```
1   conspd goal = residualize ∘ drive_disj ∘ normalize(goal, empty_substitution)
2
3   drive_disj :: [(Disjunction, Substitution)] → Process_Graph
4   drive_disj [(c₁, subst₁), ..., (cₙ, substₙ)] = ⋁ᵢ₌₁ⁿ tᵢ ← drive_conj(cᵢ, substᵢ)
5
6   drive_conj :: (Conjunction, Substitution) → Process_Graph
7   drive_conj ([], subst) = create_success_node (subst)
8   drive_conj ([r₁, ..., rₙ], subst) =
9     C@(r₁, ..., rₙ) ← propagate_substitution subst onto r₁, ..., rₙ
10    case whistle(C) of
11    | instance(C', subst')        ⇒ create_fold_node(C', subst')
12    | embedded_but_not_instance ⇒ create_stop_node(C, subst)
13    | otherwise ⇒
14    | | case heuristically_select_a_call(r₁, ..., rₙ) of
15    | | | Just r ⇒
16    | | | | t ← unfold_in_isolation(r, subst)
17    | | | | ls ← leaves(t)
18    | | | | if trivial(ls)
19    | | | | then
20    | | | | | ⋁ᵢ₌₁ᵏ tᵢ ← drive_conj(C[r ↦ get_call(i, ls)], get_subst(i, ls))
21    | | | | else
22    | | | | | t ⋀ drive_conj(C \ r, subst)
23    | | | Nothing ⇒ ⋀ᵢ₌₁ⁿ tᵢ ← drive_disj ∘ normalize ∘ unfold(rᵢ, subst)
24
25  heuristically_select_a_call :: Conjunction → Maybe Call
26  heuristically_select_a_call C =
27    find isStatic C <|> find isDeterministic C <|> find isLessBranching C
```