CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Lecture 4b: Analysis of Feature Models

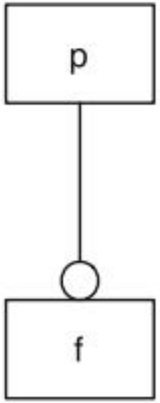
Gregory Gay

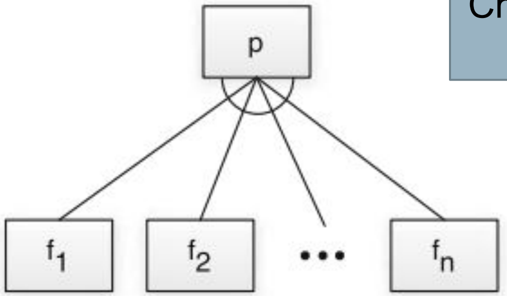
TDA/DIT 594 - November 12, 2020

Feature Diagrams

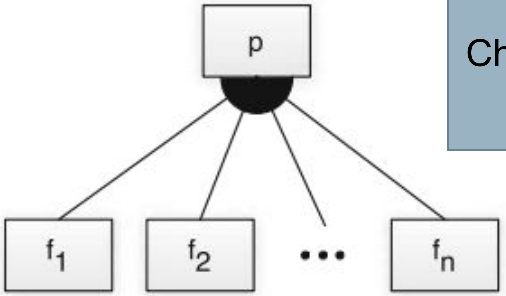
Mandatory Feature

Optional Feature



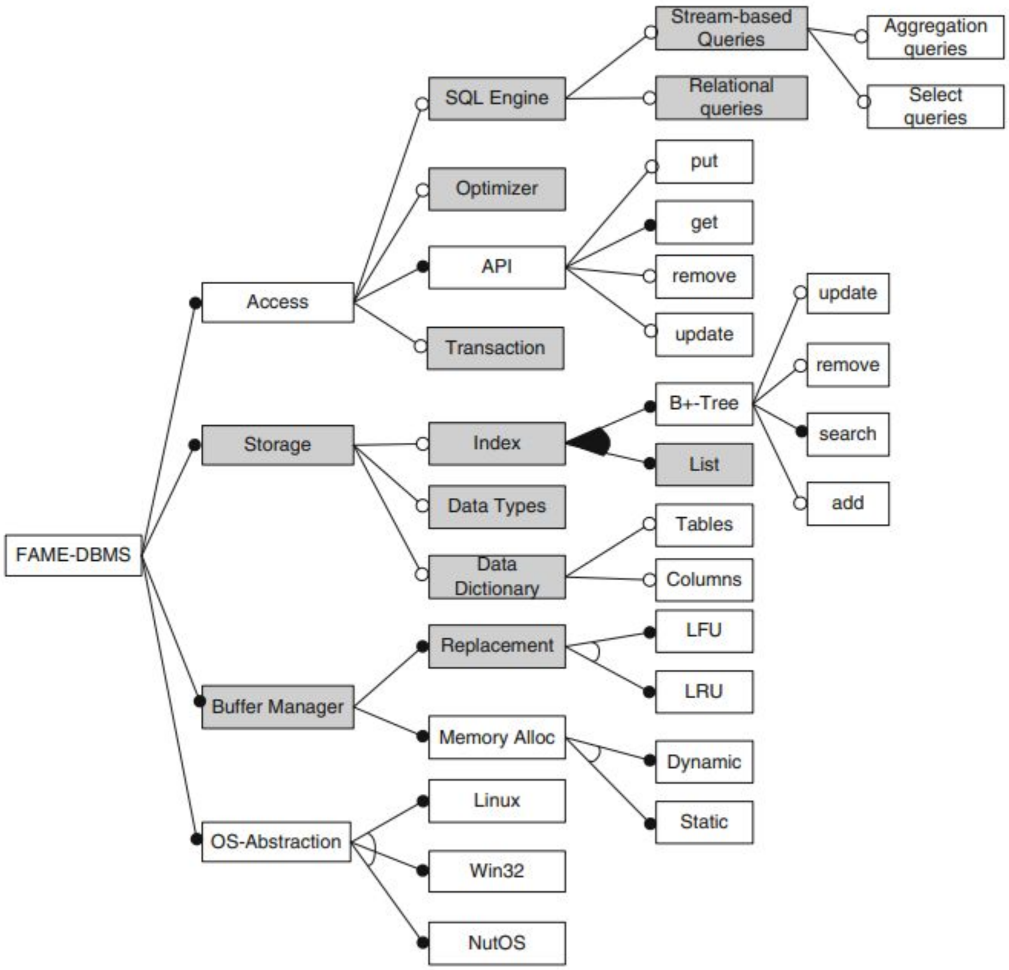


Mutually Exclusive Choice: Choose **exactly** one (**xor**)



Choose **at least** one (**or**)

Example - Data Management

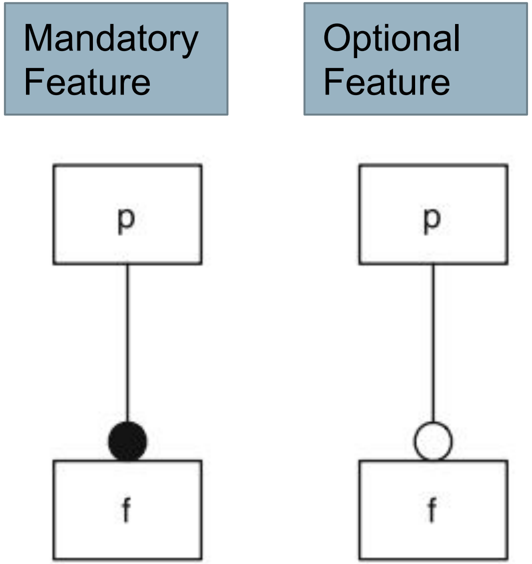


Propositional Logic

- **Cross-tree Constraints** are predicates imposing constraints between features.
  - `DataDictionary ⇒ String`
    - (Storing a data dictionary **requires** support for strings)
  - `MinimumSpanningTree ⇒ Undirected ∧ Weighted`
    - (Computing a Minimum Spanning Tree **requires** support for undirected **and** weighted edges)
  - Constraints over Boolean variables and subexpressions.
    - (i.e., (`NumProcesses >= 5`))

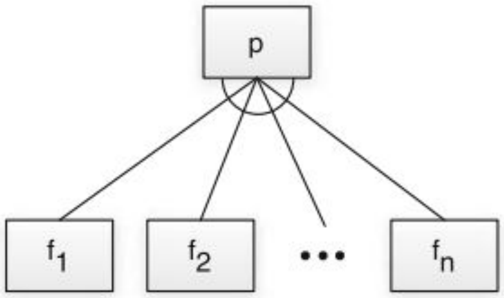
Propositional Logic

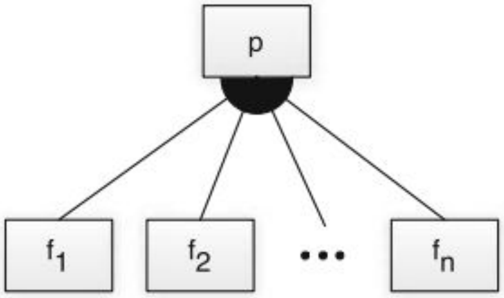
- **Mandatory:** If parent is selected, the child must be.
  - `mandatory(p, f) ≡ f ⇔ p`
- **Optional:** Child may only be chosen if the parent is.
  - `optional(p, f) ≡ f ⇒ p`



Propositional Logic

- **Alternative:** Choose **exactly** one
  - alternative(p, $\{f_1,\ldots,f_n\}$) $\equiv$ $((f_1 \lor \ldots \lor f_n) \Leftrightarrow p) \land_{(fi,fj)} \neg(f_i \land f_j)$
- **Or:** Choose **at least** one
  - or(p, $\{f_1,\ldots,f_n\}$) $\equiv$ $((f_1 \lor \ldots \lor f_n) \Leftrightarrow p)$





Today’s Goals

- Analysis of Feature Models
- Introduction to Boolean Satisfiability (SAT)
  - SAT Solvers

Analysis of Feature Models

Variability-Aware Analysis

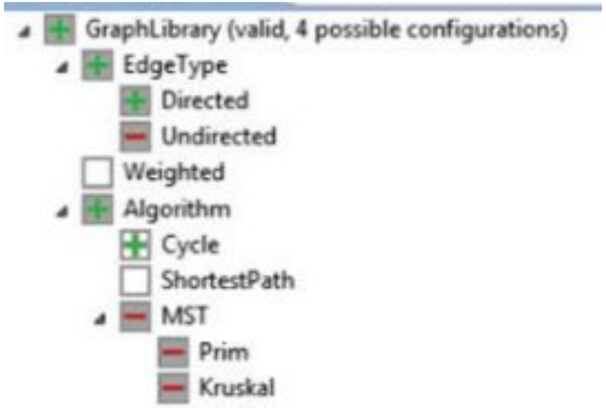
- Verification techniques do not extend to SPLs.
  - More product variations than atoms in the universe.
- Sometimes, can restrict to subset (HP printers).
- **Variability-Aware Analyses** can examine whole product line (or reasonable subset).

Analyses of Feature Models

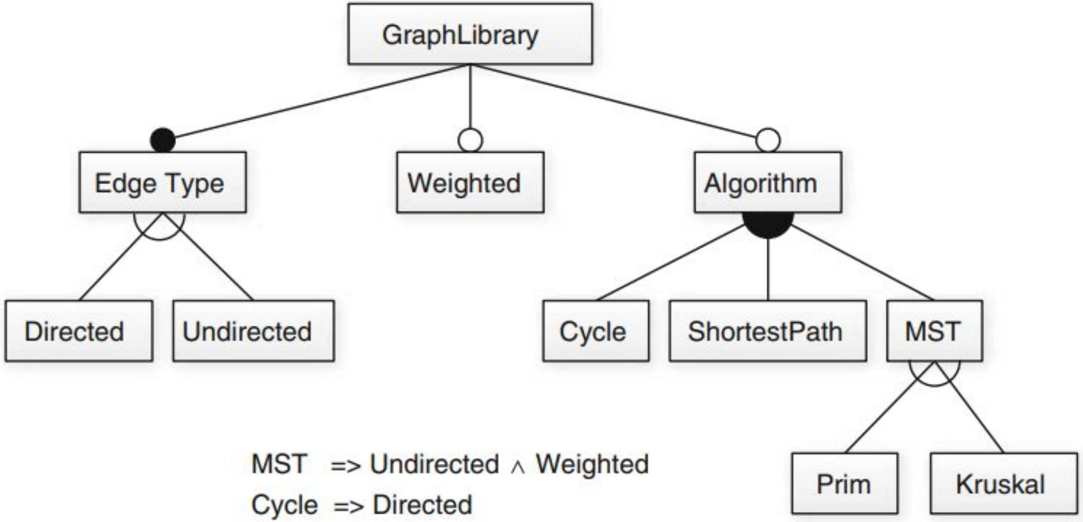
- Is a feature selection valid?
- Is the feature model consistent?
- Do our assumptions hold (testing)?
- Which features are mandatory?
- Which features can never be selected (dead)?
- How many valid selections does model have?
- Are two models equivalent?
- Given partial selection, what must be included?
- What selections give best cost/size/performance?

Valid Feature Selection

- Translate model into a propositional formula φ.
- Assign `true` to each selected feature, `false` to rest.
- Assess whether φ is `true`.
  - If yes, valid selection.

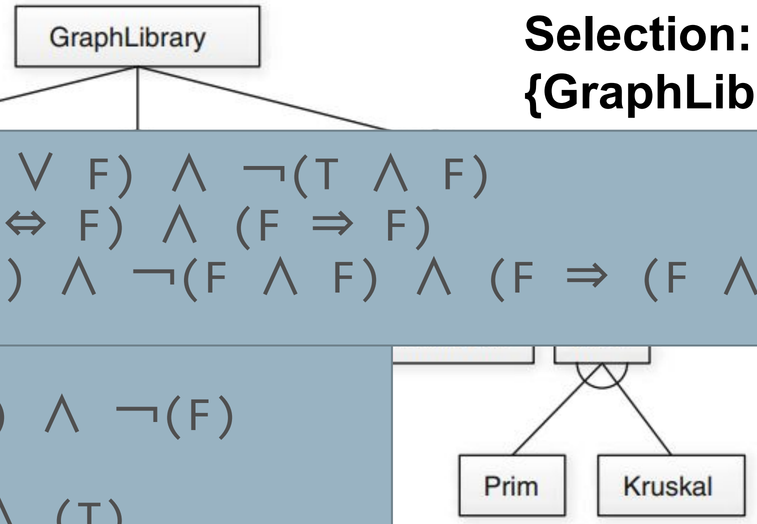


Example - Graph Library



$$\phi = \texttt{GraphLibrary} \wedge \texttt{EdgeType} \wedge (\texttt{Directed} \vee \texttt{Undirected}) \wedge \neg(\texttt{Directed} \wedge \texttt{Undirected})$$
$$\wedge ((\texttt{Cycle} \vee \texttt{ShortestPath} \vee \texttt{MST}) \Leftrightarrow \texttt{Algorithm}) \wedge (\texttt{Cycle} \Rightarrow \texttt{Directed})$$
$$\wedge ((\texttt{Prim} \vee \texttt{Kruskal}) \Leftrightarrow \texttt{MST}) \wedge \neg(\texttt{Prim} \wedge \texttt{Kruskal}) \wedge (\texttt{MST} \Rightarrow (\texttt{Undirected} \wedge \texttt{Weighted}))$$

Example - Graph Library



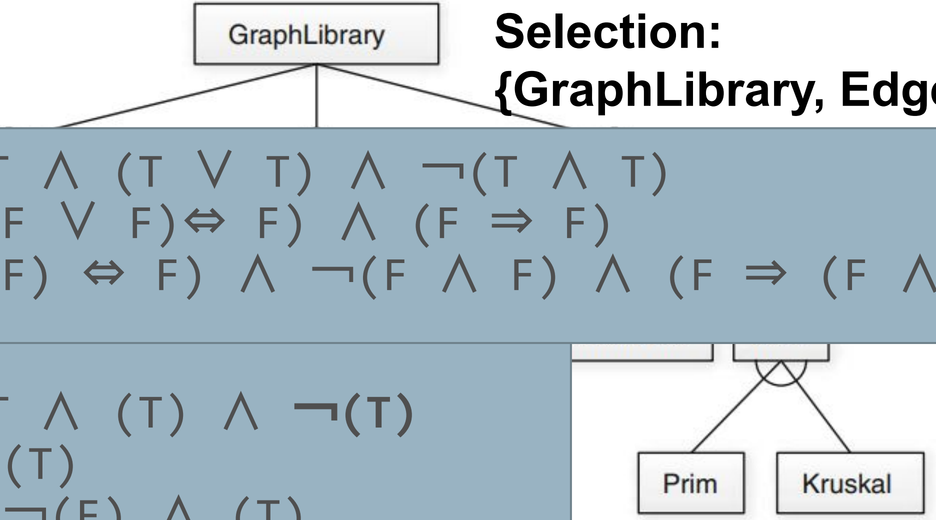
Selection:
{GraphLibrary, EdgeType, Directed}

$$\varphi = T \wedge T \wedge (T \vee F) \wedge \neg(T \wedge F) \wedge ((F \vee F \vee F) \Leftrightarrow F) \wedge (F \Rightarrow F) \wedge ((F \vee F) \Leftrightarrow F) \wedge \neg(F \wedge F) \wedge (F \Rightarrow (F \wedge F))$$

$$\varphi = T \wedge T \wedge (T) \wedge \neg(F) \wedge (T) \wedge (T) \wedge (T) \wedge \neg(F) \wedge (T)$$

$$\begin{aligned}\phi = {} & \mathtt{GraphLibrary} \wedge \mathtt{EdgeType} \wedge (\mathtt{Directed} \vee \mathtt{Undirected}) \wedge \neg(\mathtt{Directed} \wedge \mathtt{Undirected}) \\ & \wedge ((\mathtt{Cycle} \vee \mathtt{ShortestPath} \vee \mathtt{MST}) \Leftrightarrow \mathtt{Algorithm}) \wedge (\mathtt{Cycle} \Rightarrow \mathtt{Directed}) \\ & \wedge ((\mathtt{Prim} \vee \mathtt{Kruskal}) \Leftrightarrow \mathtt{MST}) \wedge \neg(\mathtt{Prim} \wedge \mathtt{Kruskal}) \wedge (\mathtt{MST} \Rightarrow (\mathtt{Undirected} \wedge \mathtt{Weighted}))\end{aligned}$$

Example - Graph Library



Selection:
{GraphLibrary, EdgeType, Directed, Undirected}

$$\varphi = T \land T \land (T \lor T) \land \neg(T \land T) \land ((F \lor F \lor F) \Leftrightarrow F) \land (F \Rightarrow F) \land ((F \lor F) \Leftrightarrow F) \land \neg(F \land F) \land (F \Rightarrow (F \land F))$$

$$\varphi = T \land T \land (T) \land \neg(T) \land (T) \land (T) \land (T) \land \neg(F) \land (T)$$

$$\begin{aligned}\phi = {} & \texttt{GraphLibrary} \land \texttt{EdgeType} \land (\texttt{Directed} \lor \texttt{Undirected}) \land \neg(\texttt{Directed} \land \texttt{Undirected}) \\ & \land ((\texttt{Cycle} \lor \texttt{ShortestPath} \lor \texttt{MST}) \Leftrightarrow \texttt{Algorithm}) \land (\texttt{Cycle} \Rightarrow \texttt{Directed}) \\ & \land ((\texttt{Prim} \lor \texttt{Kruskal}) \Leftrightarrow \texttt{MST}) \land \neg(\texttt{Prim} \land \texttt{Kruskal}) \land (\texttt{MST} \Rightarrow (\texttt{Undirected} \land \texttt{Weighted}))\end{aligned}$$

Consistent Feature Models

- A **consistent** model has 1+ valid selections.
  - **Inconsistent** models do not have any valid selection.
- Contradictory constraints are common.
- Find feature selection that results in φ = `true`
  - NP-complete problem, but SAT solvers can often find solutions quickly.

Boolean Satisfiability (SAT)

- Find assignments to Boolean variables $X_1, X_2, \ldots, X_n$ that results in expression φ evaluating to true.
- Defined over expressions written in **conjunctive normal form**.
  - $\varphi = (X_1 \lor \neg X_2) \land (\neg X_1 \lor X_2)$
  - $(X_1 \lor \neg X_2)$ is a **clause**, made of variables, ¬, ∨
  - Clauses are joined with ∧

Conjunctive Normal Form

- Variables: X_1, X_2, X_3, X_4, X_5
- Clauses (using only $\vee$ (or) and $\neg$ (not)):
  - $(\neg X_2 \vee X_5)$, $(X_1 \vee \neg X_3 \vee X_4)$, $(X_4 \vee \neg X_5)$, $(X_1 \vee X_2)$
- Expression φ joins clauses with $\wedge$ (and)
  - $(\neg X_2 \vee X_5) \wedge (X_1 \vee \neg X_3 \vee X_4) \wedge (X_4 \vee \neg X_5) \wedge (X_1 \vee X_2)$

Boolean Satisfiability

- Find assignment to X_1, X_2, X_3, X_4, X_5 to solve
  - $(\neg X_2 \lor X_5) \land (X_1 \lor \neg X_3 \lor X_4) \land (X_4 \lor \neg X_5) \land (X_1 \lor X_2)$
- One solution: 1, 0, 1, 1, 1
  - $(\neg X_2 \lor X_5) \land (X_1 \lor \neg X_3 \lor X_4) \land (X_4 \lor \neg X_5) \land (X_1 \lor X_2)$
  - $(\neg 0 \lor 1) \land (1 \lor \neg 1 \lor 1) \land (1 \lor \neg 1) \land (1 \lor 0)$
  - $(1) \land (1) \land (1) \land (1)$
  - 1

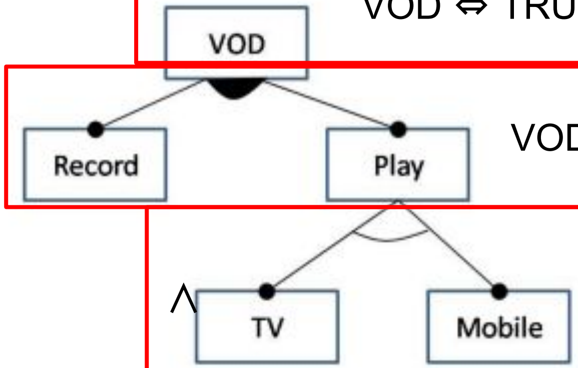
Transformation Rules

- De Morgan's Laws
  - $\neg(X \lor Y) \equiv \neg X \land \neg Y$
  - $\neg(X \land Y) \equiv \neg X \lor \neg Y$
- Distributivity
  - $X \lor (Y \land Z) \equiv (X \lor Y) \land (X \lor Z)$
  - $X \land (Y \lor Z) \equiv (X \land Y) \lor (X \land Z)$
- Double Negation
  - $\neg\neg X \equiv X$

Transformation Rules

- $X \Leftrightarrow Y$
  - X is equivalent to Y
- $\equiv (X \Rightarrow Y) \land (Y \Rightarrow X)$
  - $(X \Rightarrow Y) \equiv (\neg X \lor Y)$
  - If X is true, Y is also true.
  - If X is false, Y can be either true or false.
- $\equiv (\neg X \lor Y) \land (\neg Y \lor X)$

Transformation into CNF



$VOD \Leftrightarrow TRUE \equiv VOD$

$VOD \Leftrightarrow (Record \lor Play)$

$((Mobile \lor TV) \Leftrightarrow Play)$

$\land$

$\neg(Movile \land TV) \equiv$

$(Mobile \Leftrightarrow Play \land \neg TV) \land (TV \Leftrightarrow Play \land \neg Mobile)$

mandatory(p, f) $\equiv f \Leftrightarrow p$

optional(p, f) $\equiv f \Rightarrow p$

alternative(p, {f_1,...,f_n}) $\equiv$
$((f_1 \lor \ldots \lor f_n) \Leftrightarrow p) \land \forall(f_i,f_j)\ \neg(f_i \land f_j)$

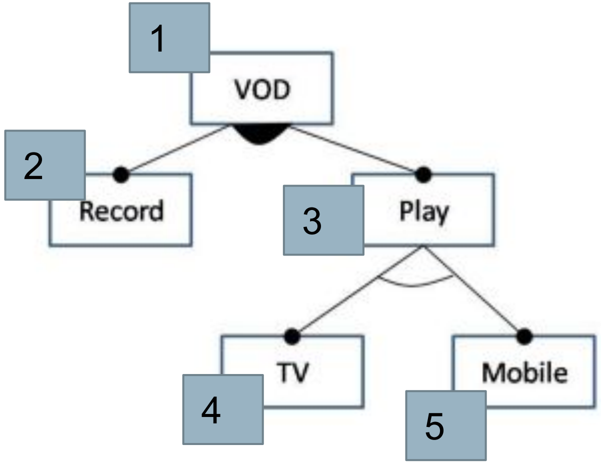
or(p, {f_1,...,f_n}) $\equiv ((f_1 \lor \ldots \lor f_n) \Leftrightarrow p)$

$VOD \land (VOD \Leftrightarrow (Record \lor Play)) \land (Mobile \Leftrightarrow Play \land \neg TV) \land (TV \Leftrightarrow Play \land \neg Mobile)$

Transformation into CNF

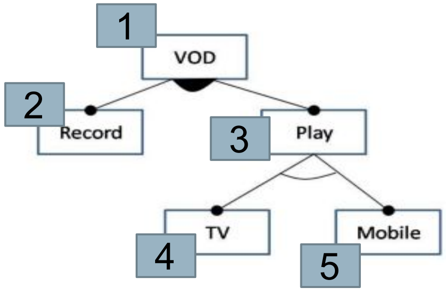
- $VOD \land (VOD \Leftrightarrow (Record \lor Play)) \land (Mobile \Leftrightarrow (Play \land \neg TV)) \land (TV \Leftrightarrow (Play \land \neg Mobile))$
  - $(VOD \Leftrightarrow (Record \lor Play))$
    - $\equiv (VOD \Rightarrow (Record \lor Play)) \land ((Record \lor Play) \Rightarrow VOD)$
    - $\equiv (\neg VOD \lor (Record \lor Play)) \land (\neg(Record \lor Play) \lor VOD)$
    - $\equiv (\neg VOD \lor (Record \lor Play)) \land (\neg Record \lor VOD) \land (\neg Play \lor VOD)$
  - $(Mobile \Leftrightarrow (Play \land \neg TV))$
    - $\equiv (Mobile \lor TV \lor \neg Play) \land (\neg Mobile \lor Play) \land (\neg Mobile \lor \neg TV)$
  - $(TV \Leftrightarrow (Play \land \neg Mobile))$
    - $\equiv (TV \lor Mobile \lor \neg Play) \land (\neg TV \lor Play) \land (\neg TV \lor \neg Mobile)$

DIMACS Format



- Map feature names to integer IDs.
  - VOD = 1
  - Record = 2
  - Play = 3
  - TV = 4
  - Mobile = 5

DIMACS Format



VOD ∧

(¬VOD ∨ (Record ∨ Play)) ∧ (¬Record ∨ VOD) ∧ (¬Play ∨ VOD) ∧

(Mobile ∨ TV ∨ ¬Play) ∧ (¬Mobile ∨ Play) ∧ (¬Mobile ∨ ¬TV) ∧

(TV ∨ Mobile ∨ ¬Play) ∧ (¬TV ∨ Play) ∧ (¬TV ∨ ¬Mobile)

1 ∧

(¬1 ∨ (2 ∨ 3)) ∧ (¬2 ∨ 1) ∧ (¬3∨ 1) ∧

(5 ∨ 4 ∨ ¬3) ∧ (¬5 ∨ 3) ∧ (¬5 ∨ ¬4) ∧

(4 ∨ 5 ∨ ¬3) ∧ (¬4 ∨ 3) ∧ (¬4∨ ¬5)

DIMACS Format

1 ∧
(¬1 ∨ (2 ∨ 3)) ∧ (¬2 ∨ 1) ∧ (¬3 ∨ 1)
∧
(5 ∨ 4 ∨ ¬3) ∧ (¬5 ∨ 3) ∧ (¬5 ∨ ¬4) ∧
(4 ∨ 5 ∨ ¬3) ∧ (¬4 ∨ 3) ∧ (¬4 ∨ ¬5)

- Each clause is stored in a row, with ∧ (AND) omitted.
- Negation (¬) translated into negative (-)

1
-1 ∨ 2 ∨ 3
-2 ∨ 1
-3 ∨ 1
5 ∨ 4 ∨ -3
-5 ∨ 3
-5 ∨ -4
4 ∨ 5 ∨ -3
-4 ∨ 3
-4 ∨ -5

- Remove disjunction signs (∨)
- Add DIMACs header
  - Comments
  - Indicates CNF format
  - Number of variables
  - Number of CNF clauses

```
1
-1 ∨ 2 ∨ 3
-2 ∨ 1
-3 ∨ 1
5 ∨ 4 ∨ -3
-5 ∨ 3
-5 ∨ -4
4 ∨ 5 ∨ -3
-4 ∨ 3
-4∨ -5
```

```
c comments
p cnf 5 10
1
-1 2 3
-2 1
-3 1
5 4 -3
-5 3
-5 -4
4 5 -3
-4 3
-4-5
```

Using a SAT Solver

- Identify assignment that results in true outcome.
  - VOD ∧ (¬VOD ∨ (Record ∨ Play)) ∧ (¬Record ∨ VOD) ∧ (¬Play ∨ VOD) ∧ (Mobile ∨ TV ∨ ¬Play) ∧ (¬Mobile ∨ Play) ∧ (¬Mobile ∨ ¬TV) ∧ (TV ∨ Mobile ∨ ¬Play) ∧ (¬TV ∨ Play) ∧ (¬TV ∨ ¬Mobile)
  - A satisfying assignment: (1, 1, 1, 1, 0)
- Returns satisfying assignment.
  - May return all satisfying assignments found.
  - If not satisfiable, may offer information on why.

Where We Stand

- Feature Models can be expressed using propositional logic formulae (φ).
  - Based on model and cross-tree constaints.
- Valid feature selections result in (φ = true).
- SAT Solvers can identify valid configurations.
  - If none can be found, the model is inconsistent.
  - Enables many different model analyses.

Next Time

- More analysis of feature models

- Assignment 1 due Sunday night. Questions?
- Assignment 2 will be assigned shortly.
  - Watch Canvas.
  - Will cover domain analysis, feature model creation, model analysis.

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Lecture 5: Analysis of Feature Models (Continued)

Gregory Gay
TDA/DIT 594 - November 17, 2020

Where We Stand

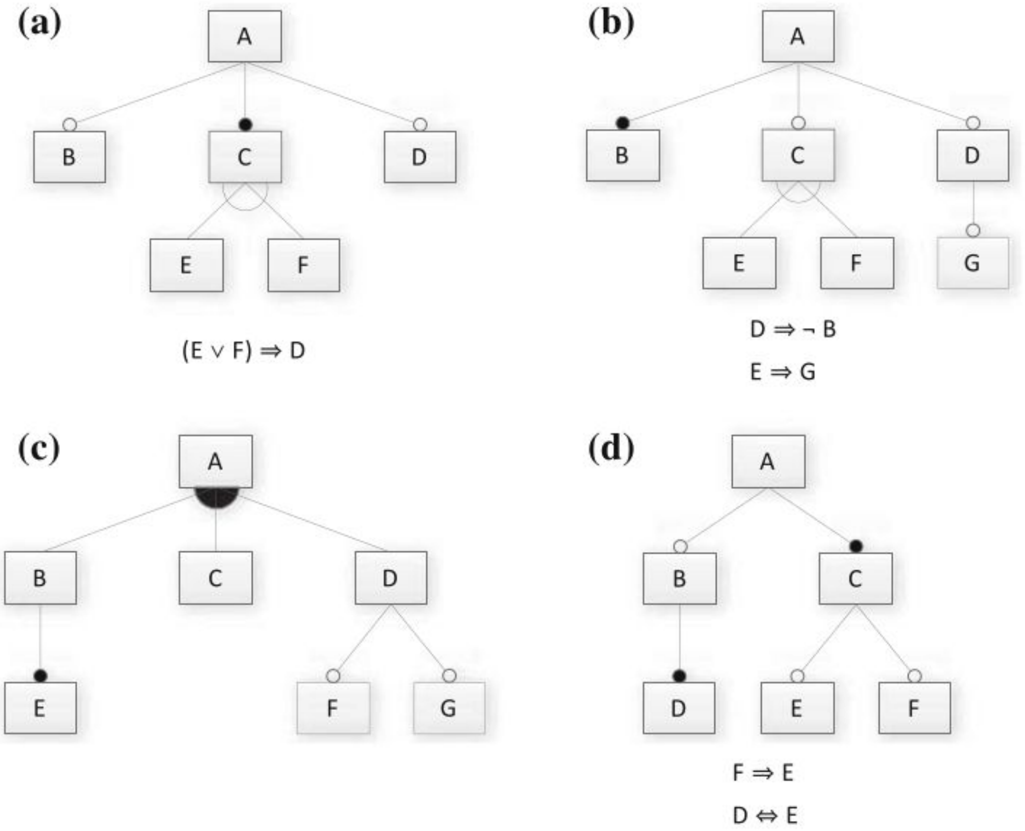
- Feature Models can be expressed using propositional logic formulae (φ).
  - Based on model and cross-tree constaints.
- Valid feature selections result in (φ = true).
- SAT Solvers can identify valid configurations.
  - If none can be found, the model is inconsistent.
  - Enables many different model analyses.

Today's Goals

- More Analysis of Feature Models
- Feature-to-Code Mappings
- Domain Implementation (Analysis of Code)

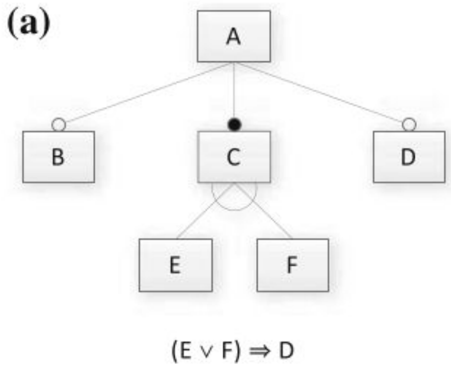
Activity

- Start with A/B.
  - Do C/D if time.
- Translate model into propositional logic formula.
- Provide two valid and two invalid features.
- Is it consistent? If not, why not?



Solution (A)

- Translate model into propositional logic formula.
- Provide two valid and two invalid features.
- Is it consistent? If not, why not?

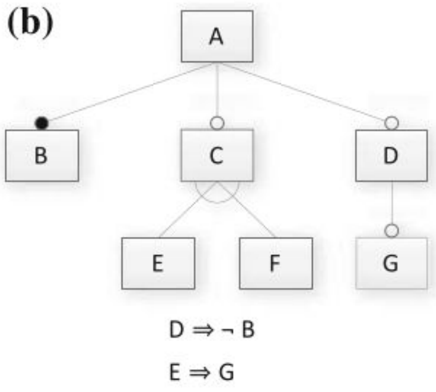


$A \wedge (B \Rightarrow A) \wedge (C \Leftrightarrow A) \wedge (D \Rightarrow A) \wedge ((C \Leftrightarrow (E \vee F)) \wedge \neg(E \wedge F)) \wedge ((E \vee F) \Rightarrow D))$

- Valid: A, B, C, D, F ; A, C, D, E
- Invalid: A, B, C, D, E, F ; A, B, C, E
- Is it consistent: Yes

Solution (B)

- Translate model into propositional logic formula.
- Provide two valid and two invalid features.
- Is it consistent? If not, why not?



A ∧ (B ⇔ A) ∧ (C ⇒ A) ∧ (D ⇒ A) ∧
((C ⇔ (E ∨ F)) ∧ ¬(E ∧ F)) ∧ (G ⇒ D) ∧ (D ⇒ ¬B)
∧
(E ⇒ G)

- Valid: A, B ; A, B, C, F
- Invalid: A, B, D, G ; A, B, C, E
- It is consistent: Yes, but D, E, and G are dead features (because B is mandatory).

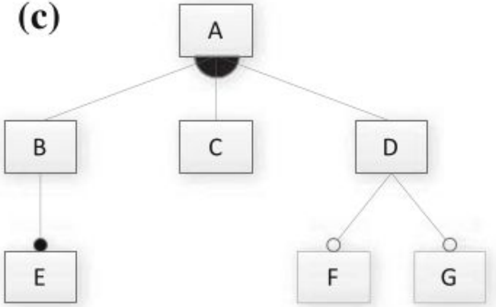
Solution (C)

- Translate model into propositional logic formula.
- Provide two valid and two invalid features.
- Is it consistent? If not, why not?

$A \land ((B \lor C \lor D) \Leftrightarrow A) \land (E \Leftrightarrow B) \land (F \Rightarrow D) \land (G \Rightarrow D)$

- Valid: A, C ; A, B, C, D, E, F, G
- Invalid: A, B, C; A, C, E
- It is consistent: Yes (just remember that B and E need to come as a pair)

(c)



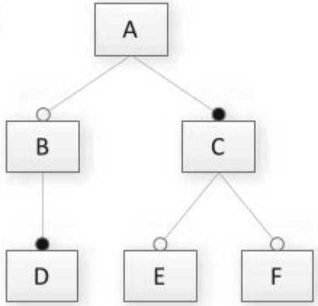
Solution (D)

- Translate model into propositional logic formula.
- Provide two valid and two invalid features.
- Is it consistent? If not, why not?

$A \wedge (B \Rightarrow A) \wedge (C \Leftrightarrow A) \wedge (D \Leftrightarrow B) \wedge (E \Rightarrow C) \wedge (F \Rightarrow C) \wedge$
$(F \Rightarrow E) \wedge (D \Leftrightarrow E)$

- Valid: A, C ; A, B, C, D, E
- Invalid: A, B, C, D ; A, C, F
- It is consistent: Yes, but remember that if you have F, you need E, D, and B as well.

(d)



$F \Rightarrow E$

$D \Leftrightarrow E$

Let’s take a break!

More Analysis of Feature Models

SAT Solver Process

- Express in conjunctive normal form:
  - $\varphi = (\neg x2 \lor x5) \land (x1 \lor \neg x3 \lor x4) \land (x4 \lor \neg x5) \land (x1 \lor x2)$
- Choose assignment based on how it affects each clause it appears in.
  - What happens if we assign x2 = true?
  - If any clauses now false, don't apply that value.
  - Continue until CNF expression is satisfied.

Branch & Bound Algorithm

- Set variable to true or false.
- Apply that value.
- Does value satisfy the clauses that it appears in?
  - If so, assign a value to the next variable.
  - If not, backtrack (bound) and apply the other value.
- Prunes branches of the boolean decision tree as values are applied.

Branch & Bound Algorithm

$\varphi = (\neg x2 \lor x5) \land (x1 \lor \neg x3 \lor x4) \land (x4 \lor \neg x5) \land (x1 \lor x2)$

1. **Set x1 to false.**
   $\varphi = (\neg x2 \lor x5) \land (\mathbf{0} \lor \neg x3 \lor x4) \land (x4 \lor \neg x5) \land (\mathbf{0} \lor x2)$
2. **Set x2 to false.**
   $\varphi = (\mathbf{1} \lor x5) \land (\mathbf{0} \lor \neg x3 \lor x4) \land (x4 \lor \neg x5) \land (\mathbf{0} \lor \mathbf{0})$
3. **Backtrack and set x2 to true.**
   $\varphi = (\mathbf{0} \lor x5) \land (\mathbf{0} \lor \neg x3 \lor x4) \land (x4 \lor \neg x5) \land (\mathbf{0} \lor \mathbf{1})$

DPLL Algorithm

- Set a variable to true/false.
  - Apply that value to the expression.
  - Remove all satisfied clauses.
  - If assignment does not satisfy a clause, then remove that variable from that clause.
  - If this leaves any **unit clauses** (single variable clauses), assign a value that removes those next.
- Repeat until a solution is found.

DPLL Algorithm

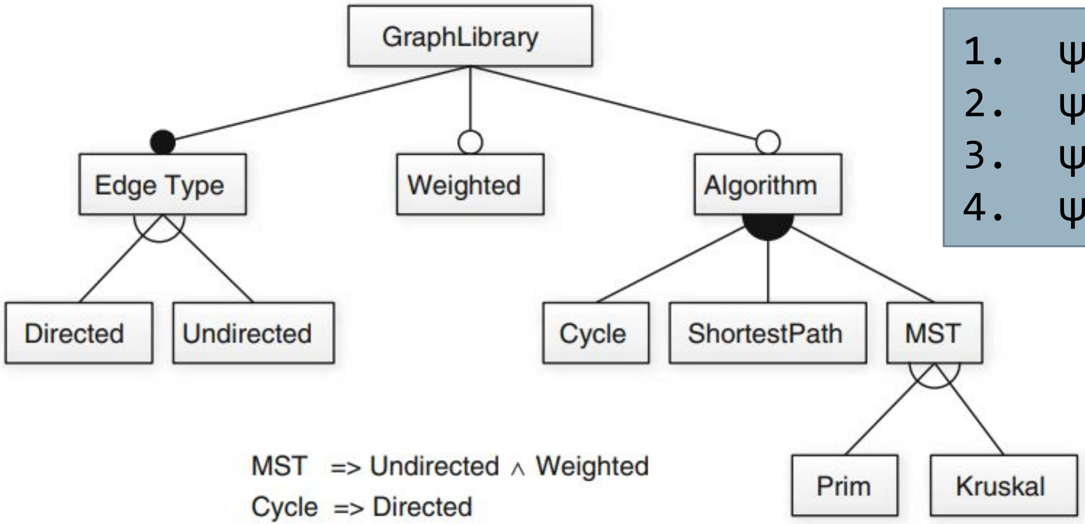
$\varphi = (\neg x2 \lor x5) \land (x1 \lor \neg x3 \lor x4) \land (x4 \lor \neg x5) \land (x1 \lor x2)$

1. **Set x2 to false.**
   $\varphi = (\neg \mathbf{0} \lor x5) \land (x1 \lor \neg x3 \lor x4) \land (x4 \lor \neg x5) \land (x1 \lor \mathbf{0})$
   $\varphi = (x1 \lor \neg x3 \lor x4) \land (x4 \lor \neg x5) \land (x1)$
2. **Set x1 to true.**
   $\varphi = (\mathbf{1} \lor \neg x3 \lor x4) \land (x4 \lor \neg x5) \land (\mathbf{1})$
   $\varphi = (x4 \lor \neg x5)$
3. **Set x4 to false, then x5 to false.**
   $\varphi = (\mathbf{0} \lor \neg x5)$
   $\varphi = (\neg \mathbf{0})$

Testing Facts about Models

- Encode a fact that *should be true* as propositional formula ψ.
- Check whether $\varphi \land \neg\psi$ is satisfiable.
  - Is there a valid feature selection for φ that does not satisfy the constraint ψ?
  - If yes, there is a problem with the model.

Example - Graph Library



1. ψ = Kruskal ⇒ Weighted
2. ψ = Prim ⇒ Weighted
3. ψ = ¬(Prim ∧ Kruskal)
4. ψ = Weighted ⇒ MST

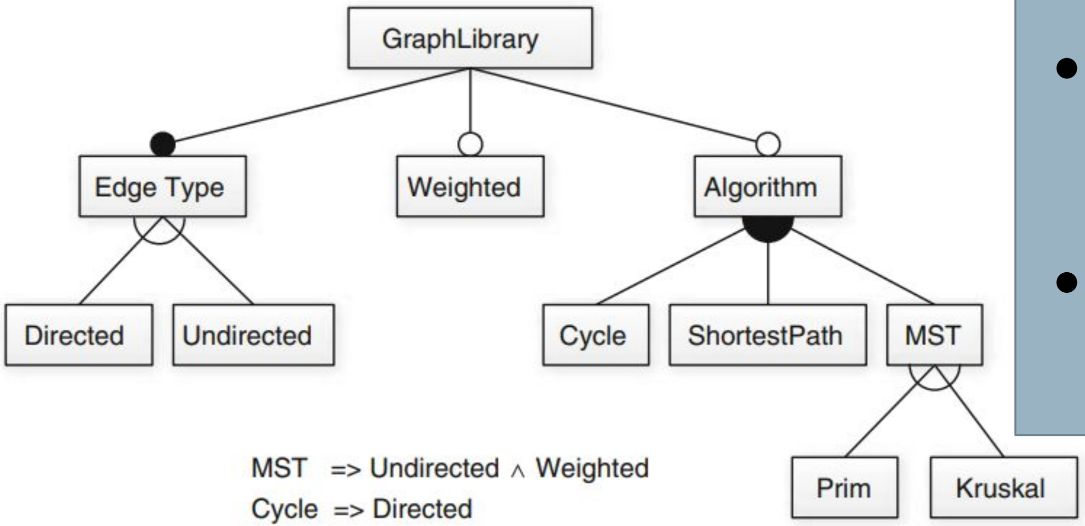
For each fact, assess whether (φ ∧ ¬ψ) is satisfiable.

$$
\begin{aligned}
\phi = {} & \mathtt{GraphLibrary} \land \mathtt{EdgeType} \land (\mathtt{Directed} \lor \mathtt{Undirected}) \land \neg(\mathtt{Directed} \land \mathtt{Undirected}) \\
& \land ((\mathtt{Cycle} \lor \mathtt{ShortestPath} \lor \mathtt{MST}) \Leftrightarrow \mathtt{Algorithm}) \land (\mathtt{Cycle} \Rightarrow \mathtt{Directed}) \\
& \land ((\mathtt{Prim} \lor \mathtt{Kruskal}) \Leftrightarrow \mathtt{MST}) \land \neg(\mathtt{Prim} \land \mathtt{Kruskal}) \land (\mathtt{MST} \Rightarrow (\mathtt{Undirected} \land \mathtt{Weighted}))
\end{aligned}
$$

Dead and Mandatory Features

- A **dead** feature is never used.
- A **mandatory** feature is always used.
- Given model φ and feature F:
  - 1+ valid selection **with F if (φ ∧ F)** is satisfiable.
  - 1+ valid selection **without F if (φ ∧ ¬F)** is satisfiable.
  - Feature is dead if no selection with it (**¬(φ ∧ F)**)
  - Feature is mandatory if no selection without it (**¬(φ ∧ ¬F)**)

Example - Graph Library



- No dead features.
  - If Undirected made mandatory, Directed and Cycle would be dead.
- GraphLibrary and EdgeType are mandatory.

$$\phi = \mathtt{GraphLibrary} \wedge \mathtt{EdgeType} \wedge (\mathtt{Directed} \vee \mathtt{Undirected}) \wedge \neg(\mathtt{Directed} \wedge \mathtt{Undirected})$$
$$\wedge ((\mathtt{Cycle} \vee \mathtt{ShortestPath} \vee \mathtt{MST}) \Leftrightarrow \mathtt{Algorithm}) \wedge (\mathtt{Cycle} \Rightarrow \mathtt{Directed})$$
$$\wedge ((\mathtt{Prim} \vee \mathtt{Kruskal}) \Leftrightarrow \mathtt{MST}) \wedge \neg(\mathtt{Prim} \wedge \mathtt{Kruskal}) \wedge (\mathtt{MST} \Rightarrow (\mathtt{Undirected} \wedge \mathtt{Weighted}))$$

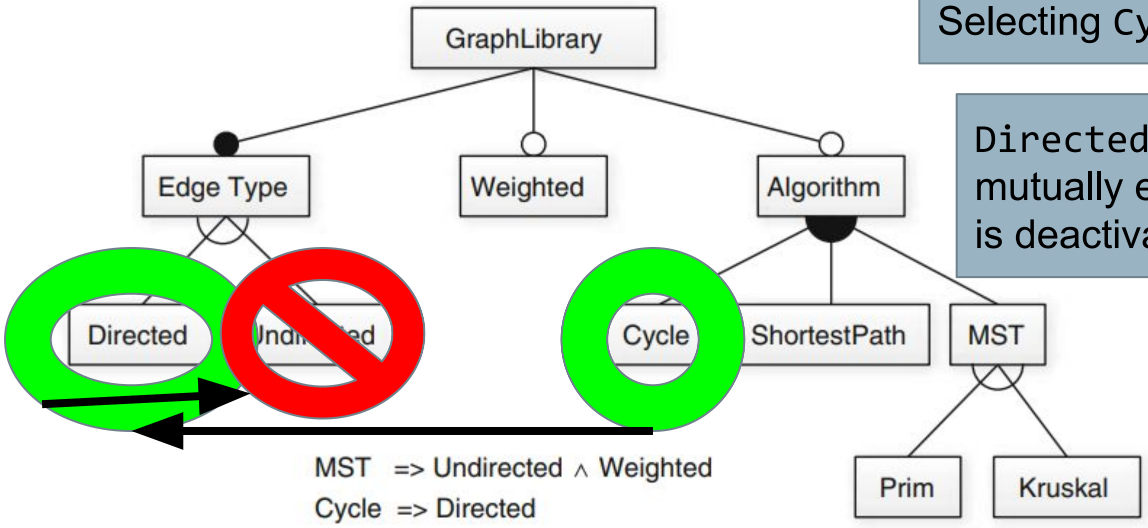
Constraint Propagation

- **Constraint Propagation** - hiding unavailable features after we make **partial selections**.
- Feature selection often iterative:
  - Feature selected, deselected, or no decision made.
- Partial feature selection:
  - Set of selected features ($S \subseteq F$)
  - Set of deselected features ($D \subseteq F$, with $S \cap D = \emptyset$)

Constraint Propagation

- Partial feature selection
  - $pfs(S,D) = \forall(s \in S)\, s \wedge \forall(d \in D)\, \neg d$
- Partial selection is valid if **($\varphi \wedge pfs(S,D)$)** satisfiable
- Feature F **deactivated** if **($\varphi \wedge pfs(S,D) \wedge F$)** is **not** satisfiable.
- Feature F **activated** if **($\varphi \wedge pfs(S,D) \wedge \neg F$)** is **not** satisfiable.

Example - Graph Library



Selecting `Cycle` activated `Directed`.

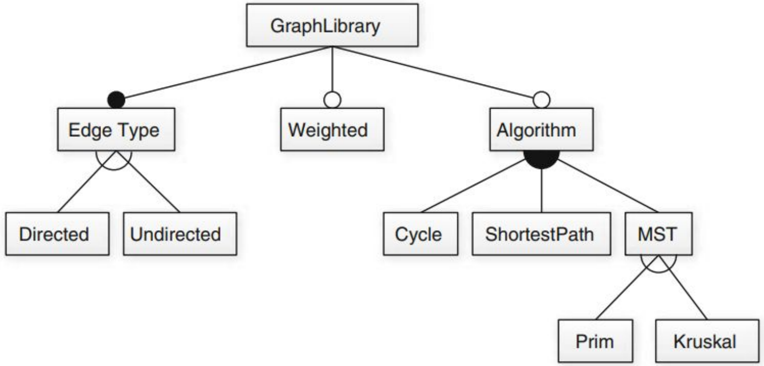
`Directed` and `Undirected` are mutually exclusive, so `Undirected` is deactivated.

$$\begin{aligned}\phi = & \texttt{GraphLibrary} \land \texttt{EdgeType} \land (\texttt{Directed} \lor \texttt{Undirected}) \land \neg(\texttt{Directed} \land \texttt{Undirected}) \\ & \land ((\texttt{Cycle} \lor \texttt{ShortestPath} \lor \texttt{MST}) \Leftrightarrow \texttt{Algorithm}) \land (\texttt{Cycle} \Rightarrow \texttt{Directed}) \\ & \land ((\texttt{Prim} \lor \texttt{Kruskal}) \Leftrightarrow \texttt{MST}) \land \neg(\texttt{Prim} \land \texttt{Kruskal}) \land (\texttt{MST} \Rightarrow (\texttt{Undirected} \land \texttt{Weighted}))\end{aligned}$$

Number of Valid Selections

- Upper bound counted recursively:
  - *count* root(c) = *count*(c)
  - *count* mandatory(c) = *count*(c)
  - *count* optional(c) = *count*(c) + 1
  - *count* and(c_1,..., c_n) = *count*(c_1) ∗ ... ∗ *count*(c_n)
  - *count* alternative(c_1,..., c_n) = *count*(c_1) + ... + *count*(c_n)
  - *count* or(c_1,..., c_n) = (*count*(c_1) + 1) ∗ ... ∗ (*count*(c_n) + 1) − 1
  - *count* leaf = 1

Example - Graph Library



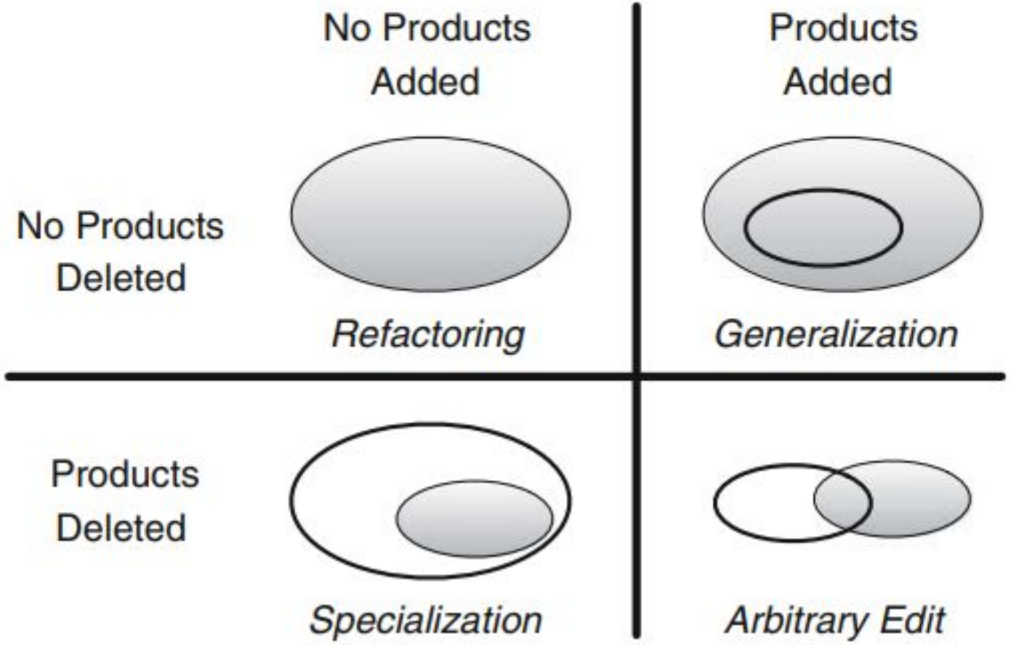
count(f) = 1 //for all leaf nodes

$$
\begin{aligned}
count(EdgeType) &= count(Directed) + count(Undirected) \\
&= 1 + 1 = 2 \\
count(MST) &= count(Prim) + count(Kruskal) \\
&= 1 + 1 = 2 \\
count(Algorithm) &= (count(Cycle) + 1) * \\
&\quad (count(ShortestPath) + 1) * (count(MST) + 1) - 1 \\
&= (1 + 1) * (1 + 1) * (2 + 1) - 1 = 11 \\
count(GraphLibrary) &= count(Mandatory(EdgeType)) * \\
&\quad count(Optional(Weighted)) * \\
&\quad count(Optional(Algorithm)) \\
&= 2 * (1 + 1) * (11 + 1) = 48
\end{aligned}
$$

Number of Valid Selections

- This provides upper bound.
  - Constraints lower the number of actual valid selections.
- Generally, do not need exact number.
  - Upper bound used for estimating worst-case scenarios.

Comparing Feature Models



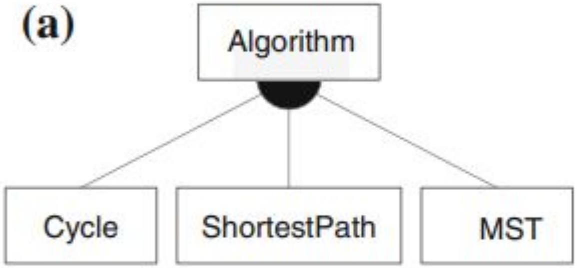
Comparing Feature Models

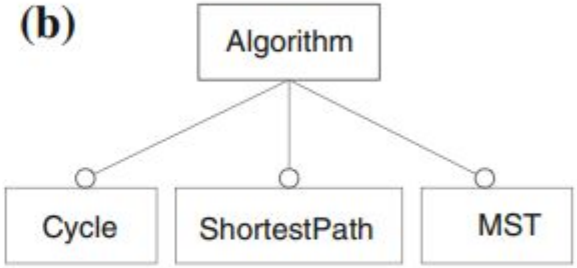
- Models are equivalent if formulae are equivalent.
  - $\neg(\varphi_1 \Leftrightarrow \varphi_2)$ is **not satisfiable**.
- φ_1 is a specialization of φ_2 if $(\varphi_2 \Rightarrow \varphi_1)$
  - and φ_2 is a generalization of φ_1
- SAT solver can compare two models and identify relationships.

Example - Graph Library

Use SAT Solver to prove

$\varphi_{right} \Leftrightarrow \varphi_{left}$





Cycle v ShortestPath v MST

$$\phi_{\texttt{left}} = \text{Algorithm} \wedge ((\text{Cycle} \vee \text{ShortestPath} \vee \text{MST}) \Leftrightarrow \text{Algorithm})$$

$$\phi_{\texttt{right}} = \text{Algorithm} \wedge (\text{Cycle} \Rightarrow \text{Algorithm}) \wedge (\text{ShortestPath} \Rightarrow \text{Algorithm}) \wedge (\text{MST} \Rightarrow \text{Algorithm}) \wedge (\text{Cycle} \vee \text{ShortestPath} \vee \text{MST})$$

Let’s take a break!

Feature-to-Code Mappings

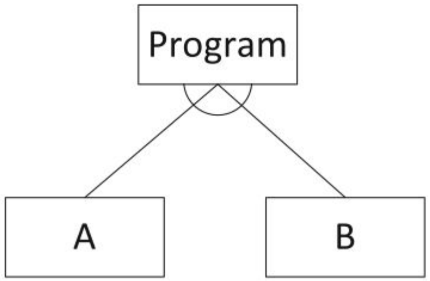
Feature-To-Code Mappings

- Feature models describe the problem space.
- Models are implemented in source code.
- Similar analyses can examine mapping of feature models to code.
  - Which code assets are never used?
  - Which code assets are always used?
  - Which features have no influence on product portfolio?

Dead Code

- Features that can never be incorporated.
- Feature B, in the code, required Feature A to also be selected.
- Model states that A and B are mutually exclusive.

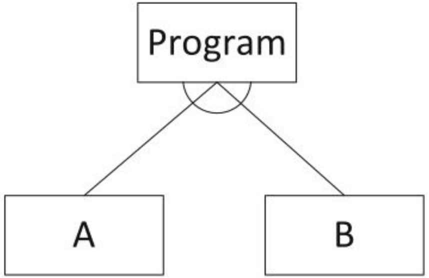
```
line 1
#ifdef A
line 3
#ifdef B
line 5
#endif
#else
line 8
#endif
```



Presence Conditions

- Describes the set of products containing a code fragment.
- **pc(c) = (conditions for c to be included in a product)**
    - pc(line 3) = A
    - pc(line 5) = A ∧ B
    - pc(line 8) = ¬ A

```
1 line 1
2 #ifdef A
3 line 3
4 #ifdef B
5 line 5
6 #endif
7 #else
8 line 8
9 #endif
```

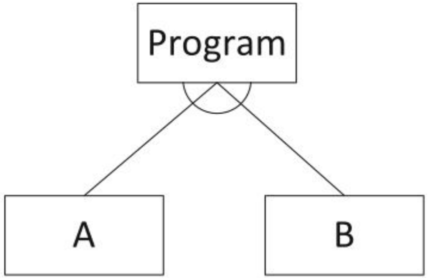


- pc(lines 3-5) = A ∧ B
- pc(lines 3-8) = A ∧ B ∧ ¬A
    - (cannot be included in any product)

Dead Code

- Fragment is dead if never included in any product.
  - φ represents all valid products.
  - Fragment C is dead iff (**φ ∧ pc(C)**) is not satisfiable.

	C	pc()
1	`line 1`	True
2	`#ifdef A`	
3	`line 3`	A
4	`#ifdef B`	
5	`line 5`	A ∧ B
6	`#endif`	
7	`#else`	
8	`line 8`	¬A
9	`#endif`	



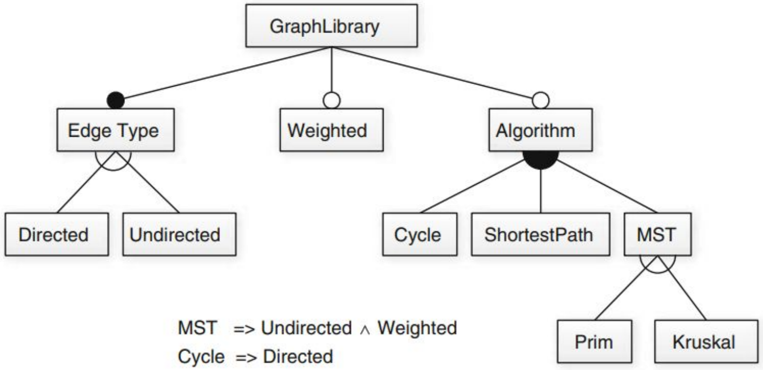
φ = Program ∧ (A ∨ B) ∧ ¬(A ∧ B)

(φ ∧ pc(line 5)) is not satisfiable:

Program ∧ (A ∨ B) ∧ ¬(A ∧ B) ∧ (A ∧ B)

Mandatory Code

- Fragment is mandatory if always included in a product.
  - φ represents all valid products.
  - Fragment C is mandatory iff (φ ∧ ¬pc(C)) is not satisfiable.



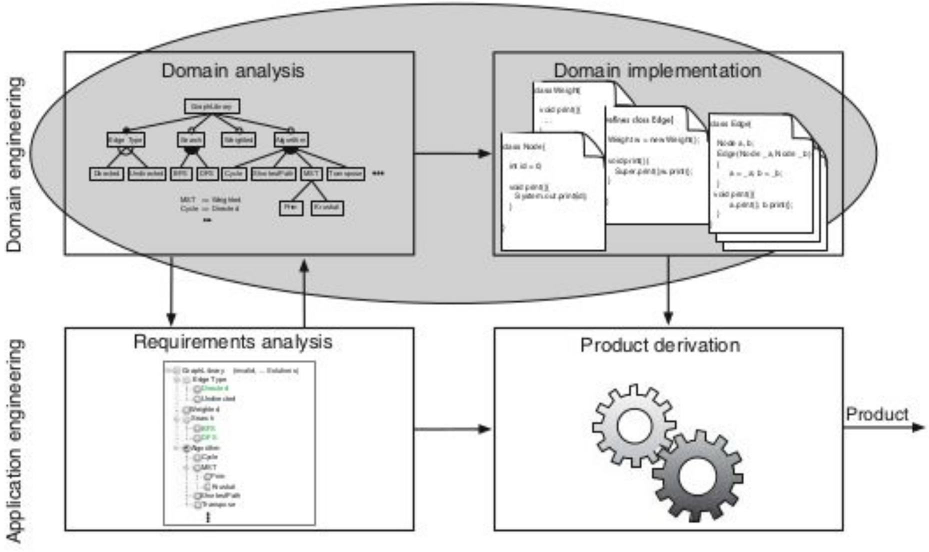
$$\phi = \texttt{GraphLibrary} \wedge \texttt{EdgeType} \wedge (\texttt{Directed} \vee \texttt{Undirected}) \wedge \neg(\texttt{Directed} \wedge \texttt{Undirected}) \wedge ((\texttt{Cycle} \vee \texttt{ShortestPath} \vee \texttt{MST}) \Leftrightarrow \texttt{Algorithm}) \wedge (\texttt{Cycle} \Rightarrow \texttt{Directed}) \wedge ((\texttt{Prim} \vee \texttt{Kruskal}) \Leftrightarrow \texttt{MST}) \wedge \neg(\texttt{Prim} \wedge \texttt{Kruskal}) \wedge (\texttt{MST} \Rightarrow (\texttt{Undirected} \wedge \texttt{Weighted}))$$

If code implemented correctly, the fragment for EdgeType will be mandatory.

Domain Implementation

Analysis of Product Line Code

- Focus on analyzing variability in program structures
- Variability-aware Analyses
  - Traditional analyses (i.e., type checking) extended from one product to entire line.
  - Goal of analyzing whole line in one pass instead of all individual products.



Example: Type Checking

- Verifying and enforcing constraints of data types.
  - Is String being used as Integer?
  - If we call a method, does it return the right type of data?
- Can be checked during compilation or at runtime.
- Same analyses can be applied to other properties.

```
Part1 = 10
Part2 = “Wobuffet”
Sum = Part1 + Part2
```

```
String getName() {
   return “Wobuffet”; }
Part1 = 10
Sum = Part1 + getName()
```

Terminology

- Check **properties** about program or feature model.
  - Type Checking: Does the program have type errors?
  - We assume a property must hold over **all products**.
- **Complete** variability-aware analyses give same results as brute-force analysis.
- **Sound** analyses ensure all violations in domain artifacts hold in concrete products.

Sampling Strategies

- Instead of brute-force, try a subset of products.
- Selection criteria:
  - **Feature Coverage:** All features covered at least once.
  - **Feature-Code Coverage:** All code fragments included at least once.
  - **Pairwise Feature Coverage:** All pairs of features covered at least once.
    - **N-wise Coverage:** All N-way (3-way, 4-way,...) combinations.

Sampling Strategies

- Strategies:
  - **Popular Features:** Focus on what customers use
  - **Domain-Specific:** Base coverage on factors important to product domain.
- Balance between # of analyses and error detection.
  - Sampling is **sound**, but **not complete**.
    - Detected errors hold in products, but not all products tested.

Family-Based Type Checking

- Compiler uses #ifdef annotation to decide what code to include in binary.
- Graph product line, Node class.
  - Features: NAME, NONAME, COLOR.
  - Selecting neither or both NAME/NONAME leads to error.

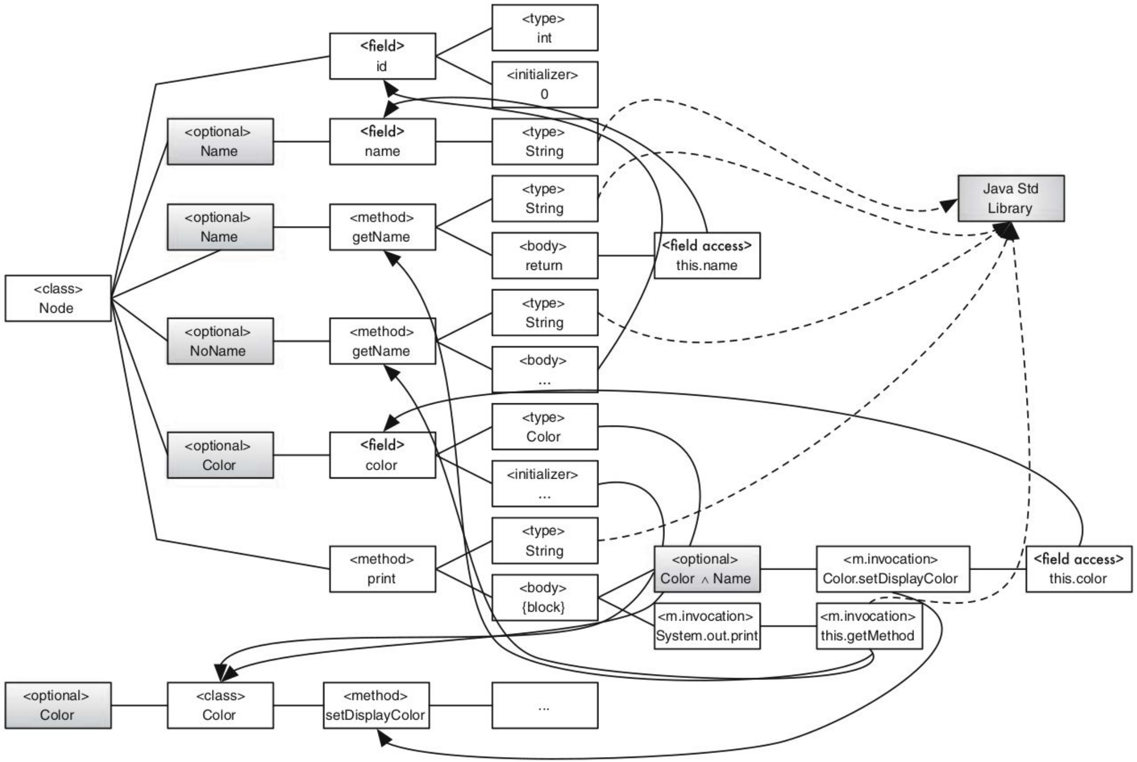
```
class Node {
    int id = 0;

    //#ifdef NAME
    private String name;
    String getName() { return name; }
    //#endif
    //#ifdef NONAME
    String getName() { return String.valueOf(id); }
    //#endif

    //#ifdef COLOR
    Color color = new Color();
    //#endif

    void print() {
        //#if defined(COLOR) && defined(NAME)
        Color.setDisplayColor(color);
        //#endif
        System.out.print(getName());
    }
}
//#ifdef COLOR
class Color {
    static void setDisplayColor(Color c){/*...*/}
}
//#endif
```

Presence Conditions on Structures

- Can identify presence conditions for classes, methods, fields, variables.
  - pc(getName() [line 6]) = NAME
  - pc(getName() [line 9]) = NONAME
  - pc(Color.setDisplayColor(color) [line 18]) = COLOR ∧ NAME
  - pc(System.out.print(getName()) [line 20]) = TRUE ⇒ (NAME ∨ NONAME)
    - Calls getName(), requires at least one to exist.

```
1  class Node {
2      int id = 0;
3
4      //#ifdef NAME
5      private String name;
6      String getName() { return name; }
7      //#endif
8      //#ifdef NONAME
9      String getName() { return String.valueOf(id); }
10     //#endif
11
12     //#ifdef COLOR
13     Color color = new Color();
14     //#endif
15
16     void print() {
17         //#if defined(COLOR) && defined(NAME)
18         Color.setDisplayColor(color);
19         //#endif
20         System.out.print(getName());
21     }
22 }
23 //#ifdef COLOR
24 class Color {
25     static void setDisplayColor(Color c){/*...*/}
26 }
27 //#endif
```

Presence Conditions on Structures



Reachability

- Examine lines reachable from each line to identify presence conditions.
- If NAME ∧ NONAME, error on line 9.
- If ¬NAME ∧ ¬NONAME, error on line 20.

```
1  class Node {
2      int id = 0;
3
4      //#ifdef NAME
5      private String name;
       String getName() { return name; }
       //#endif
8      //#ifdef NONAME
9      String getName() { return String.valueOf(id); }
10     //#endif
11
12     //#ifdef COLOR
13     Color color = new Color();
14     //#endif
15
16     void print() {
17         //#if defined(COLOR) && defined(NAME)
18         Color.setDisplayColor(color);
19         //#endif
20         System.out.print(getName());
23 //#ifdef COLOR
24 class Color {
25     static void setDisplayColor(Color c){/*...*/}
26 }
27 //#endif
```

$\phi \Rightarrow \neg(\mathrm{NAME} \wedge \mathrm{NONAME})$

$\phi \Rightarrow (\mathrm{NONAME} \Rightarrow \top)$

$\phi \Rightarrow (\top \Rightarrow (\mathrm{NAME} \vee \mathrm{NONAME}))$

$\phi \Rightarrow ((\mathrm{COLOR} \wedge \mathrm{NAME}) \Rightarrow \mathrm{COLOR})$

```
Found 2 type errors:
 - [NAME & NONAME] file Node.java:9
       'getName()' is already defined in 'Node'
 - [!NAME & !NONAME] file Node.java:20
       cannot resolve method 'getName()'
```

Reachability Conditions

- When a call is made from source to target, a valid target must exist.
  - $\varphi \Rightarrow (pc(s) \Rightarrow \bigvee_{t \in T} pc(t))$
- If negation of this constraint can be satisfied, there are feature selections that will not compile.
  - SAT solver can identify selections where there are no valid targets for a call from a source.

Reachability

Construct	Source	Target	Constraint
String (type reference)	5	JSL	$\phi \Rightarrow (\mathsf{Name} \Rightarrow \top)$
String (type reference)	6	JSL	$\phi \Rightarrow (\mathsf{Name} \Rightarrow \top)$
name (field access)	6	5	$\phi \Rightarrow (\mathsf{Name} \Rightarrow \mathsf{Name})$
String (type reference)	9	JSL	$\phi \Rightarrow (\mathsf{NoName} \Rightarrow \top)$
String.valueOf (method invocation)	9	JSL	$\phi \Rightarrow (\mathsf{NoName} \Rightarrow \top)$
id (field access)	9	2	$\phi \Rightarrow (\mathsf{NoName} \Rightarrow \top)$
Color (type reference)	13	24	$\phi \Rightarrow (\mathsf{Color} \Rightarrow \mathsf{Color})$
Color (instantiation)	13	24	$\phi \Rightarrow (\mathsf{Color} \Rightarrow \mathsf{Color})$
Color.setDisplayColor (method inv.)	18	25	$\phi \Rightarrow ((\mathsf{Color} \wedge \mathsf{Name}) \Rightarrow \mathsf{Color})$
color (field access)	18	13	$\phi \Rightarrow ((\mathsf{Color} \wedge \mathsf{Name}) \Rightarrow \mathsf{Color})$
System.out (field access)	20	JSL	$\phi \Rightarrow (\top \Rightarrow \top)$
PrintStream.print (method invocation)	20	JSL	$\phi \Rightarrow (\top \Rightarrow \top)$
getName (method invocation)	20	6, 9	$\phi \Rightarrow (\top \Rightarrow (\mathsf{Name} \vee \mathsf{NoName}))$
Color (type reference)	25	24	$\phi \Rightarrow (\mathsf{Color} \Rightarrow \mathsf{Color})$
getName (method redeclaration)	9	6	$\phi \Rightarrow \neg(\mathsf{Name} \wedge \mathsf{NoName})$

JSL = Java Standard Library

```
class Node {
    int id = 0;

    //#ifdef NAME
    private String name;
    String getName() { return name; }
    //#endif
    //#ifdef NONAME
    String getName() { return String.valueOf(id); }
    //#endif

    //#ifdef COLOR
    Color color = new Color();
    //#endif

    void print() {
        //#if defined(COLOR) && defined(NAME)
        Color.setDisplayColor(color);
        //#endif
        System.out.print(getName());

//#ifdef COLOR
class Color {
    static void setDisplayColor(Color c){/*...*/}
}
//#endif
```

$\phi \Rightarrow \neg(\mathtt{NAME} \wedge \mathtt{NONAME})$

$\phi \Rightarrow (\mathtt{NONAME} \Rightarrow \top)$

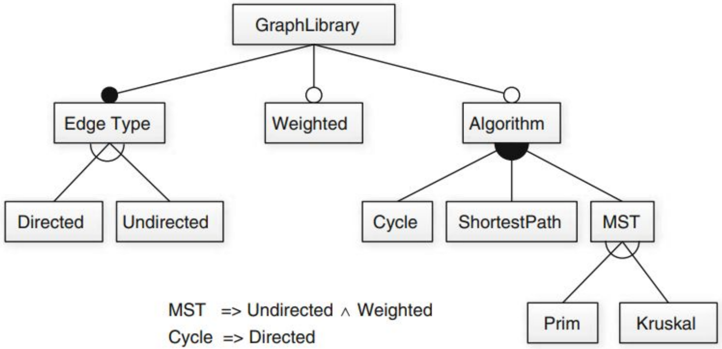
$\phi \Rightarrow (\top \Rightarrow (\mathtt{NAME} \vee \mathtt{NONAME}))$

$\phi \Rightarrow ((\mathtt{COLOR} \wedge \mathtt{NAME}) \Rightarrow \mathtt{COLOR})$

```
Found 2 type errors:
 - [NAME & NONAME] file Node.java:9
        'getName()' is already defined in 'Node'
 - [!NAME & !NONAME] file Node.java:20
        cannot resolve method 'getName()'
```

Beyond Type Checking

- Same approach can be used for checking many properties.
- Lift from individual product to whole line.
  - Analyze shared code once.
  - Reason about configurations using logic and SAT solvers.



$$\begin{aligned}\phi =\ & \mathtt{GraphLibrary} \wedge \mathtt{EdgeType} \wedge (\mathtt{Directed} \vee \mathtt{Undirected}) \wedge \neg(\mathtt{Directed} \wedge \mathtt{Undirected}) \\ & \wedge ((\mathtt{Cycle} \vee \mathtt{ShortestPath} \vee \mathtt{MST}) \Leftrightarrow \mathtt{Algorithm}) \wedge (\mathtt{Cycle} \Rightarrow \mathtt{Directed}) \\ & \wedge ((\mathtt{Prim} \vee \mathtt{Kruskal}) \Leftrightarrow \mathtt{MST}) \wedge \neg(\mathtt{Prim} \wedge \mathtt{Kruskal}) \wedge (\mathtt{MST} \Rightarrow (\mathtt{Undirected} \wedge \mathtt{Weighted}))\end{aligned}$$

We Have Learned

- Feature Models can be expressed using propositional logic formulae (φ).
  - Based on model and cross-tree constaints.
- Valid feature selections result in (φ = true).
- SAT Solvers can identify valid configurations.
  - If none can be found, the model is inconsistent.
  - Enables many different model analyses.

We Have Learned

- Feature-Model Analysis
  - Check properties of model are true.
  - Dead and mandatory features
  - Effects of partial selections
  - Comparisons between two models
- Mapping of models and code
  - Dead and mandatory code
- Implementation analysis
  - Do called assets exist and return the correct data type?

Next Time

- Implementation of variability using design patterns.

- Assignment 2 is out now!
  - See description on Canvas.
  - Questions?

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY