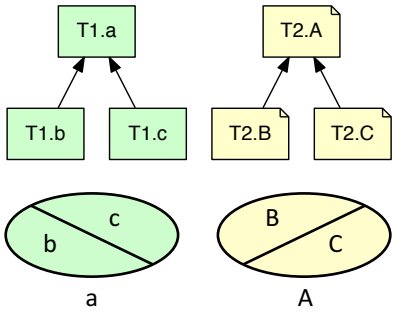
Underspecified TAP? Explore many *possible worlds*!

Want to resolve overlaps? Use *combined concepts* in "*zoom-in view*"

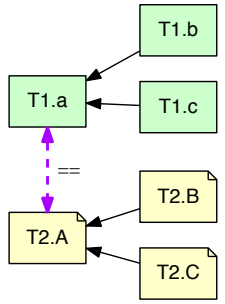
Taxonomy Alignment Problems (TAPs) can have many solutions, i.e., many *possible worlds* {W_1, … W_n}. To resolve *overlaps* in these worlds, employ a "zoom-in view" (i.e., the *combined concept* option "`-e mncb`").

- **Congruent solutions** ("perfect matches", **0 new names**, no overlaps)
- **"In-between" solutions** (common in practice: not every region/possible new names actually exists; here: **1 new name**)
- **"Finest resolution"** (but often least desirable and indication of bugs in the TAP: many overlaps & new names; here: **4 new names**)



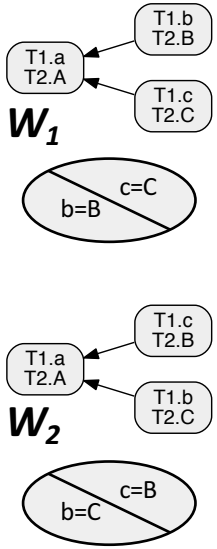
Input Taxonomies

… often are **partitions**: i.e., siblings are **disjoint** and parents **covered** by their children



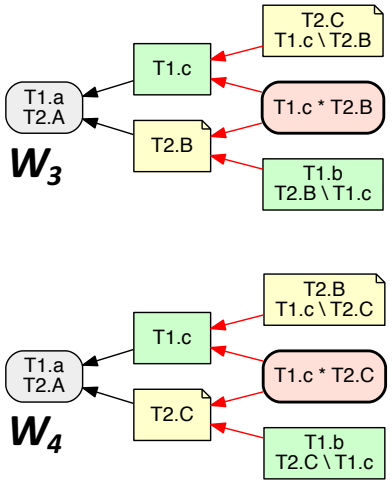
Underspecified TAP

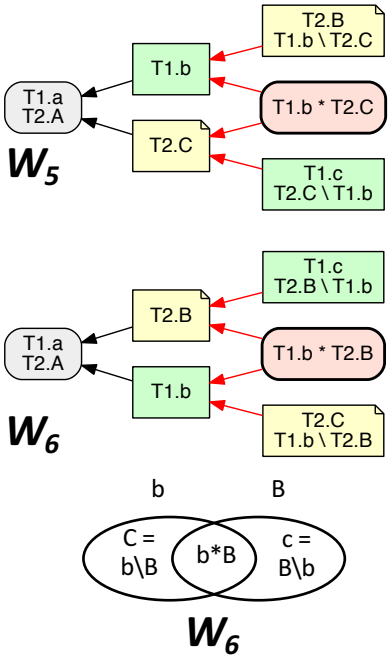
(Taxonomy Alignment Problem) includes **articulations** between concepts



Congruent solutions (2)

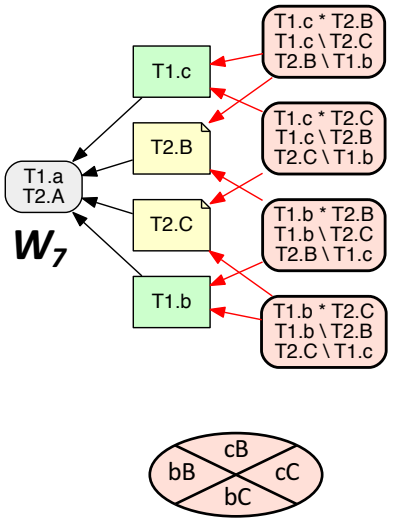
input taxonomies are isomorphic, i.e., **permutations** of each other





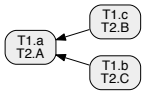
"Intermediate" solutions (4)

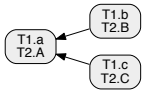
sometimes (like here) **solutions** are isomorphic, i.e., permutations of each other; fewer new names



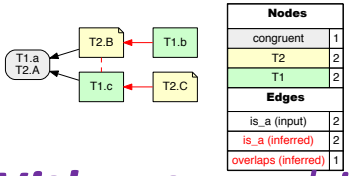
"Finest" solution (1)

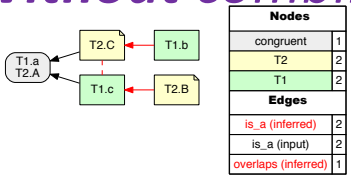
most complex PWs, with many new names; rarely the "real" solution(s)

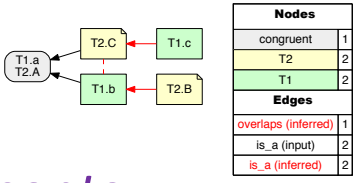


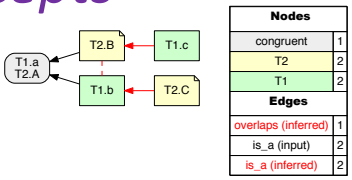


0 overlaps

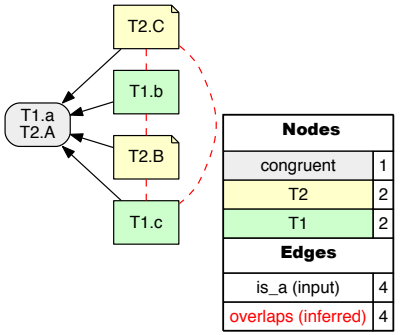






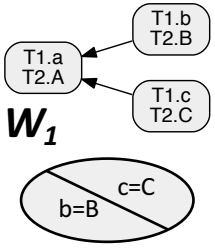


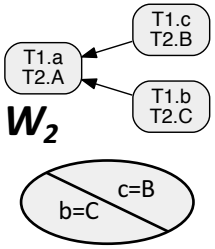
1 overlap



4 overlaps

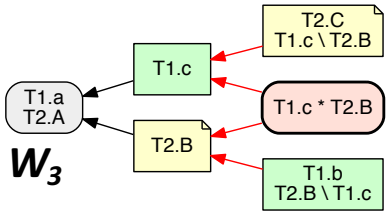
Without *combined concepts*

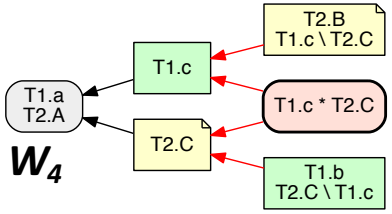




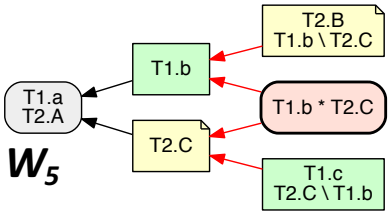
Congruent solutions (2)

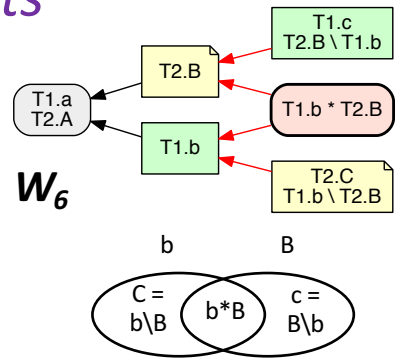
input taxonomies are isomorphic, i.e., **permutations** of each other





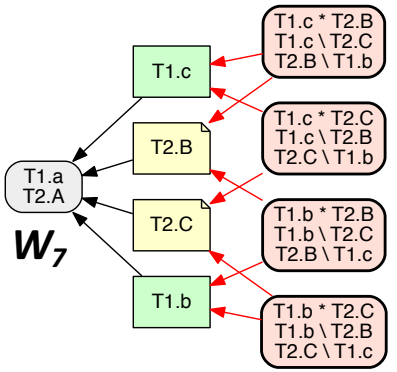
With *combined concepts*

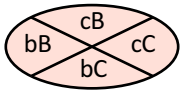




"Intermediate" solutions (4)

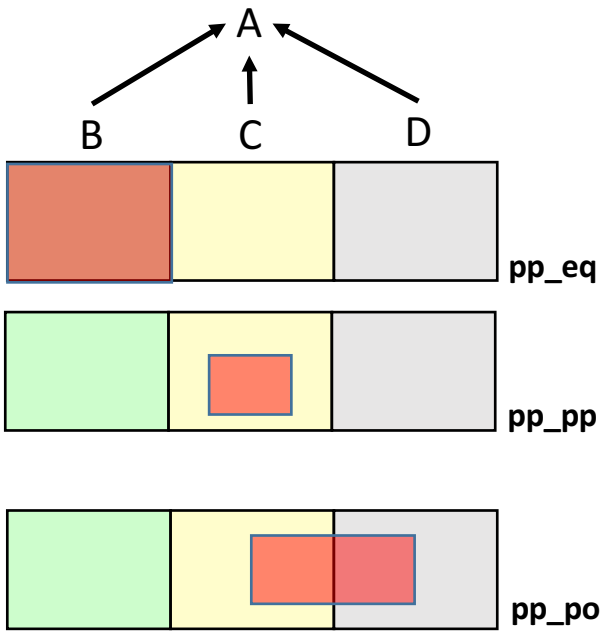
sometimes (like here) **solutions** are isomorphic, i.e., permutations of each other; fewer new names





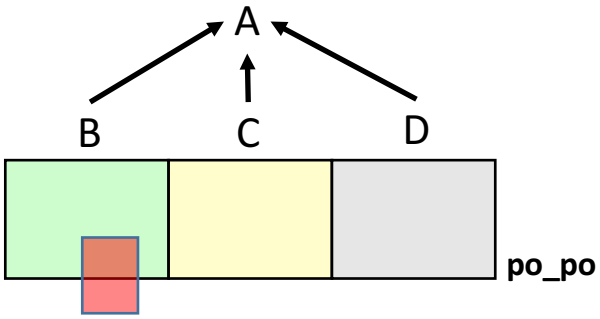
"Finest" solution (1)

most complex PWs, with many new names; rarely the "real" solution(s)




```
5   % (1) Let X be a proper-part of A, or pp(X,A) for short.
6
7   % If there's a child (block) B of A ..
8   % ..  and eq(X,B) => OK_ppeq
9   ppeq(A,X) :- pp(X,A), bl(A,B), eq(X,B).
10
11  % .. and pp(X,A) => OK_pppp
12  pppp(A,X) :- pp(X,A), bl(A,B), pp(X,B).
13
14  % .. and a second block C, and X overlaps both B and C => OK_pppo
15  pppo(A,X) :- pp(X,A), bl(A,B), bl(A,C), B != C, po(X,B), po(X,C).
16
17  % If X pp A, then you can't have none of the three cases above!
18  x1(X,A) :- pp(X,A), not ppeq(A,X), not pppp(A,X), not pppo(A,X).
19
20  :- x1(_,_), x1on. % use x1on to switch x1 ON/OFF
```

- Consider a parent A partitioned into children **bl**ocks B, C, D, …
- If a region **X is proper-part of A,** then exactly one of three cases must hold, right?
  - pp_eq or pp_pp or pp_po
- **x1(X,A)** signals an IC **violation**: although pp(X,A) holds, none of ppeq, pppp, or pppo holds!



```
23  % (2) Now let X (partially) overlap A, or po(X,A) for short.
24
25  % If there's a child (block) B of A which overlaps X => OK_popo
26  popo(A,X) :- po(X,A), bl(A,B), po(X,B).
27
28  % If X po A, then you can't not have an overlapping block B with X
29  x2(X,A) :- po(X,A), not popo(A,X).
30
31  :- x2(_,_), x2on. % use x2on to switch x2 ON/OFF
```

- Consider a parent A partitioned into children blocks B, C, D, …
- If a region **X partially-overlaps A**, then there must be *at least one* block which X overlaps, right?
  - Hence po_po must hold for some block B
- **x2(X,A)** signals an IC **violation**: po(X,A) holds, but no block B of A partially overlaps A!