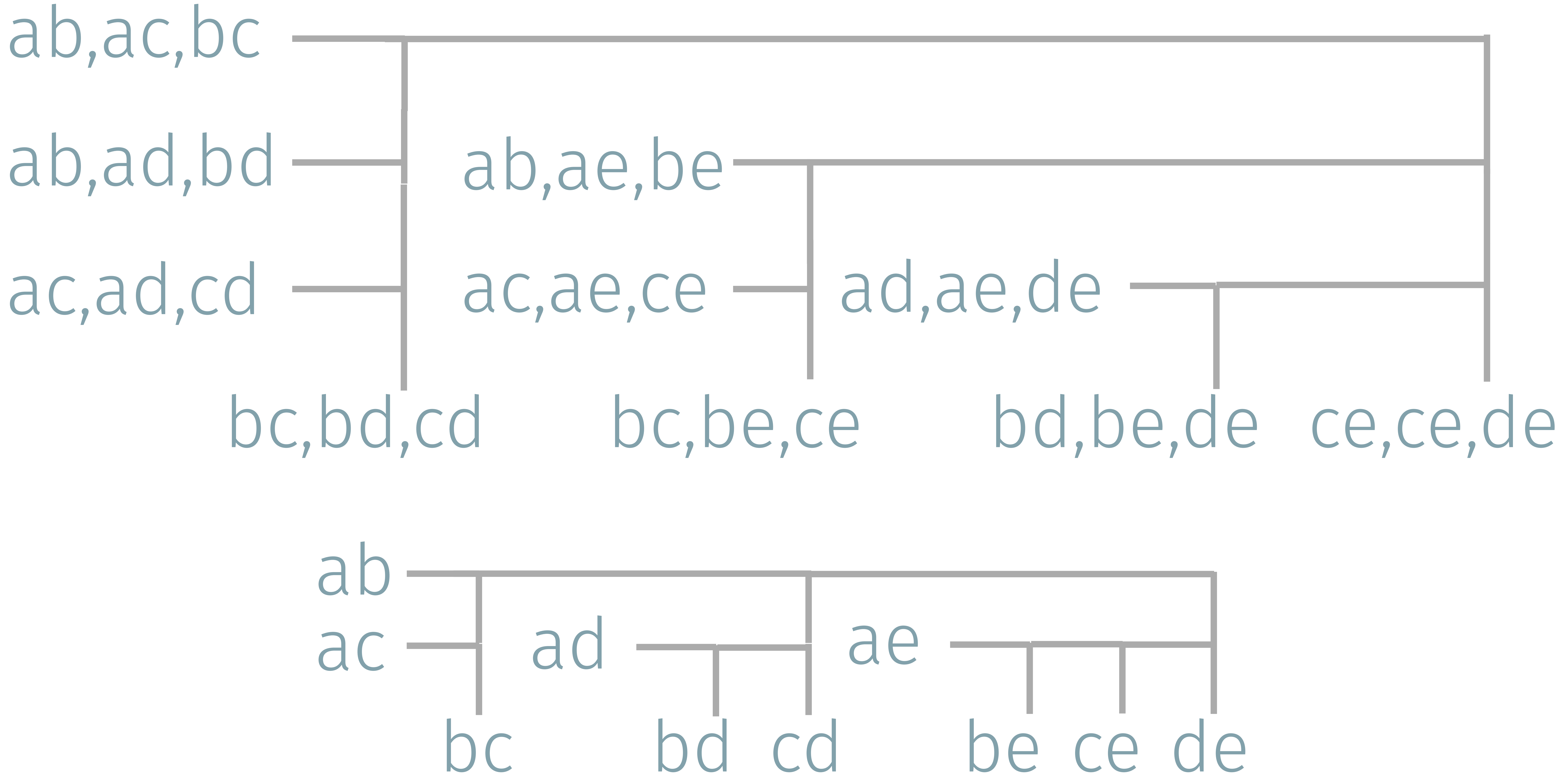


up (N t u) = N (up t) (zipBW (:) t (up u))



```
up (N t      u     ) = N (up t) (zipBW (:) t (up u))
```




ab,ac,bc
ab,ad,bd
ac,ad,cd
ab,ae,be
ac,ae,ce
ad,ae,de
bc,bd,cd
bc,be,ce
bd,be,de
ce,ce,de
ab
ac
ad
ae
bc
bd
cd
be
ce
de

End