Course on Model-Driven Engineering (EOM / MDE)

2022/2023
LAB 4 - Exercises - Domain Modelling

Vasco Amaral
October 2022

1. Feature Model Language

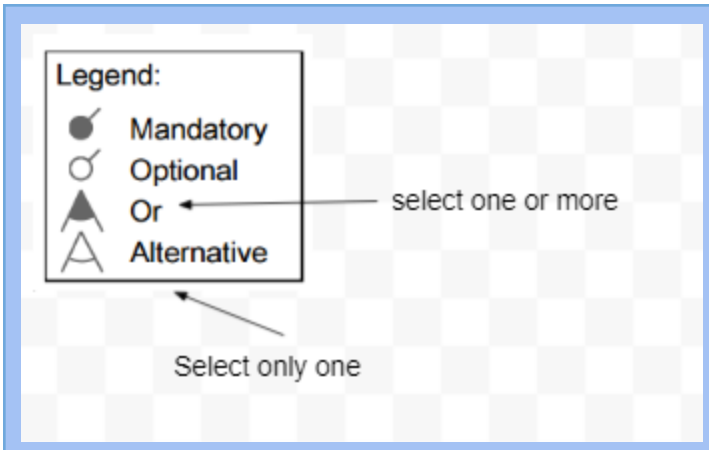
Knowing the concepts introduced in the theoretical classes regarding the abstract syntax of the Feature Models, define the metamodel and implement it in an ecore file.

Answer:

Remember from the lectures the notation and the well-formedness rules:

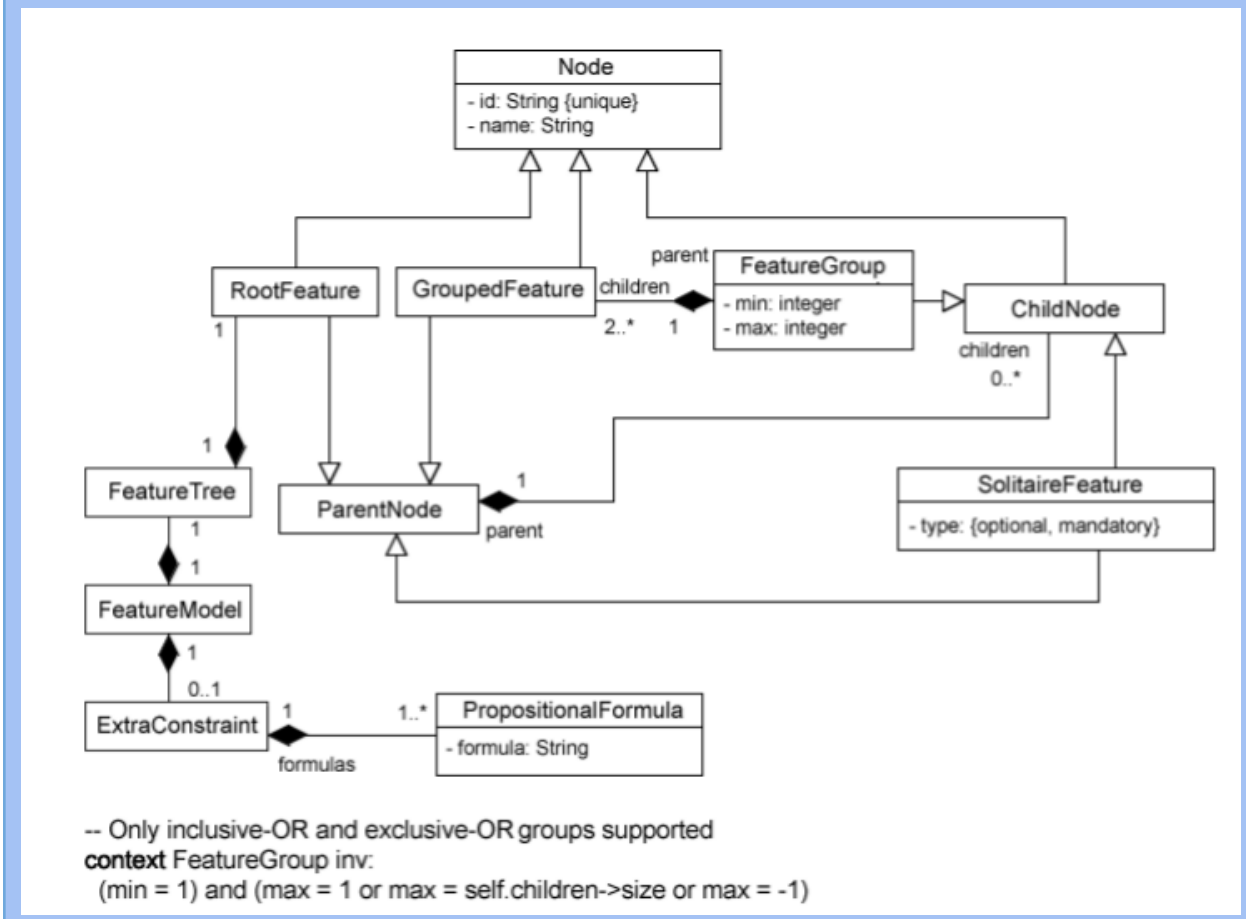
W1- It has only one root feature
W2- No feature becomes one of its own super-features
W3- No feature is an Island

Also:



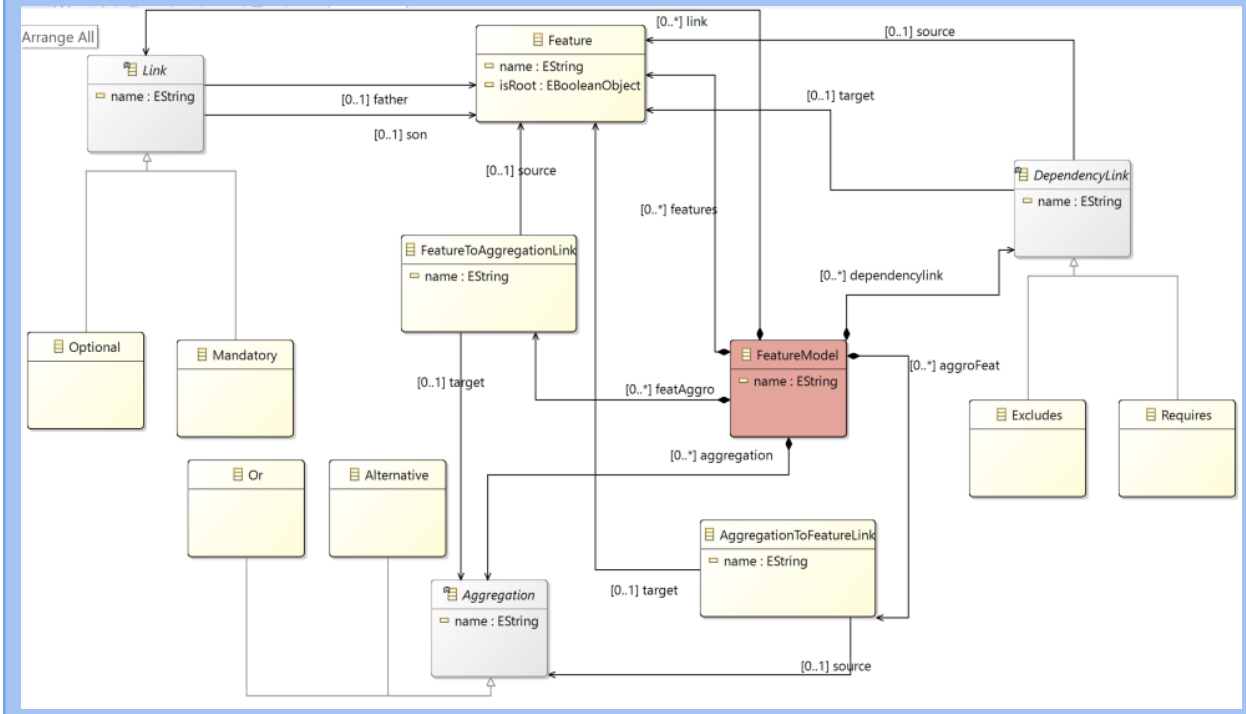
Consider that the excludes and includes relations can be directly represented as constraints using a propositional formula.

A possible model to answer this question could be (implement it in ecore):



Many other alternative metamodels can be designed. For instance, how would you adapt this model to hardcode implicitly in the metamodel the “excludes” and “includes” relations to be different from propositional formulas? Which EVL rules would you add to this metamodel?

Another example of the implementation in Eclipse would be




```
context FeatureModel{
	constraint mustHaveName {
		check: self.name <> ''

		message: 'A Feature Diagram must have a name'

		fix {
			title: 'Insert ' + self.eClass().name + ' Name'

			do {
				var name := UserInput.prompt('Insert ' + self.eClass().name + ' Name',
'');

				self.name = name;
			}
		}
	}
}

context Feature {
	constraint noFatherOfSelf{
		check: Link.allInstances.select(l | l.father.name = l.son.name and self.name =
l.son.name).size() = 0

		message: 'Features cant be father of themselves'
```

```
	}

	constraint noNameRepetions {
		check: Feature.allInstances.excluding(self)->forAll(i | i.name <> self.name)

		message: 'Two Features cant have the same name'
	}

	constraint noRootFeatures {
		guard: Feature.allInstances.select(f | f.isRoot = true).size() = 0
		check {
			return Feature.allInstances.select(f | f.isRoot = true).size() <> 0;
			}

		message: 'There are no Root Features. This can be fixed by Quick Fix'

		fix {
			title: 'Root ' + self.eClass().name + ' is needed'

			do {
				var rootFeatureSequence = Feature.allInstances.select(f |
Link.allInstances.forAll(l | l.son.name <> f.name)
					and AggregationToFeatureLink.allInstances.forAll(l |
l.target.name <> f.name));

				if(rootFeatureSequence.size() > 1) {
					rootFeatureSequence.size().println('Size do If: ');
					var sonFeature := UserInput.choose('Select feature',
_Model.getAllOfType(self.eClass().name));

				sonFeature.isRoot = true;
					} else {
					rootFeatureSequence[0].isRoot = true;
				}
			}
		}
	}

	constraint onlyOneRootFeature {
		guard: Feature.allInstances.select(f | f.isRoot = true).size() > 0
		check: Feature.allInstances.select(f | f.isRoot = true).size() = 1 or not (self.isRoot
= true)

		message: 'There can only be one Root Feature'
	}

	constraint nonRootFeatureMustHaveAFather {
		check: Link.allInstances.select(l | l.son.name = self.name).size() +
AggregationToFeatureLink.allInstances.select(l | l.target.name = self.name).size() = 1 or self.isRoot
= true

		message: 'Non-Root Features can only have one father feature'
```

```
	}
}

context Aggregation {
	constraint atLeastTwoFeatures{
		check: AggregationToFeatureLink.allInstances.select(l | l.source = self).size() > 1

		message: 'Aggregation Nodes must have at least two child Features'
	}
}

context Link {
	constraint notFatherOfRoot {
		check: self.son.isRoot <> true

		message: 'The Root Feature cant have fathers'
	}

}

//Excludes
operation DependencyLink noExcludesMandatories() : Boolean {
	var excludesSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
	var excludesTargetLinks = Link.allInstances.select(l | l.son.name = self.target.name)[0];
	return excludesSourceLinks.eClass().name <> 'Mandatory' and excludesSourceLinks.eClass().name
<> 'Mandatory'
	 or excludesSourceLinks.father.name <> excludesTargetLinks.father.name;
}

operation DependencyLink noExcludesAlternative() : Boolean {
	var excludesSourceLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name)[0];
	var excludesTargetLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name)[0];
	return excludesSourceLinks.source.name <> excludesTargetLinks.source.name or
excludesSourceLinks.source.eClass().name <> 'Alternative';
}

operation DependencyLink noExcludesBetweenMandatoryAndAlternative() : Boolean {
	var excludesSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
	var excludesTargetLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name)[0];
	return excludesSourceLinks.eClass().name <> 'Mandatory' or
excludesTargetLinks.source.eClass().name <> 'Alternative';
}

operation DependencyLink noExcludesBetweenAlternativeAndMandatory() : Boolean {
	var excludesSourceLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name)[0];
	var excludesTargetLinks = Link.allInstances.select(l | l.son.name = self.target.name)[0];
	return excludesSourceLinks.source.eClass().name <> 'Alternative' or
excludesTargetLinks.eClass().name <> 'Mandatory';
```

```
}

operation DependencyLink noExcludesBetweenMandatoryAndOr() : Boolean {
	var excludesSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
	var excludesTargetLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name)[0];
	return excludesSourceLinks.eClass().name <> 'Mandatory' or
excludesTargetLinks.source.eClass().name <> 'Or'
		or AggregationToFeatureLink.allInstances.select(l | l.source =
excludesTargetLinks.source).size() > 2;
}

operation DependencyLink noExcludesBetweenOrAndMandatory() : Boolean {
	var excludesSourceLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name)[0];
	var excludesTargetLinks = Link.allInstances.select(l | l.son.name = self.target.name)[0];
	return excludesSourceLinks.source.eClass().name <> 'Or' or excludesTargetLinks.eClass().name
<> 'Mandatory'
		or AggregationToFeatureLink.allInstances.select(l | l.source =
excludesSourceLinks.source).size() > 2;
}

//Requires
operation DependencyLink noRequiresMandatory() : Boolean {
	var requiresSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
	var requiresTargetLinks = Link.allInstances.select(l | l.son.name = self.target.name)[0];
	return requiresTargetLinks.eClass().name <> 'Mandatory' or requiresSourceLinks.father.name <>
requiresTargetLinks.father.name;
}

operation DependencyLink noRequiresAlternative() : Boolean {
	var requiresSourceLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name)[0];
	var requiresTargetLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name)[0];
	return requiresSourceLinks.source <> requiresTargetLinks.source or
requiresSourceLinks.source.eClass().name <> 'Alternative';
}

operation DependencyLink noRequiresBetweenMandatoryAndAlternative() : Boolean {
	var requiresSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
	var requiresTargetLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name)[0];
	return requiresSourceLinks.eClass().name <> 'Mandatory' or
requiresTargetLinks.source.eClass().name <> 'Alternative';
}

operation DependencyLink noRequiresBetweenMandatoryAndOptionalAlternative() : Boolean {
	var requiresSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
	var requiresTargetLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name)[0];
```

```
	var featureToAggregation = FeatureToAggregationLink.allInstances.select(l | l.target =
requiresTargetLinks.source)[0];
	var featureLink = Link.allInstances.select(l | l.son.name =
featureToAggregation.source.name)[0];
	return requiresSourceLinks.eClass().name <> 'Mandatory' or
requiresTargetLinks.source.eClass().name <> 'Alternative'
		or featureLink.eClass().name <> 'Optional';
}

operation DependencyLink noRequiresBetweenMandatoryAndOr() : Boolean {
	var requiresSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
	var requiresTargetLinks = AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name)[0];
	return requiresSourceLinks.eClass().name <> 'Mandatory' or
requiresTargetLinks.source.eClass().name <> 'Or';
}

operation DependencyLink noRequiresMandatoryAndOptional() : Boolean {
	var requiresSourceLinks = Link.allInstances.select(l | l.son.name = self.source.name)[0];
	var requiresTargetLinks = Link.allInstances.select(l | l.son.name = self.target.name)[0];
	return requiresSourceLinks.eClass().name <> 'Mandatory' or requiresTargetLinks.eClass().name
<> 'Optional'
	 or requiresSourceLinks.father.name <> requiresTargetLinks.father.name;
}

context Excludes {
	constraint noExcludesBetweenMandatoryAndOpcional {
		guard: Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
			and Link.allInstances.select(l | l.son.name = self.target.name).size() > 0
		check: self.noExcludesMandatories()

		message: 'Mandatory features cant be the source or target of an Excludes dependency
with other feature that shares the same father'
	}

	constraint noExcludesAndAlternative {
		guard: AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name).size() > 0
			and AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name).size() > 0
		check: self.noExcludesAlternative()

		message: 'This Excludes Dependency is Redundant'
	}

	constraint noExcludesBetweenMandatoryAndAlternative {
		guard: Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
			and AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name).size() > 0
		check: self.noExcludesBetweenMandatoryAndAlternative()

		message: 'The feature ' + self.target.name + ' is a Dead Feature'
```

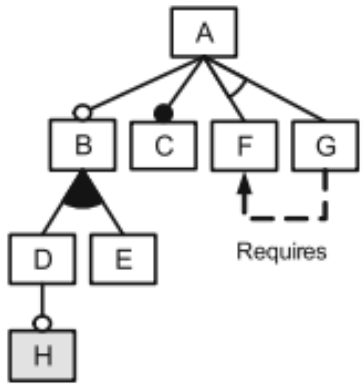
```
	}

	constraint noExcludesBetweenAlternativeAndMandatory{
		guard: AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name).size() > 0
			and Link.allInstances.select(l | l.son.name = self.target.name).size() > 0
		check: self.noExcludesBetweenAlternativeAndMandatory()

		message: 'The feature ' + self.source.name + ' is a Dead Feature'
	}

	constraint noExcludesBetweenMandatoryAndOr {
		guard: Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
			and AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name).size() > 0
		check: self.noExcludesBetweenMandatoryAndOr()

		message: 'The Feature ' + self.target.name + ' is a False Optional Feature'
	}

	constraint noExcludesBetweenOrAndMandatory{
		guard: AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name).size() > 0
			and Link.allInstances.select(l | l.son.name = self.target.name).size() > 0
		check: self.noExcludesBetweenOrAndMandatory()

		message: 'The Feature ' + self.source.name + ' is a False Optional Feature'
	}
}

context Requires{
	constraint noRequiresAndAlternative{
		guard: AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.source.name).size() > 0
			and AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name).size() > 0
		check: self.noRequiresAlternative()

		message: 'The Feature ' + self.source.name + ' is a Dead Feature'
	}

	constraint noRequiresAndMandatory {
		guard: Link.allInstances.select(l | l.son.name = self.target.name).size() > 0
			and Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
		check: self.noRequiresMandatory()

		message: 'Mandatory Features being the target of a Requires Dependency whose source
shares the same father is Redundant'
	}

	constraint noRequiresBetweenMandatoryAndOptional {
		guard: Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
```

```
                    and Link.allInstances.select(l | l.son.name = self.target.name).size() > 0

             check: self.noRequiresMandatoryAndOptional()

             message: 'This Requires Dependency makes the non-target Feature (from the same Or node)
a false optional feature'
       }

       constraint onlyOneRequires {
             check: Requires.allInstances.select(i | i.target.name = self.target.name).size() = 1

             message: 'More than one Requires Dependency is Redundant'
       }

       constraint noRequiresBetweenMandatoryAndAlternative {
             guard: Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
                    and AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name).size() > 0
             check: self.noRequiresBetweenMandatoryAndAlternative()

             message: 'This Requires Dependency makes the non-target Features (from the same
Alternative node) Dead Features'
       }

       constraint noRequiresBetweenMandatoryAndOptionalAlternative {
             guard: Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
                    and AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name).size() > 0
             check: self.noRequiresBetweenMandatoryAndOptionalAlternative()

             message: 'This Requires Dependency makes ' + self.target.name + ' a false optional
feature'
       }

             constraint noRequiresBetweenMandatoryAndOr {
             guard: Link.allInstances.select(l | l.son.name = self.source.name).size() > 0
                    and AggregationToFeatureLink.allInstances.select(l | l.target.name =
self.target.name).size() > 0
             check: self.noRequiresBetweenMandatoryAndOr()

             message: 'This Requires Dependency makes ' + self.target.name + ' a false optional
feature'
       }

       constraint noRequiresBetweenFatherAndSon {
             check: Link.allInstances().select(l | l.father.name = self.source.name and l.son.name =
self.target.name).size() = 0

             message: 'Father and son features cant be related by a dependency relationship'
       }
}
```

```
context Alternative {
	constraint eachAlternativeHasOnlyOneFatherFeature {
		check: FeatureToAggregationLink.allInstances.select(l | l.target = self).size() = 1

		message: 'An Alternative Node can only have one Father Feature'
	}
}

context Or {
	constraint eachOrHasOnlyOneFatherFeature {
		check: FeatureToAggregationLink.allInstances.select(l | l.target = self).size() = 1

		message: 'An Or Node can only have one Father Feature'
	}
}
```

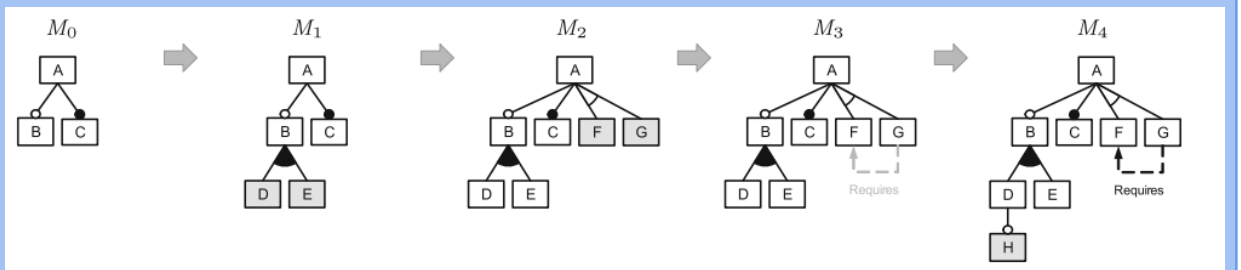
2. Determining the number of possible variants in a Feature Model

Determine the variants for the following feature model:



Answer:

Let's do a stepwise and incremental approach:



M_0:
$p_1 = \{A,C\}$
$p_2 = \{A,B,C\}$

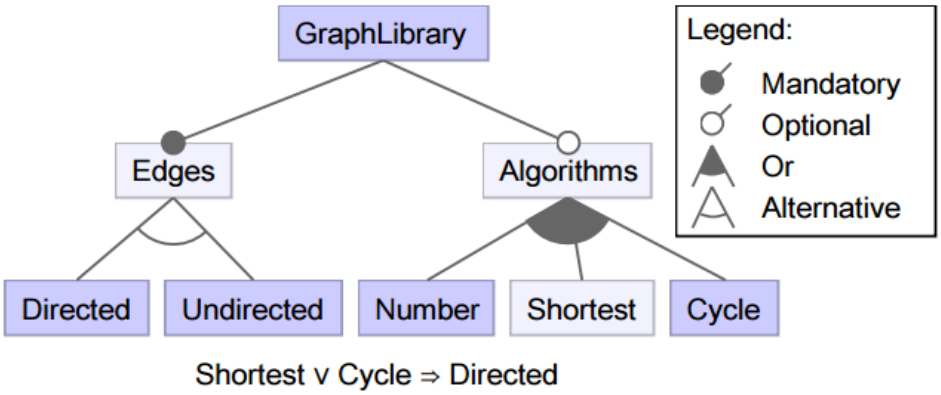
M_1:
$p_1 = \{A,C\}$
$p_2 = \{A,B,C,\mathbf{D}\}$
$p_3 = \{\mathbf{A,B,C,E}\}$
$p_4 = \{\mathbf{A,B,C,D,E}\}$

M_2:
$p_1 = \{A,C,\mathbf{F}\}$
$p_2 = \{A,B,C,D,\mathbf{F}\}$
$p_3 = \{A,B,C,E,\mathbf{F}\}$
$p_4 = \{A,B,C,D,E,\mathbf{F}\}$
$p_5 = \{\mathbf{A,C,G}\}$
$p_6 = \{\mathbf{A,B,C,D,G}\}$
$p_7 = \{\mathbf{A,B,C,E,G}\}$
$p_8 = \{\mathbf{A,B,C,D,E,G}\}$

M_3:
$p_1 = \{A,C,F\}$
$p_2 = \{A,B,C,D,F\}$
$p_3 = \{A,B,C,E,F\}$
$p_4 = \{A,B,C,D,E,F\}$
~~$p_5 = \{A,C,G\}$~~
~~$p_6 = \{A,B,C,D,G\}$~~
~~$p_7 = \{A,B,C,E,G\}$~~
~~$p_8 = \{A,B,C,D,E,G\}$~~

M_4:
$p_1 = \{A,C,F\}$
$p_2 = \{A,B,C,D,F\}$
$p_3 = \{A,B,C,E,F\}$
$p_4 = \{A,B,C,D,E,F\}$
$p_5 = \{\mathbf{A,B,C,D,F,H}\}$
$p_6 = \{\mathbf{A,B,C,D,E,F,H}\}$

3. Representing the correspondence of the feature diagram to its target semantic domain in Propositional Formulas



Answer:

Remember:

Feature Model	Propositional Formula
Optional feature C_i	$C_i \Rightarrow P$
Mandatory feature C_i	$(C_i \Rightarrow P) \wedge (P \Rightarrow C_i)$
Or-group	$P \Leftrightarrow \bigvee_{1 \leq i \leq n} C_i$
Alternative-group	$(P \Leftrightarrow \bigvee_{1 \leq i \leq n} C_i) \wedge \bigwedge_{i<j} (\neg C_i \vee \neg C_j)$

f_1 excludes f_2	$\neg(f_1 \wedge f_2)$
f_1 requires f_2	$f_1 \Rightarrow f_2$

As a matter of simplification, consider the first letter of the feature name. The solution would be:

(E ⇒ G) ∧
(G ⇒ E) ∧
(A ⇒ G) ∧
(E ⇔ (D ∨ U)) ∧
(¬D ∨ ¬U) ∧
(A ⇔ (N ∨ S ∨ C)) ∧
(S ∨ C ⇒ D) ∧ G