```
1  | conspd goal = residualize ∘ drive_disj ∘ normalize(goal, empty_substitution)
2  |
3  | drive_disj :: [(Disjunction, Substitution)] → Process_Graph
4  | drive_disj [(c_1, subst_1), ..., (c_n, subst_n)] = ⋁_{i=1}^{n} t_i ← drive_conj(c_i, subst_i)
5  |
6  | drive_conj :: (Conjunction, Substitution) → Process_Graph
7  | drive_conj ((r_1, ..., r_n), subst) =
8  |   C@(r_1, ..., r_n) ← propagate_substitution subst onto r_1, ..., r_n
9  |   case whistle(C) of
10 |   | instance(C', subst')        ⇒ create_fold_node(C', subst')
11 |   | embedded_but_not_instance ⇒ create_stop_node(C , subst )
12 |   | otherwise ⇒
13 |   | | case heuristically_select_a_call(r_1, ..., r_n) of
14 |   | | | Just r ⇒
15 |   | | | | t ← one_step_unfold(r, subst)
16 |   | | | | ls  ← leaves(t)
17 |   | | | | if trivial(ls)
18 |   | | | | then
19 |   | | | | | ⋁_{i=1}^{k} t_i ← drive_conj(C[r ↦ get_call(i, ls)], get_subst(i, ls))
20 |   | | | | else
21 |   | | | | | r ⋀ drive_conj(C \ r, subst)
22 |   | | | Nothing ⇒ ⋀_{i=1}^{n} t_i ← drive_disj ∘ normalize ∘ unfold(r_i, subst)
```