Here is the GUIDSL specification of the Graph Product Line.

/// grammar

GPL : Alg+ [Src] Wgt Gtp;
Gtp : Directed | Undirected ;
Wgt : Weighted | Unweighted ;
Src : BFS | DFS ;
Alg : Number | Connected | SC | Cycle | MSTPrim | MSTKruskal |
    | ShortestPath ;
SC : Transpose StronglyConnected;

%% // constraints

Number implies Gtp and Src;   // means a Gtp and Src must be selected
Connected implies Undirected and Src;  // means a Src must be selected
StrongConnect implies Directed and DFS;
Cycle implies DFS;
MSTKruskal or MSTPrim implies Undirected and Weighted;
MSTKruskal or MSTPrim implies not (MSTKruskal and MSTPrim);
ShortestPath implies Directed and Weighted;

a) (MSTKruskal, MSTPrim)  legal  illegal
   One excludes the other

b) (Directed, Cycle) legal illegal
   they are compatible

c) (ShortestPath, StronglyConnect) legal illegal
   they are compatible

Each of the questions below I start with no selected features. Then I select:

d) (Undirected) what other assignments are selected or deselected for me?
directed, shortest path, stronglyConnect are deselected

e) (BFS)  
DFS, Cycle, StronglyConnect are deselected
2. Recall this slide from class lectures. I gave to one person the triangle and to another the rectangle and a pair of scissors to “modularize” their shapes. And then I tried to use these existing modules to produce the rhomboid below and couldn’t do it without serious hacking. I then “remodularized” the triangle and rectangle along the dashed lines below, showing that if I had chosen these modularities, I could build the rhomboid instantly at virtually no cost.

I have labeled my modules above (LT for left triangle, LB for left box, etc.).

Question: what is the feature model of this product line? You will:

(a) Provide a Feature diagram tree OR a GUIDSL context free grammar. You can not use A, B, C as labels or features. Only LT, RT, LB, RB as primitive tokens.
(b) Feature constraints (if none, say “none”).

This was much, much harder than I thought. Even I got it wrong (when designing the test). Anyone who proposed what I proposed got an A- (for this problem’s grade). Those who got it correct although I directed them away from the simplest answer got full credit 25pts (see below. For answers I couldn’t tell, I entered them into guidsl to see if they were correct. If so, they got 30pts (extra credit). I will announce the 2nd winner of the “TinkerToys award” for most elegant answer soon, with award. There are many solutions.

<table>
<thead>
<tr>
<th>The simplest that I tried to direct you away from</th>
<th>Sol 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPL : Aprod</td>
<td>Bprod</td>
</tr>
<tr>
<td>Aprod : LT RT;</td>
<td>Bprod : LB RB;</td>
</tr>
<tr>
<td>Cprod : LT RB;</td>
<td>RT implies LT;</td>
</tr>
<tr>
<td></td>
<td>BR implies LB;</td>
</tr>
<tr>
<td></td>
<td>LT and LB implies not RT and not RB;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sol2</th>
</tr>
</thead>
<tbody>
<tr>
<td>spl : [LT] [RT] [LB] [RB];</td>
</tr>
<tr>
<td>%</td>
</tr>
<tr>
<td>choose1(LT,RB);</td>
</tr>
<tr>
<td>LT implies (choose1(RT,LB));</td>
</tr>
<tr>
<td>RB implies LB and not RT;</td>
</tr>
<tr>
<td>LT and LB implies not RT and not RB;</td>
</tr>
</tbody>
</table>
Recall from lectures the metamodel of all class diagrams with associations and no inheritance. I do not list the constraints, but assume they are present.

### Question

a) What is the minimal addition to this diagram and constraints that permits inheritance relationships among classes? Draw the revised diagram.

b) What additional constraints, in English, are needed to retain the sanity of such diagrams?

c) Does the original diagram (above) conform to your revised metamodel of b)? Yes or No + briefly explain why.

**Answer: (3a)**

![Diagram](image)

**Additional Constraints**

- No inheritance cycles is basically what I was looking for.

**Yes it conforms.** The original diagram just doesn’t have inheritance relationships. The tabular representation of the modified diagram adds a field to the class table. This column would contain nulls for the original diagram.

**Note:** there were other answers – like augmenting associations to indicate that they can now represent inheritance relationships. This answer, while “correct”, is very invasive – lots of changes are needed to qualify association constraints from those that are inheritance constraints.
(3) **15 minutes max.** A common refactoring pushes an association “through” an abstract class to its subclasses:

By making class A associations reference abstract class B’s subclasses, a constraint must be added: each A instance is bound to a B1 or B2 instance, but never both.

Using the above refactoring and **any that we have discussed in class along with their names** – show that the left model can (or cannot) be mapped to the right model. **STATE ANY CONSTRAINTS THAT MUST BE APPLIED AS YOU PROCEED.**
constraint: oval is connected to Box or Diamond, but not both