```
cop (θ o′) (φ o′) = let ! ! ! ! tl , c , tr = cop θ φ in ! ! ! ! tl t-″  , c      , tr t-″
cop (θ o′) (φ os) = let ! ! ! ! tl , c , tr = cop θ φ in ! ! ! ! tl t′s′ , c c′s  , tr tsss
cop (θ os) (φ o′) = let ! ! ! ! tl , c , tr = cop θ φ in ! ! ! ! tl tsss , c cs′  , tr t′s′
cop (θ os) (φ os) = let ! ! ! ! tl , c , tr = cop θ φ in ! ! ! ! tl tsss , c css  , tr tsss
cop    oz    oz   =                                      ! ! ! !  tzzz ,   czz ,    tzzz
```

The copU proof goes by induction on the triangles which share ψ' and inversion of the coproduct.

A further useful property of coproduct diagrams is that we can selectively refine them by a thinning into the covered scope.

$$\begin{tikzcd} iz \arrow[r, dashed, "\psi_0"] \arrow[dr, dashed, "\theta"] & iz' \arrow[drr, "\theta'"] & & \\ & kz \arrow[rr, "\psi"] & & kz' \\ jz \arrow[r, dashed, "\psi_1"'] \arrow[ur, dashed, "\phi"'] & jz' \arrow[urr, "\phi'"'] & & \end{tikzcd}$$

```
subCop : (ψ : kz ⊑ kz′) → Cover ov θ′ φ′ →
         Σ _ λ iz → Σ _ λ jz → Σ (iz ⊑ kz) λ θ → Σ (jz ⊑ kz) λ φ →
         Σ (iz ⊑ iz′) λ ψ0 → Σ (jz ⊑ jz′) λ ψ1 → Cover ov θ φ
```

The implementation is a straightforward induction on the diagram.

The payoff from coproducts is the type of *relevant pairs* — the co-de-Bruijn touchstone:

```
record _×R_ (S T : K̄) (ijz : Bwd K) : Set where
  constructor pair
  field outl  : S ⇑ ijz;   outr : T ⇑ ijz
        cover : Cover tt (thinning outl) (thinning outr)
```

```
_,R_ : S ⇑ kz → T ⇑ kz → (S ×R T) ⇑ kz
(s ↑ θ) ,R (t ↑ φ) =
  let ! ψ , θ′ , φ′ , _ , c , _ = cop θ φ
  in  pair (s ↑ θ′) (t ↑ φ′) c ↑ ψ
```

The corresponding projections are readily definable.

```
outlR : (S ×R T) ⇑ kz → S ⇑ kz
outlR (pair s _ _ ↑ ψ) = thin⇑ ψ s
```

```
outrR : (S ×R T) ⇑ kz → T ⇑ kz
outrR (pair _ t _ ↑ ψ) = thin⇑ ψ t
```

7 Monoidal Structure of Order-Preserving Embeddings

Variable bindings extend scopes. The λ construct does just one 'snoc', but binding can be simultaneous, so the monoidal structure on Δ_+ induced by concatenation is what we need.

```
_++_ : Bwd K → Bwd K → Bwd K
kz ++ []       = kz
kz ++ (iz -, j) = (kz ++ iz) -, j
```

```
_++⊑_ : iz ⊑ jz → iz′ ⊑ jz′ → (iz ++ iz′) ⊑ (jz ++ jz′)
θ ++⊑    oz   = θ
θ ++⊑ (φ os)  = (θ ++⊑ φ) os
θ ++⊑ (φ o′)  = (θ ++⊑ φ) o′
```

Concatenation further extends to Coverings, allowing us to build them in chunks.

```
_++C_ : Cover ov θ φ → Cover ov θ′ φ′ → Cover ov (θ ++⊑ θ′) (φ ++⊑ φ′)
c ++C (d c′s)                = (c ++C d) c′s
c ++C (d cs′)                = (c ++C d) cs′
c ++C (_css {both = b} d)    = _css {both = b} (c ++C d)
c ++C czz                    = c
```

One way to build such a chunk is to observe that two scopes cover their concatenation.