[Add Title]

[Add Author name]

June 2021

1 Algorithm

1.1 Implementation for each Atom

The following list mentions how each of the atoms are dealt with to create the Feature tree.

1. $x[f]y$: When ever it is encountered a mapping is added to node of x from f to y.

2. $x[f] \uparrow$: Use the special variable 0 (which represents absent) and execute atom x[f]var0

3. $x =_* y$: union the node of x and that of y and map both the variables to this node. In the node keep record of all the variables it is mapped from

4. $x =_F y$: During the initial loop of the clause keep a record of the set F and the other variable in node of x and that of y. And use this information in the dissolve phase.

5. $x[F]$: Check all features that have a mapping in x. And if any such feature f $\notin$ F then the input clause is not valid (there is a clash). This check needs to be done after equality dissolve

6. $x \neq_F y$: Go through each feature in set F, if any of them give a different mapping in node of x and node of y. Then the atom is already satisfied. One of the node is having a mapping for a feature and other does not. In that case for the other node add a absent mapping for the feature.

If both cases don't satisfy find an element f ∈ F which has no mapping in node of x or that of y. And create a mapping with a new feature and variable to x and an absent mapping to y. If not such f exists then it is a clash(invalid input clause) [Note to self: Think of cases where this solution wouldn't work]

7. $\neg x[F]$: For all direct and indirect mapping of feature in node of x, check if for any feature f $\notin$ F. If so, atom is already satisfied. If not, create a new feature and variable and add a map to node of x for them.

8. $\phi[x \to y]$: [Note to self: I think its just for representation purpose, should be same as $x =_* y$, check rules]

9. $x \sim_F y$: This atom means nodes of x and y have same mapping everywhere except maybe in some elements(features) of set F. For this atom we do something similar to that for $x =_F y$. While dissolving if $f \notin F$ we go search for it in y.

10. $x \not\sim_F y$: This atom means x and y differ in s feature f $\notin$ F. Check for a feature $f \notin F$ which has a mapping in node of x what differs in node of y [if feature has no mapping in node of y we can add an absent]. If no such feature is found do the same with roles of x,y interchanged. If either is satisfied then the atom is satisfied by our current tree. If not create a new feature and a new variable and map them in node of x and an absent map in y for the feature. If there exists a x[E] in the clause, the step mentioned above cant be performed , so we replace the roles of x and y in previous step. If y also has a y[G]. We will have to find a feature $f \in [E - F]$ which has no specific mapping in x and add a different mapping than in y(add absent in node of y for f, if no mapping present).If no such f exists try with y. If not luck even then, its a crash(invalid input clause)

1.2 Order of handling atoms

Results of few of the atoms are interdependent so are required to look at the problem in a certain order.The following is an order in which the problem can be handled.

- On the first loop over the clause collect information from $x[f]y$, $x[f] \uparrow$, $x =_F y$, $x[F]$ and $x \sim_F y$ atoms and store them in the required node

- During the second iteration, consider all $x =_* y$ atoms. In this case both x and y can share a node which is obtained by:

  1. union of the feature mappings(obtained from $x[f]y$, $x[f] \uparrow$), equality mappings (obtained from $x =_F y$) and sim mapping (obtained from $x \sim_F y$) [Note to self: Consider for $x =_F y$, $z =_F y$,$x =_* z$]
  2. intersection of Fen list (obtained from $x[F]$)

- Dissolve phase Go through each node and and for all features it has an equality find its mapping. This mapping can be present in the current node itself, in the node of the variable the equality is with, or an equality this node on current feature and so on. If no such mapping is present we go on to the next equality on the feature. If till the end we get no mapping we skip the current feature and go for the next. [Note to self: add sim in dissolve phase]

- ...

1.3 Problem cases to be handled

1. In the presence of $x[f]y$ and $x[f]z$. Due to transformations during satisfiability test a $y =_* z$ will already be present so for feature f in node of x we can map to either y or z. But while adding features we need to make sure there isn't already a mapping with var0 (meaning absent)

2. Now if we consider $x \sim_F y$ for dissolving we cant just consider only features having a direct mapping in node of x. We can run dissolve for each node while considering the whole list of features in the problem

3. atoms like $\neg x[F]$ and $x \not\sim_F y$ can introduce new features. So if we were to implement them after dissolve. Dissolve for sim mapping would be incomplete. and thus have to be performed again. and this might change the result of the atoms mentioned before. [**Possible solution:** Possible solution: Let the dissolve not be a hard one. But when every checking for a feature mapping in a node dissolve for that feature if found indirectly add it to direct feature mapping, and do a final dissolve at the very end before displaying results.]

4. for $x[F]$ and $\neg x[F]$ we need to check if any feature with direct or indirect mapping in node of x belongs F or not. With Sim it might mean a lot of computation.

1.4 TODO: code

1. As for now while dissolving it returns 0 if now mapping found at all. change it as 0 is for absent

2. add space to store sim and fen in a node

3. implement abs with var0

4. edit the dissolve function to get a function that checks if a feature has a direct or indirect mapping in a node. for $x =_F y$ got to y if $f \in F$ and for $x \sim_F y$ got to y if $f \notin F$. Use List.mem for the checks

5. Make a function that given a node dissolves all its direct and indirect mapping.[add sim to the existing version]