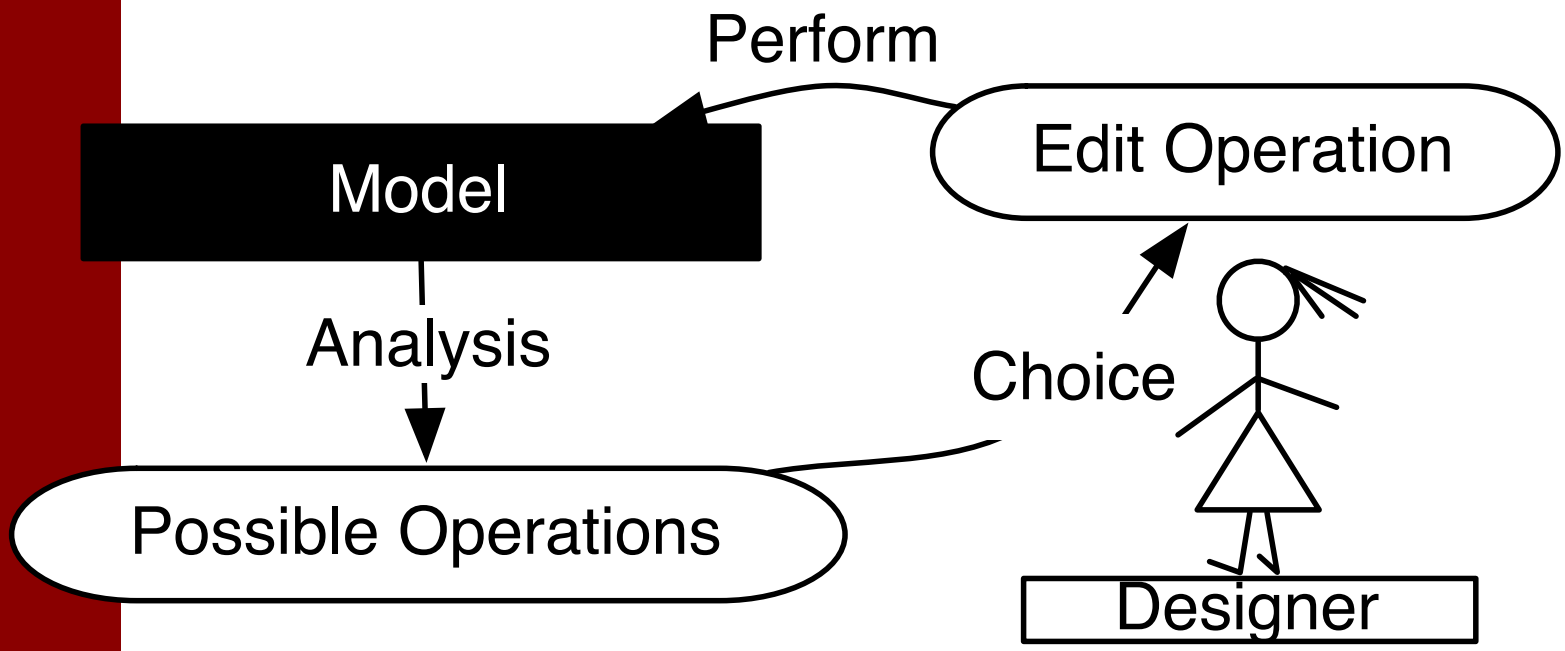
Interactive **M**odel **D**erivation with **E**xternal **C**onstraints

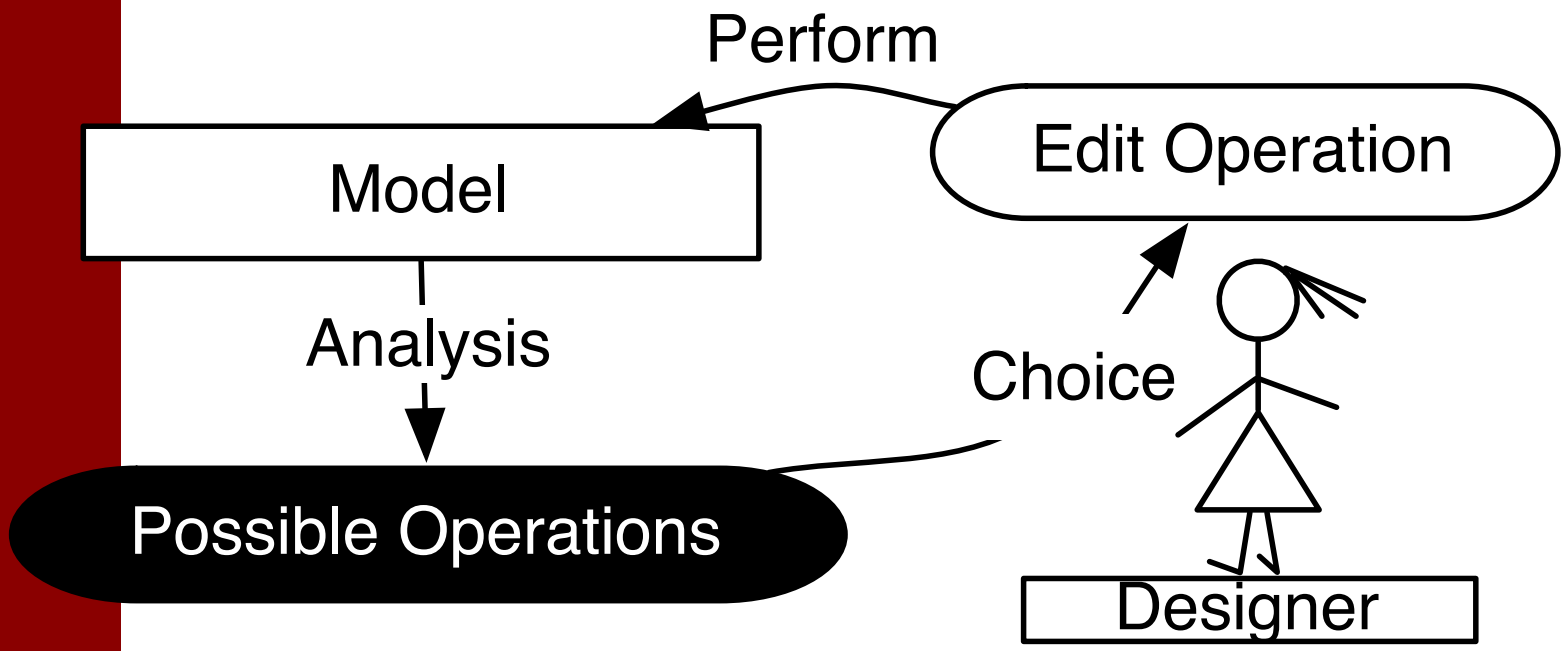
Mikoláš **J**anota
University College Dublin, Ireland

Victoria **K**uzina
Andrzej **W**ąsowski
IT University of Copenhagen, Denmark

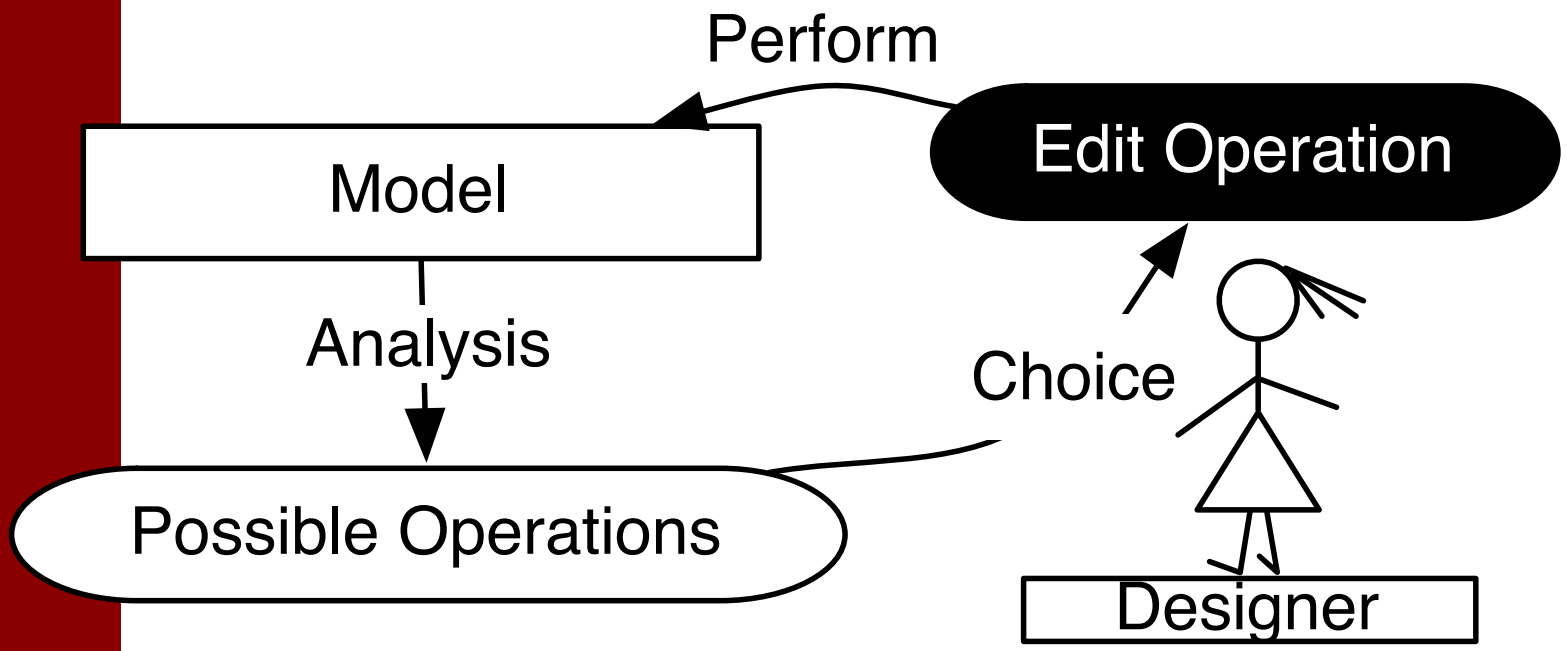
Interactive **M**odel **D**erivation



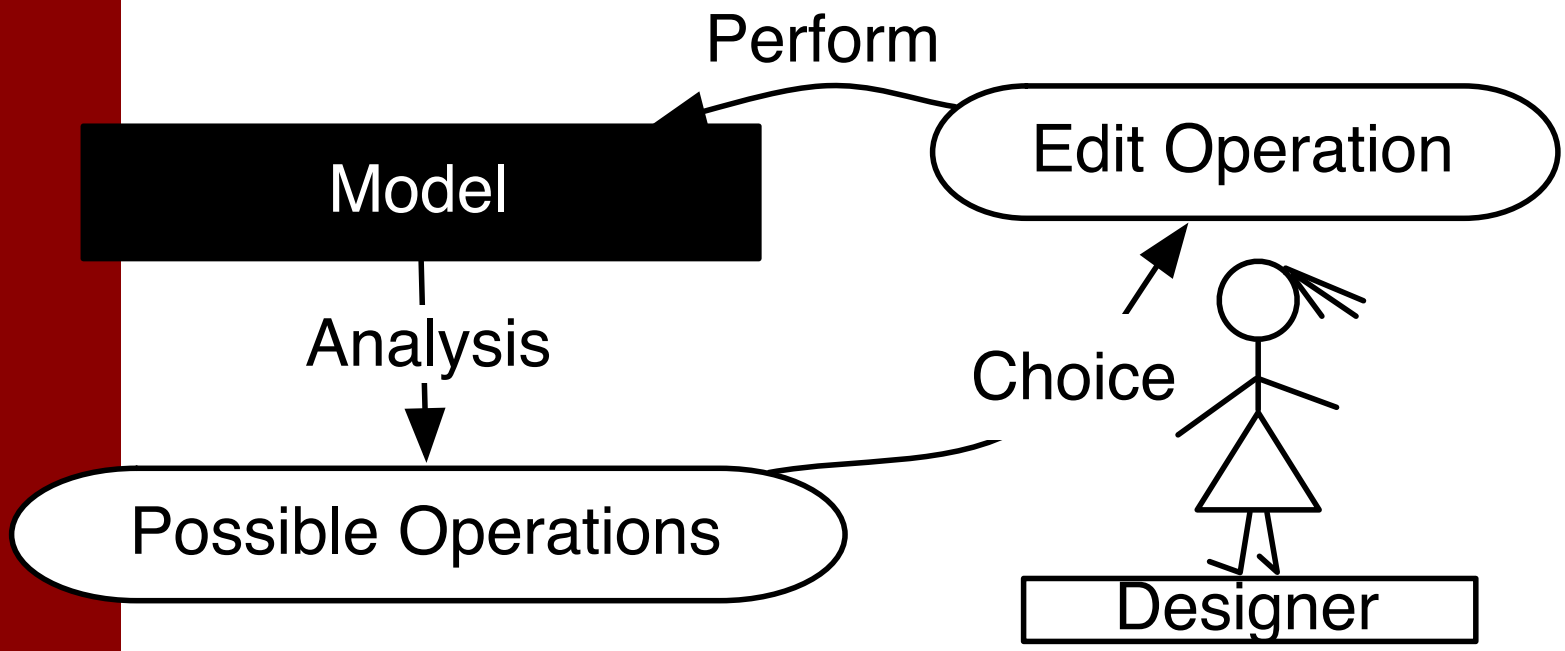
Interactive **M**odel **D**erivation



Interactive Model Derivation



Interactive **M**odel **D**erivation



Model **C**arving

Model **C**arving

Model **C**arving

Model **C**arving

Classification of **D**erivation

- *Soundness-preserving* derivation seen in instance derivation

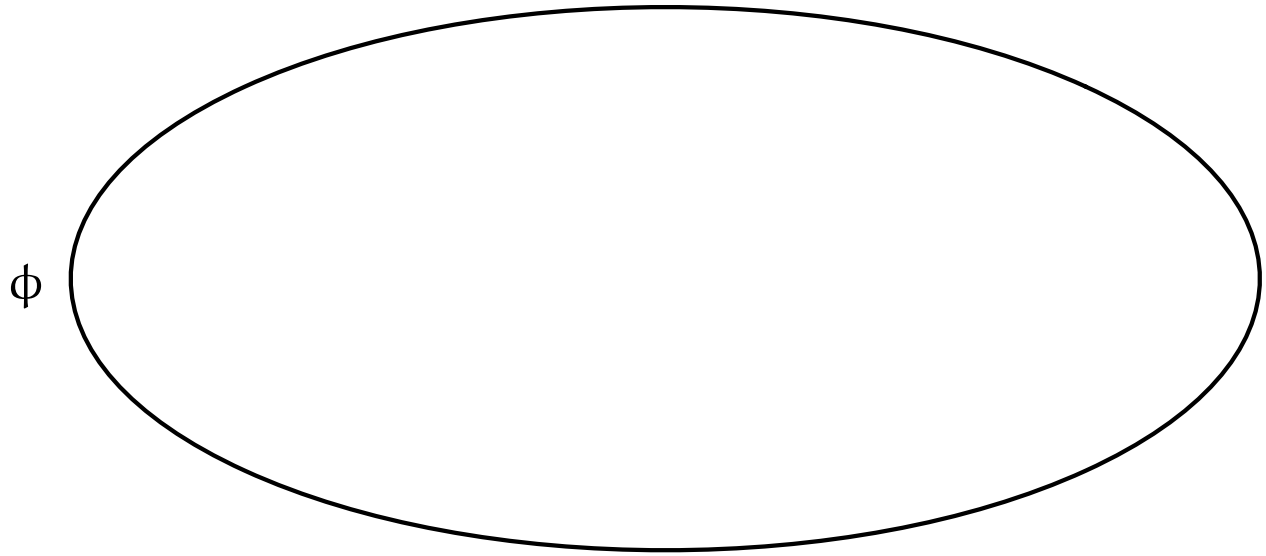
Classification of **D**erivation

- *Soundness-preserving* derivation seen in instance derivation
- *Completeness-preserving* derivation will be illustrated by a prototype for *feature diagrams*

Classification of **D**erivation

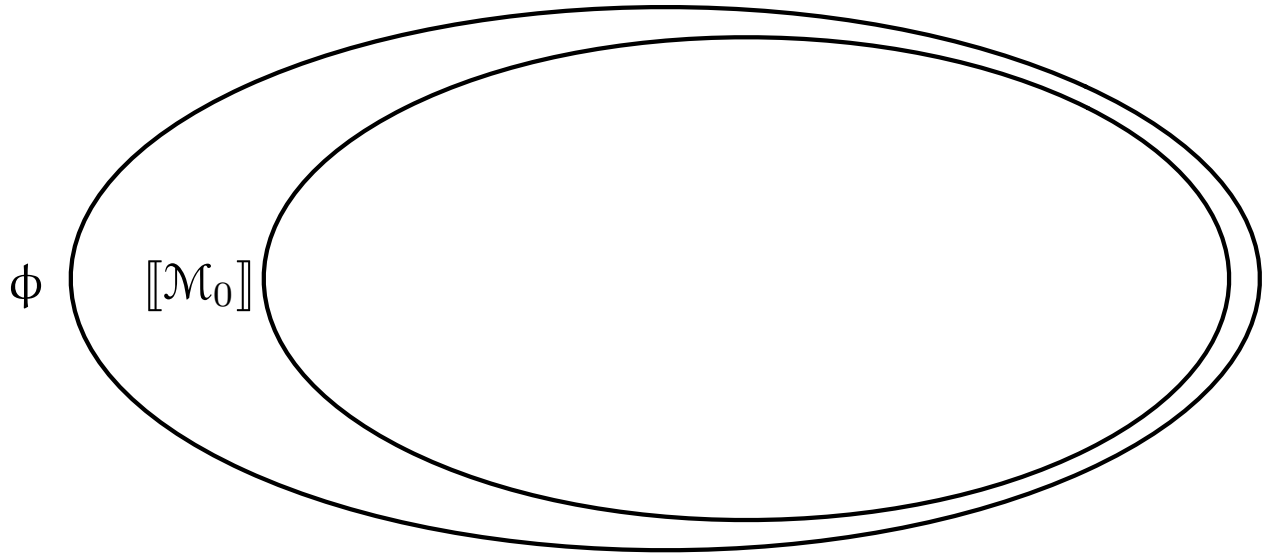
- *Soundness-preserving* derivation seen in instance derivation
- *Completeness-preserving* derivation will be illustrated by a prototype for *feature diagrams*
- *Semantics-preserving* derivation will be illustrated by a prototype for *feature models*

Soundness-**P**reserving **D**erivation



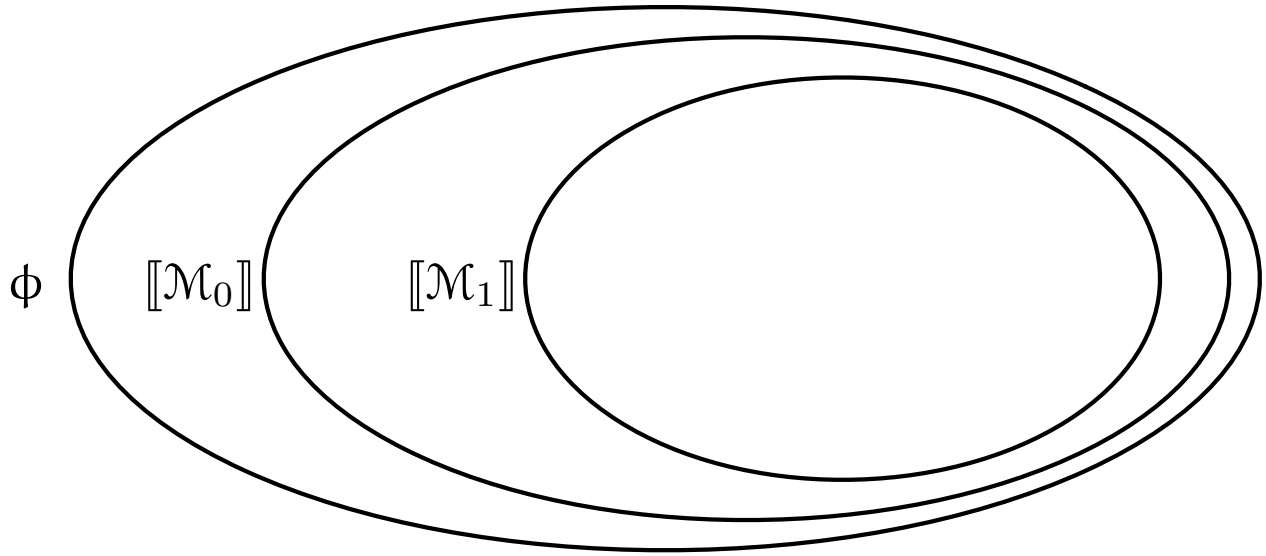
model $\longrightarrow$ an instance

Soundness-**P**reserving **D**erivation



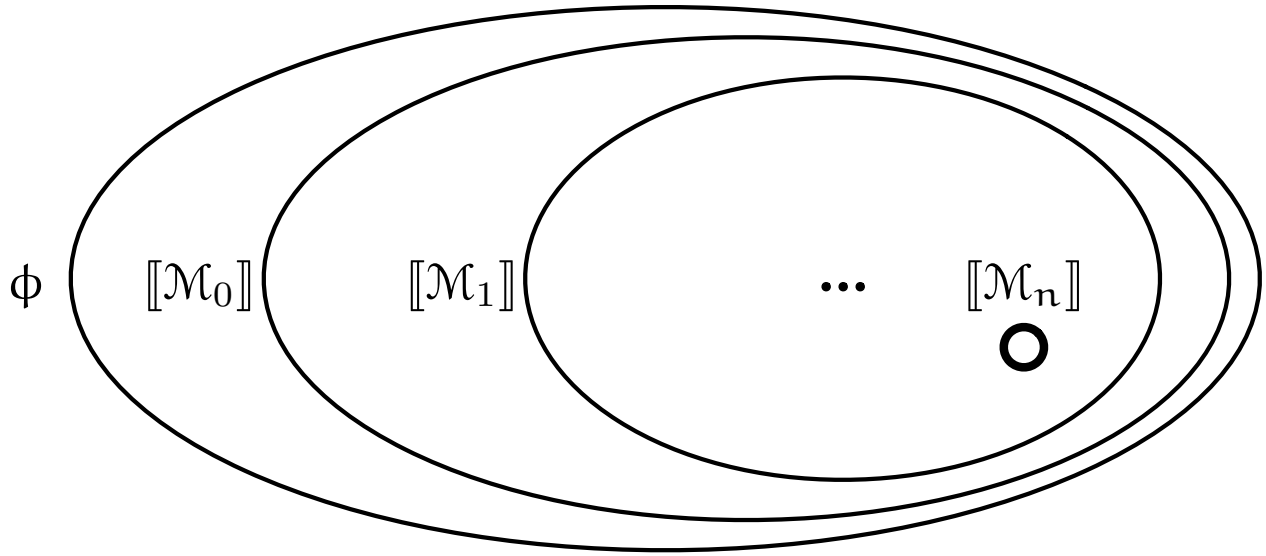
model $\longrightarrow$ an instance

Soundness-**P**reserving **D**erivation



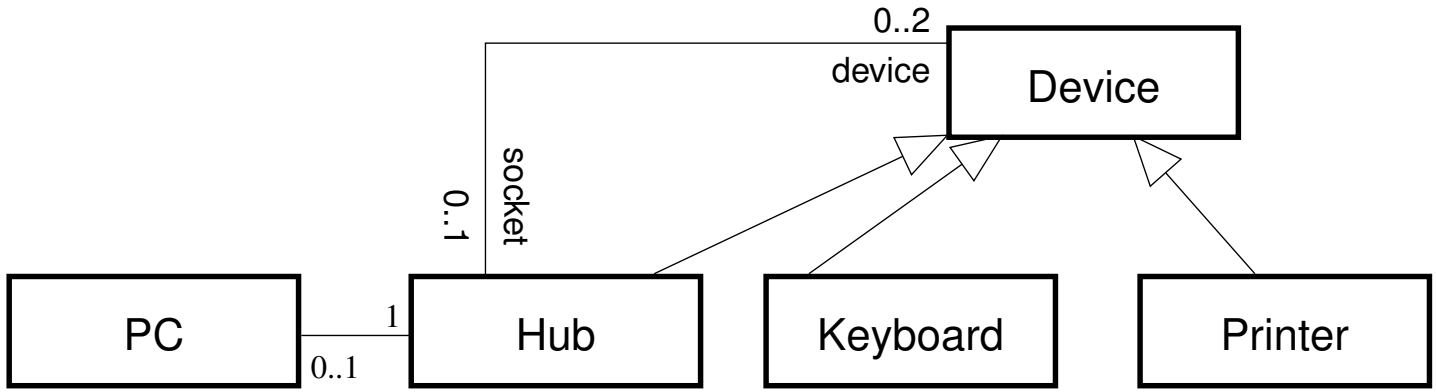
model $\longrightarrow$ an instance

Soundness-**P**reserving **D**erivation



model $\longrightarrow$ an instance

USB Language



C1 Each USB must contain exactly one instance of PC.

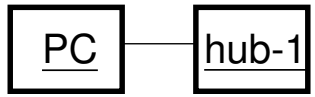
C2 Every device is connected to a port or to the PC instance.

C3 Every USB has a keyboard connected or a free port to connect one.

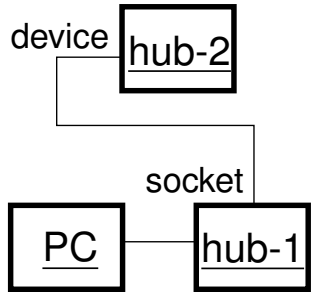
Deriving a **USB**

1.

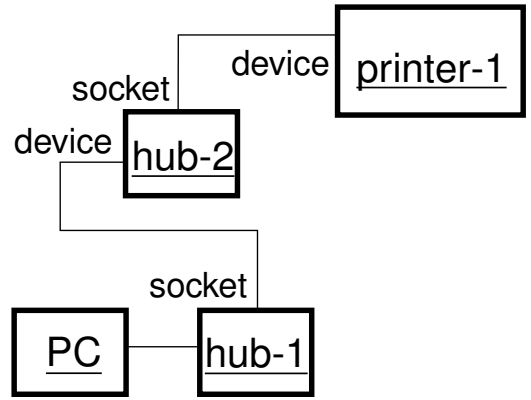
2.



3.



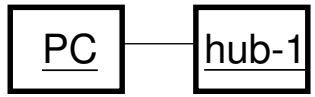
4.



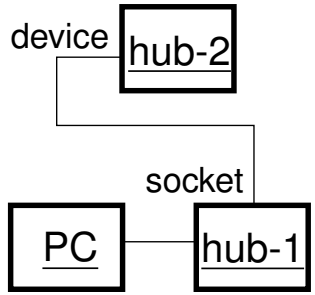
Deriving a **USB**

1.

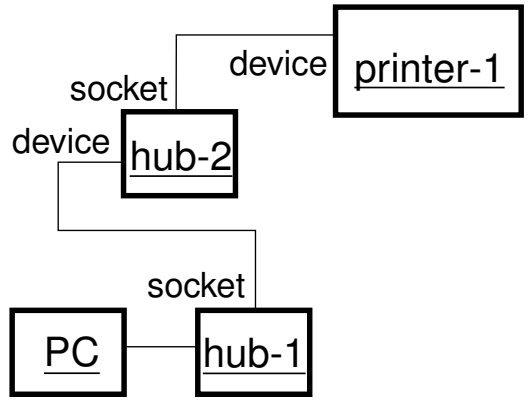
2.



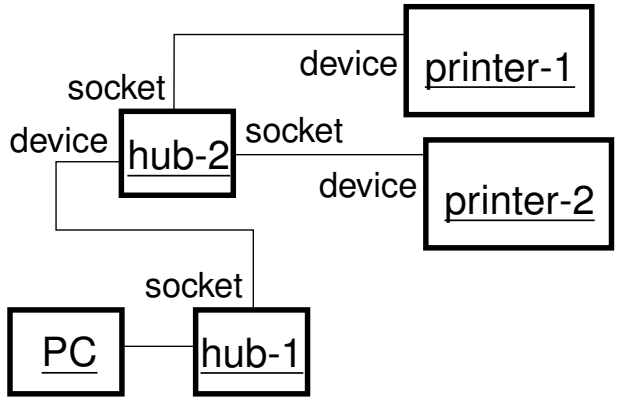
3.



4.



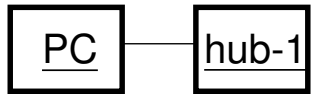
5.



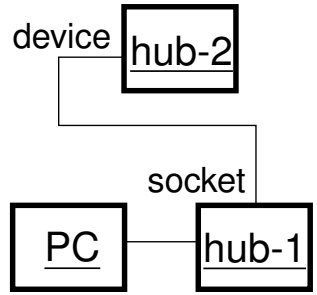
Deriving a **USB**

1.

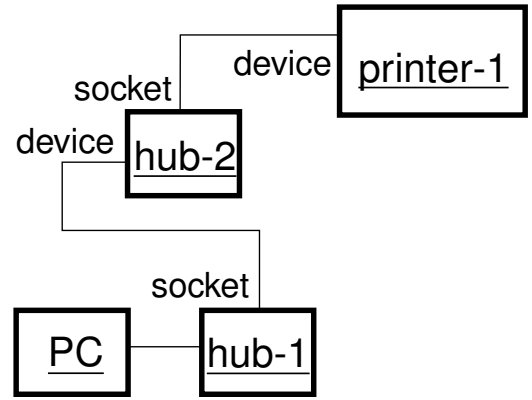
2.



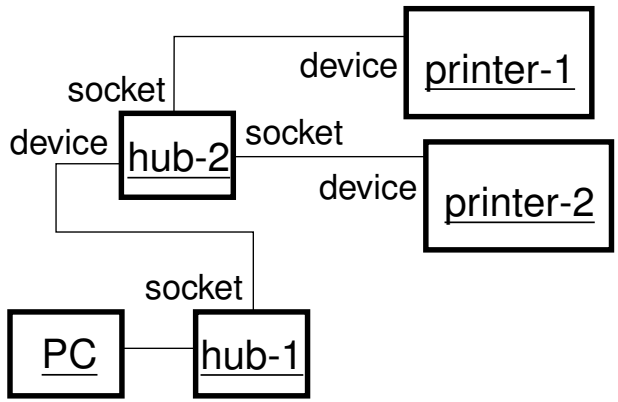
3.

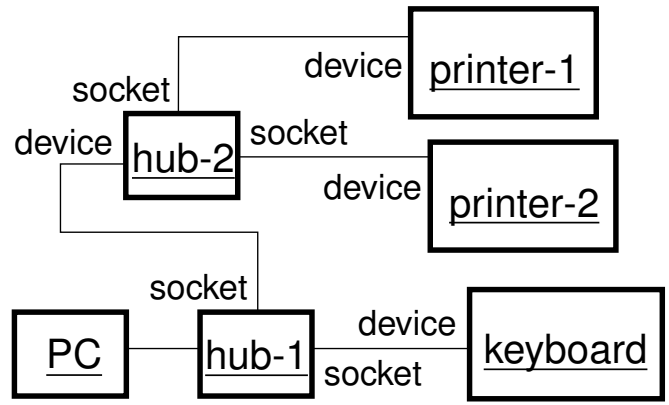


4.



5.





Properties of **D**erivation

- **Validity of advice:** no sequence of operations leads to an invalid model

Properties of **D**erivation

- **Validity of advice:** no sequence of operations leads to an invalid model
  - For the USB language: all derivable USBs satisfy the language constraints (the diagram and C1–C3)

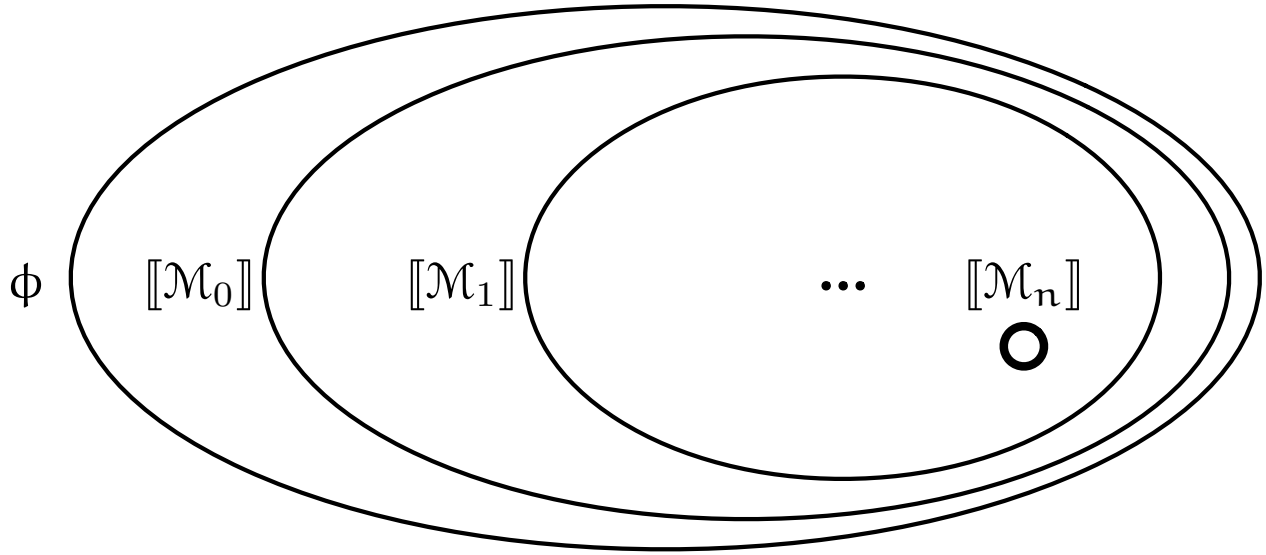
Properties of **D**erivation

- **Validity of advice:** no sequence of operations leads to an invalid model
  - For the USB language: all derivable USBs satisfy the language constraints (the diagram and C1–C3)
- **Exhaustiveness of advice:** all conforming instances are derivable.

Properties of **D**erivation

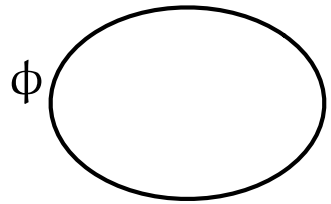
- **Validity of advice:** no sequence of operations leads to an invalid model
  - For the USB language: all derivable USBs satisfy the language constraints (the diagram and C1–C3)
- **Exhaustiveness of advice:** all conforming instances are derivable.
  - For the USB language: all legal USBs can be derived (*van der Meer* 2006)

Soundness-**P**reserving **D**erivation

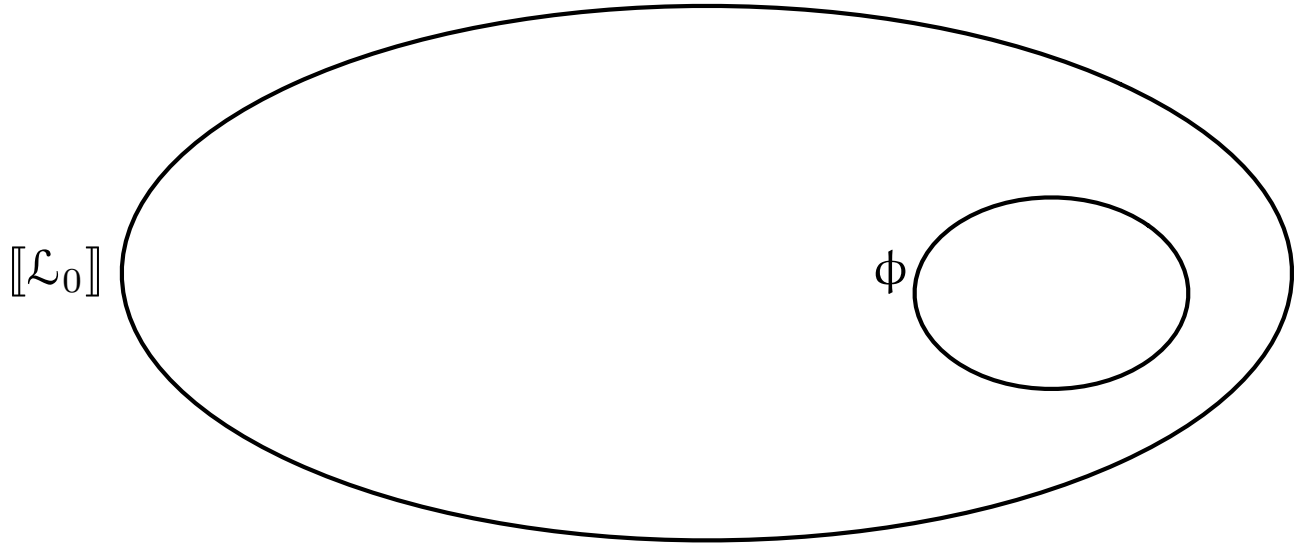


model $\longrightarrow$ an instance

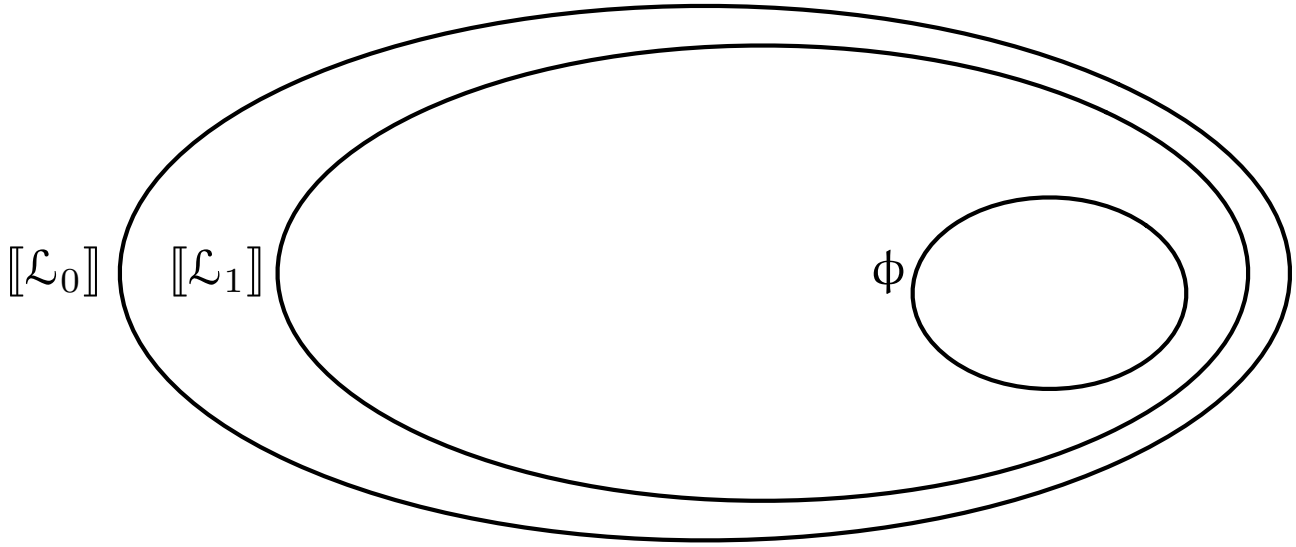
Completness-**P**reserving **D**erivation



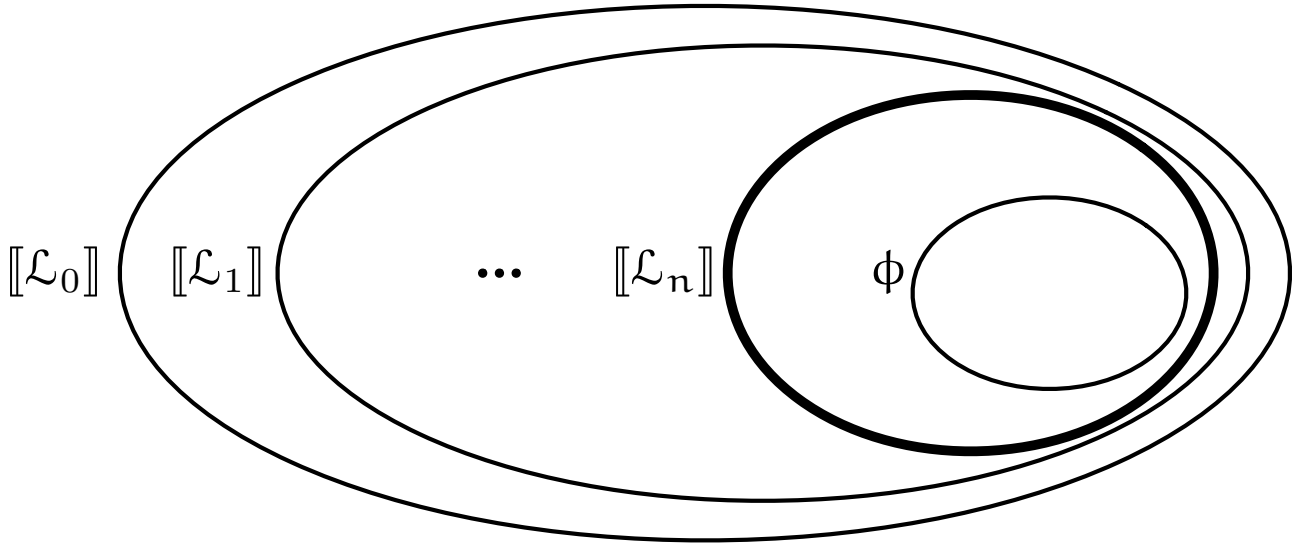
Completness-**P**reserving **D**erivation



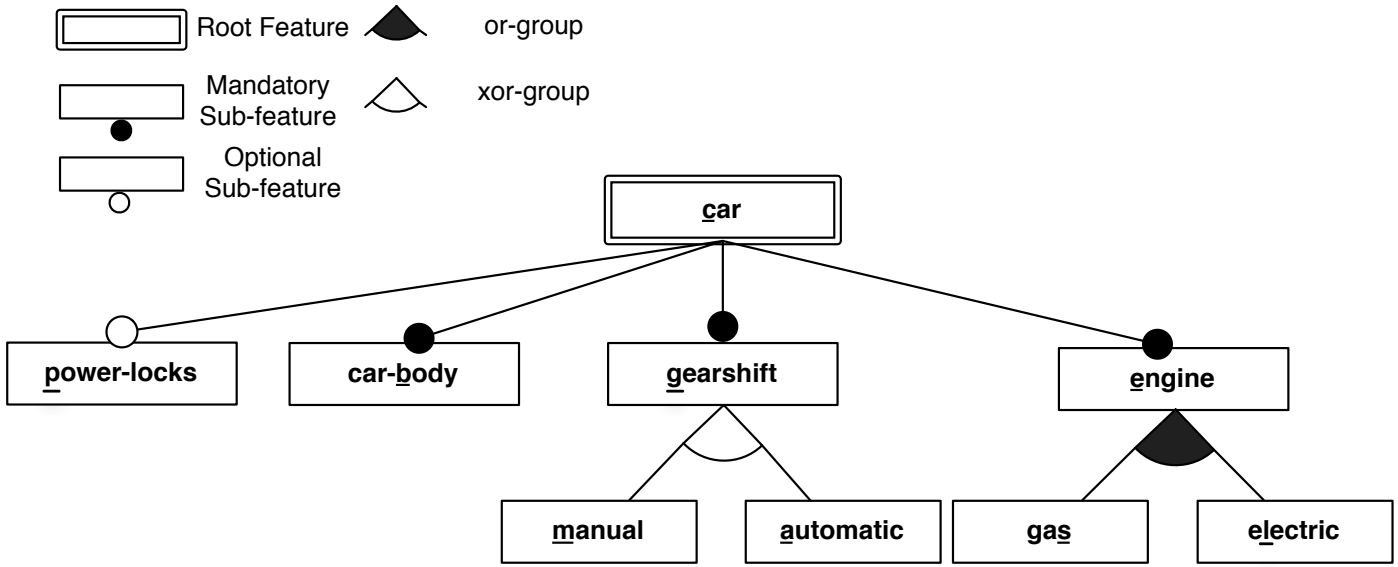
Completness-**P**reserving **D**erivation



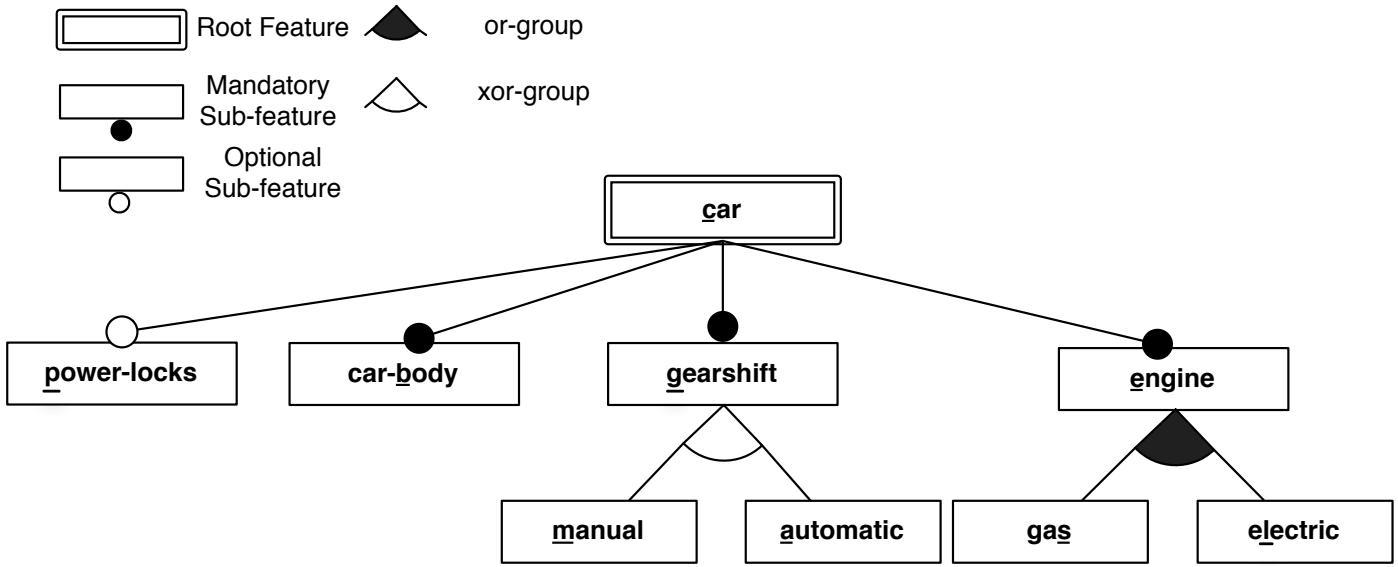
Completness-**P**reserving **D**erivation



Feature Model Example



Feature Model Example

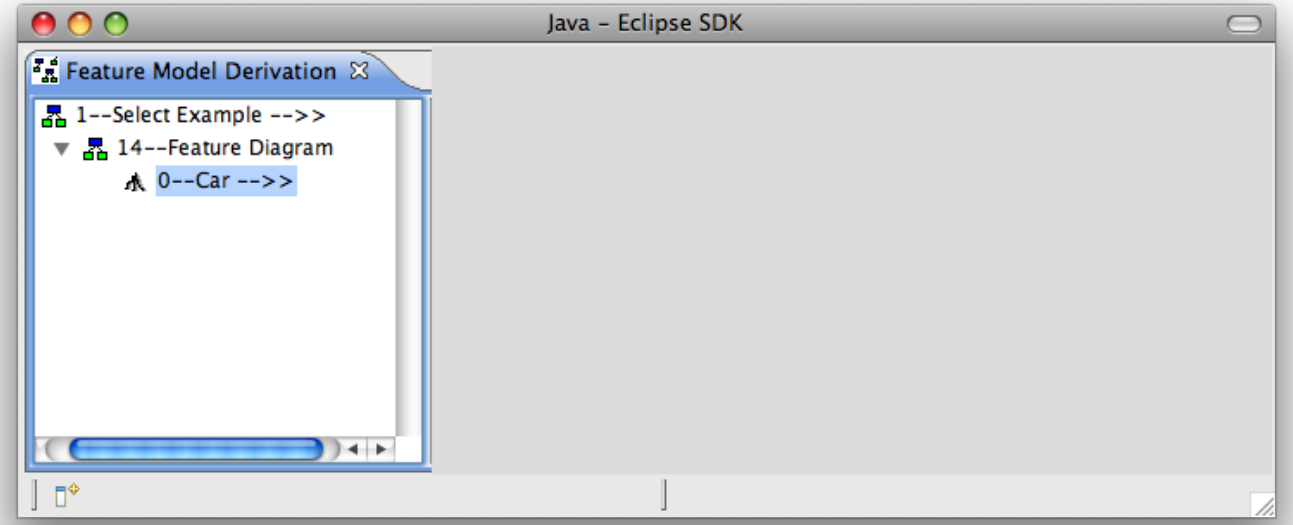


additional constraint:

$$\text{electric} \rightarrow \text{automatic}$$

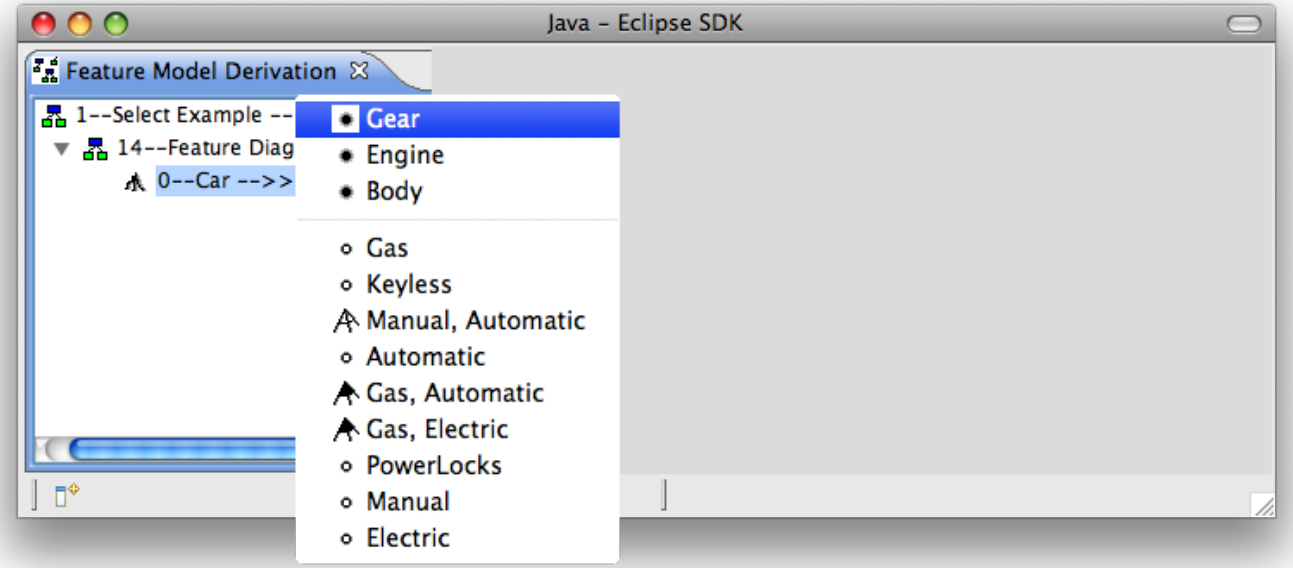
Practically **S**peaking

Feature Model approximating a constraint



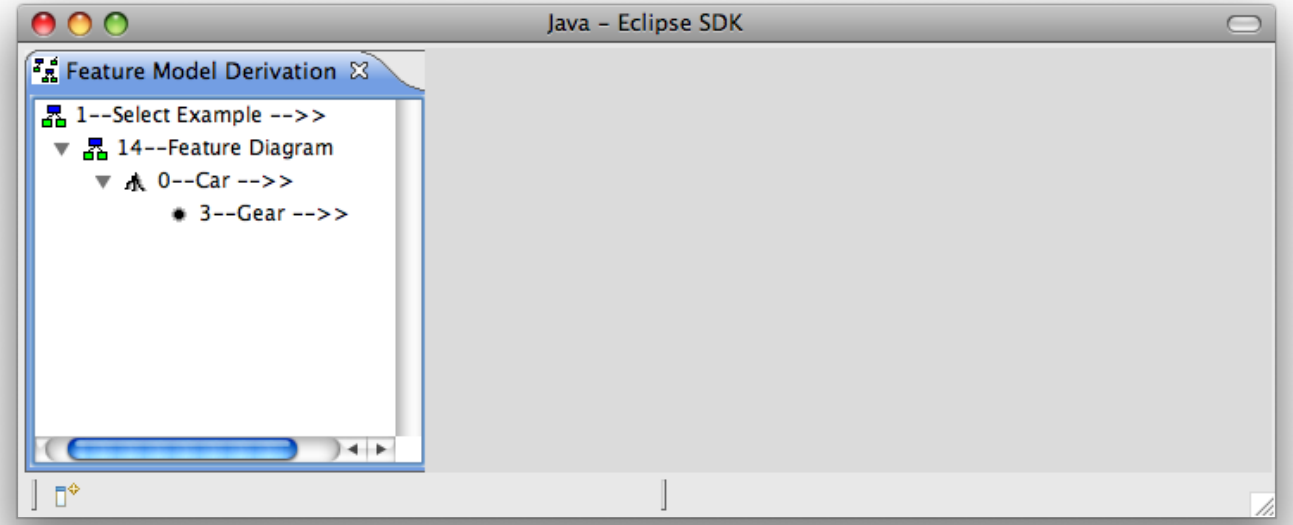
Practically **S**peaking

Feature Model approximating a constraint



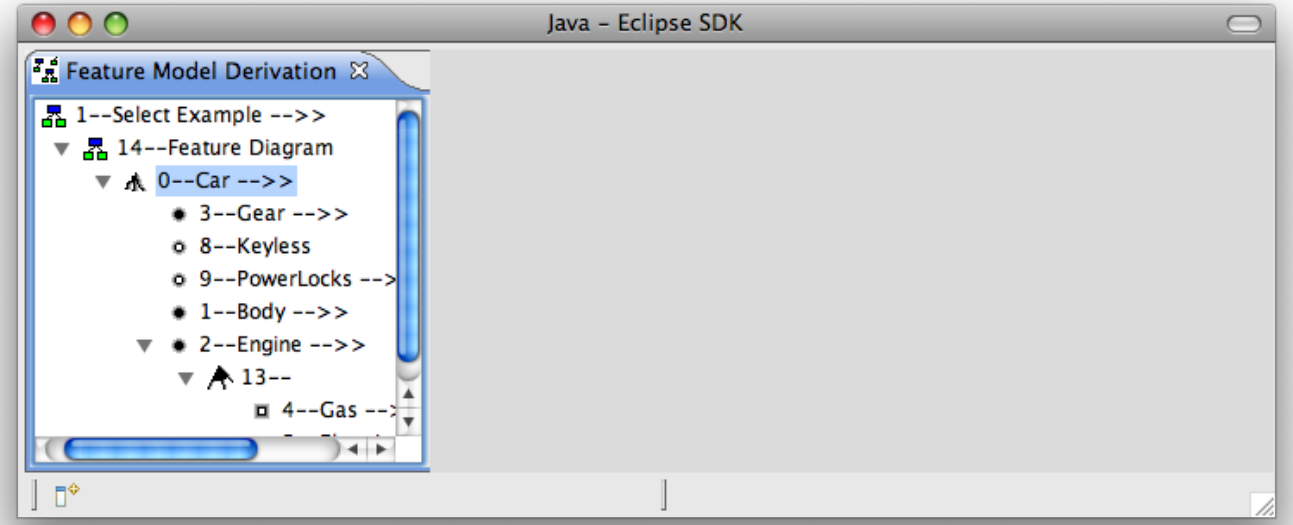
Practically **S**peaking

Feature Model approximating a constraint



Practically **S**peaking

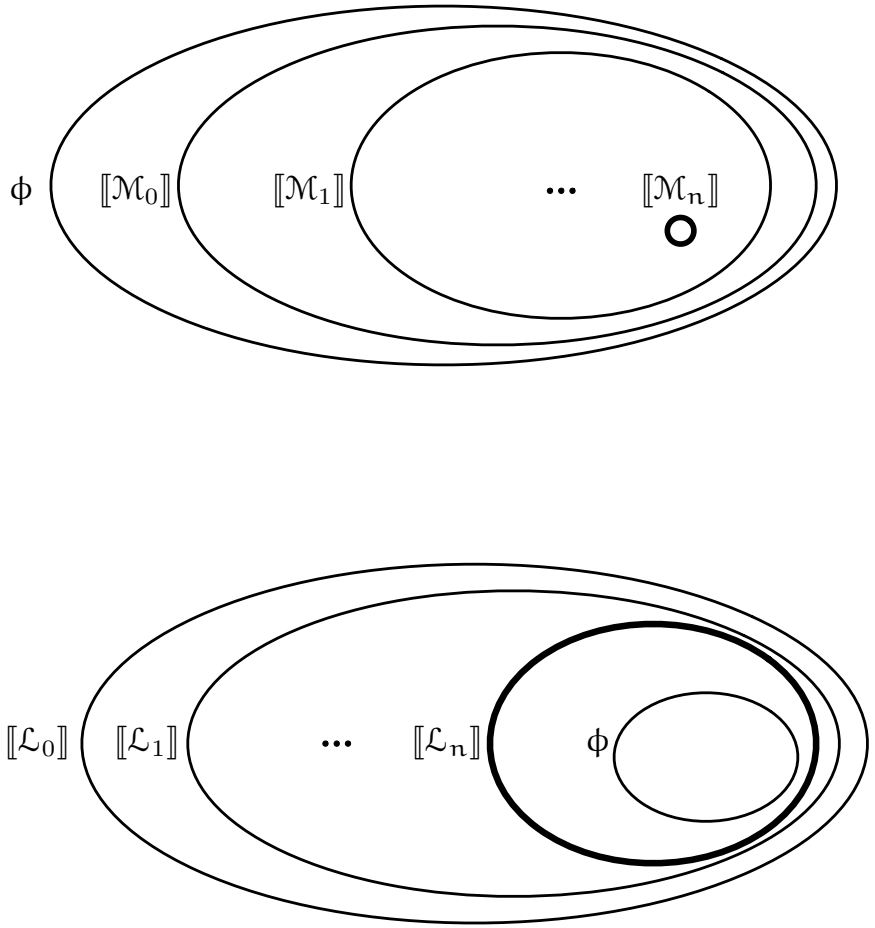
Feature Model approximating a constraint



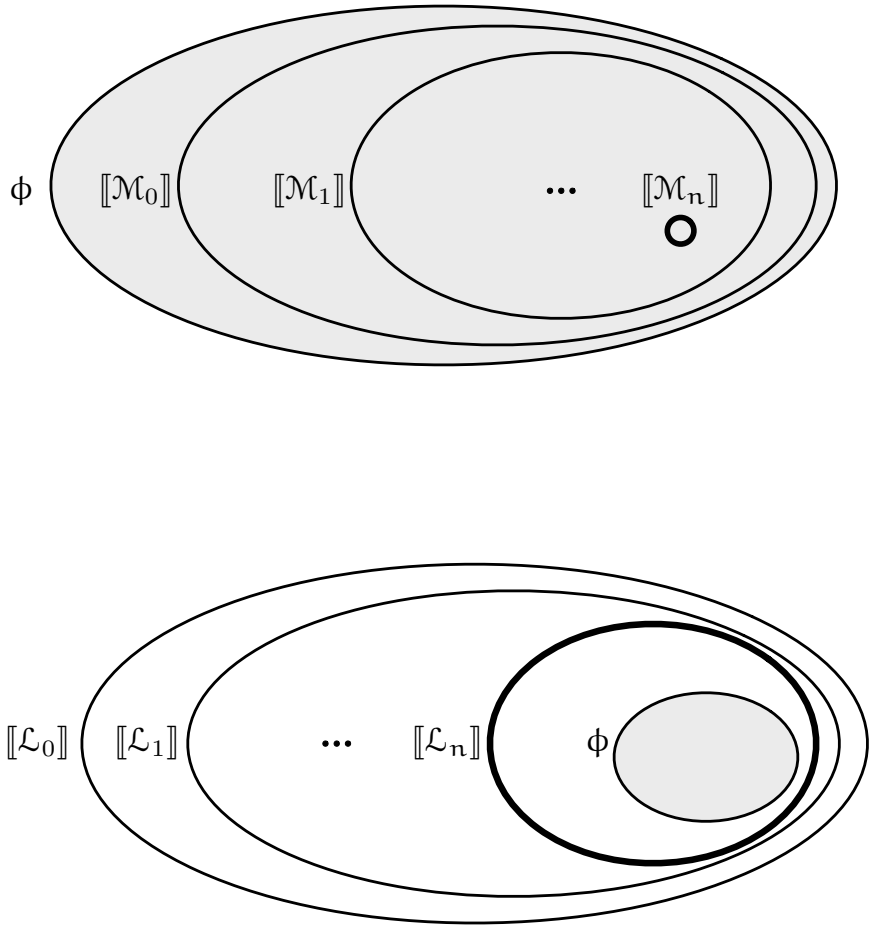
Claims

- The algorithm has the properties of **validity** and **exhaustiveness** of advice.
- The algorithm is efficient compared to approaches based on Constraint Satisfaction or Logic Programming.

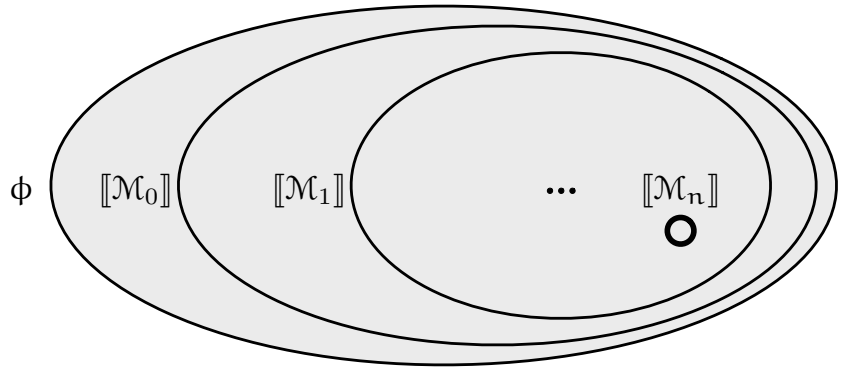
Compl**e**teness-**P**reserving **D**erivation



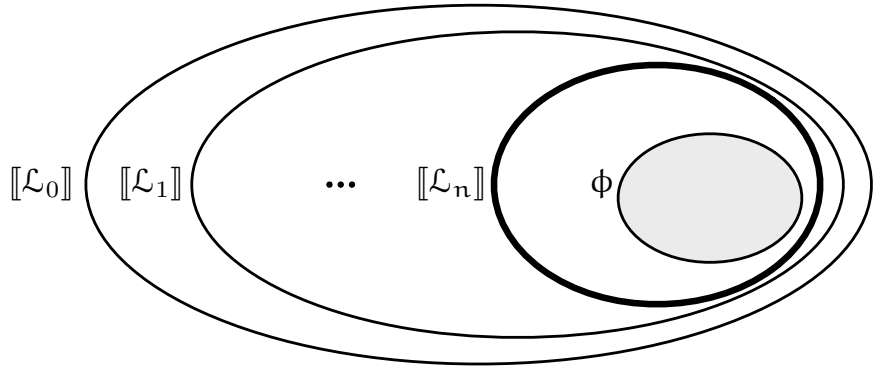
Compl**e**teness-**P**reserving **D**erivation



Completeness-Preserving Derivation



model $\longrightarrow$ an instance



meta-model $\longrightarrow$ model

Semantics-**P**reserving **D**erivation

semantics-preserving =
completeness-preserving
+ soundness-preserving

Semantics-**P**reserving **D**erivation

semantics-preserving =
completeness-preserving
+ soundness-preserving

Example: any refactoring

Semantics-**P**reserving **D**erivation

semantics-preserving =
completeness-preserving
+ soundness-preserving

Example: any refactoring
Example: feature **model** derivation

Full **S**etup

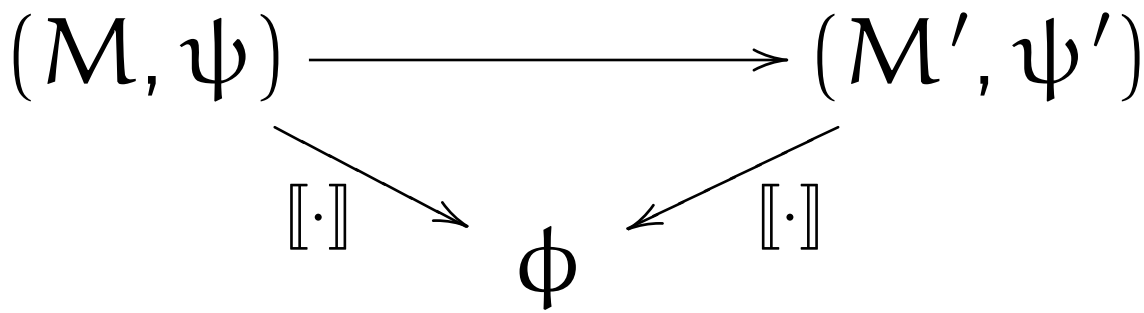
- the model comprises two components
  - feature diagram (M)
  - additional constraint (ψ)
- overall semantics must be preserved
  - “ $\mathrm{M} + \psi = \mathrm{M}' + \psi'$”

Full **S**etup

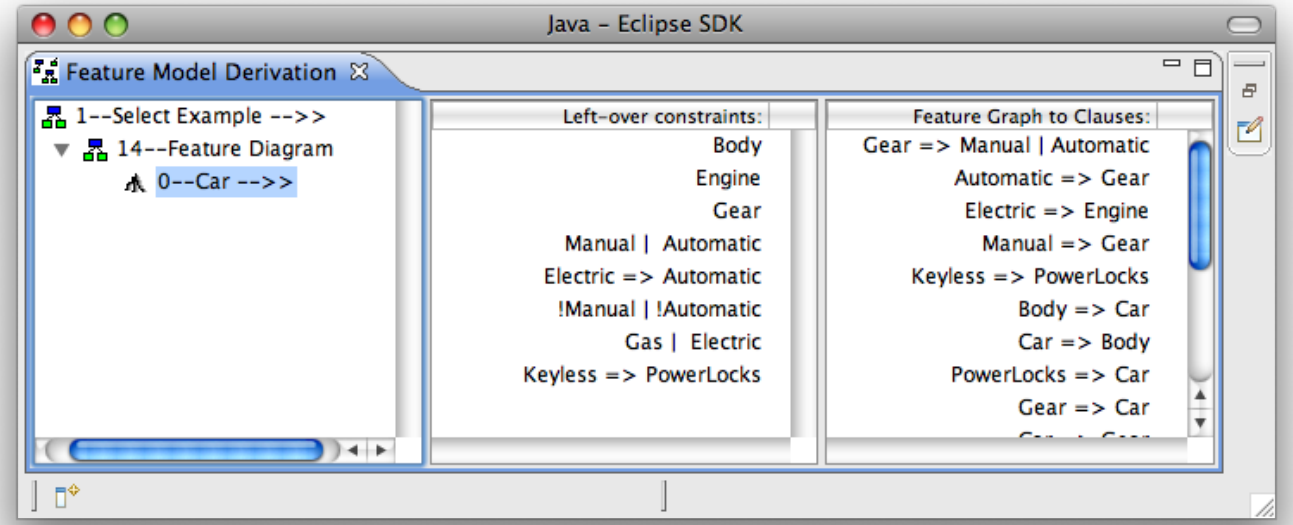
- the model comprises two components
  - feature diagram (M)
  - additional constraint (ψ)
- overall semantics must be preserved

  " $\mathrm{M} + \psi = \mathrm{M}' + \psi'$"

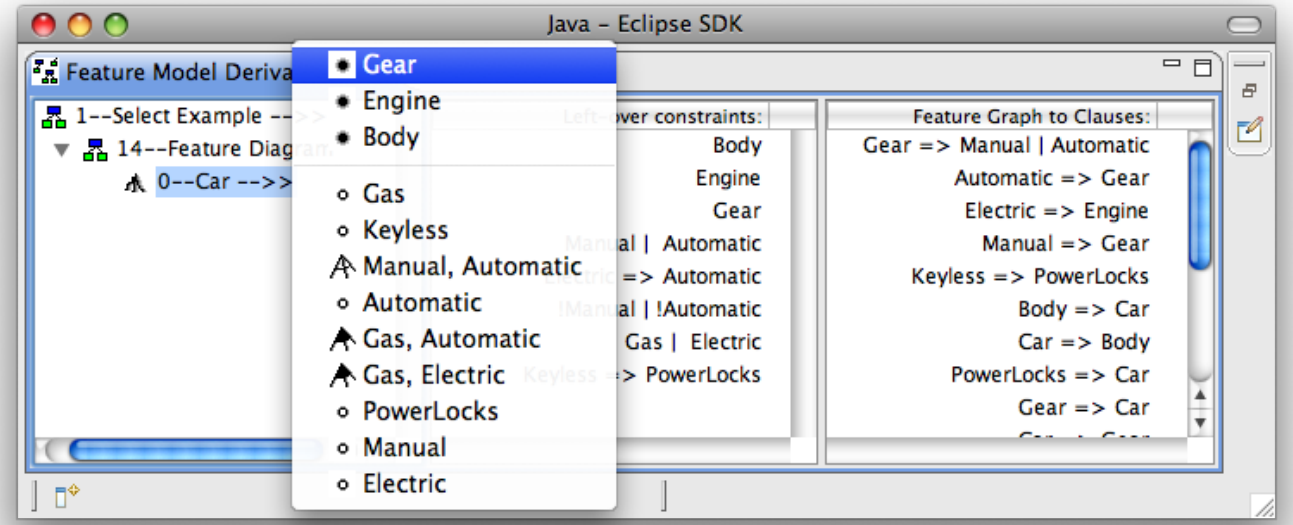
or



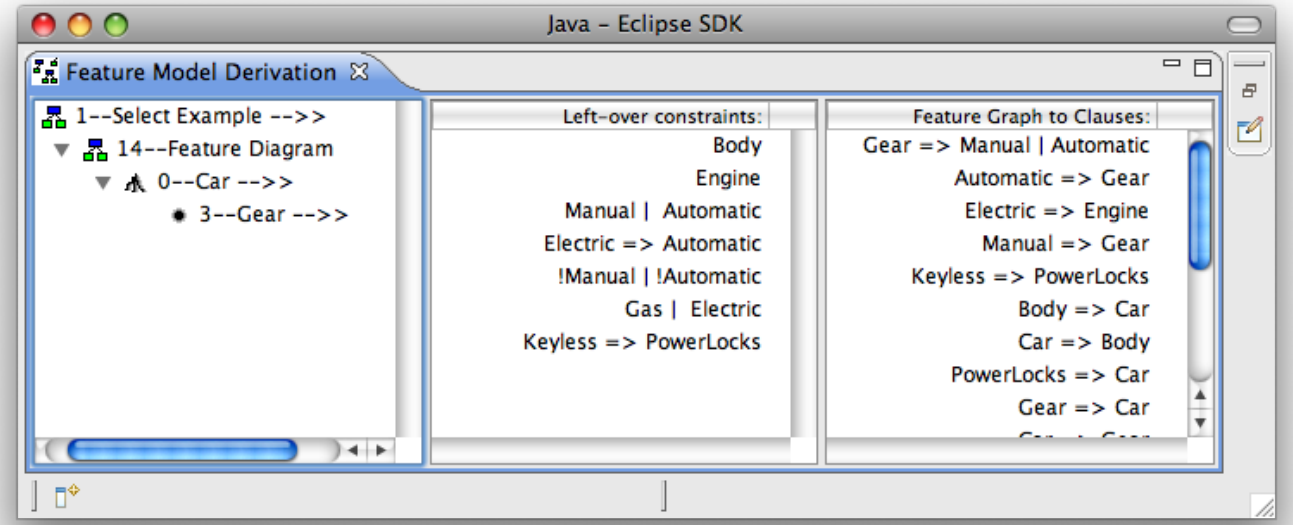
Practically **S**peaking



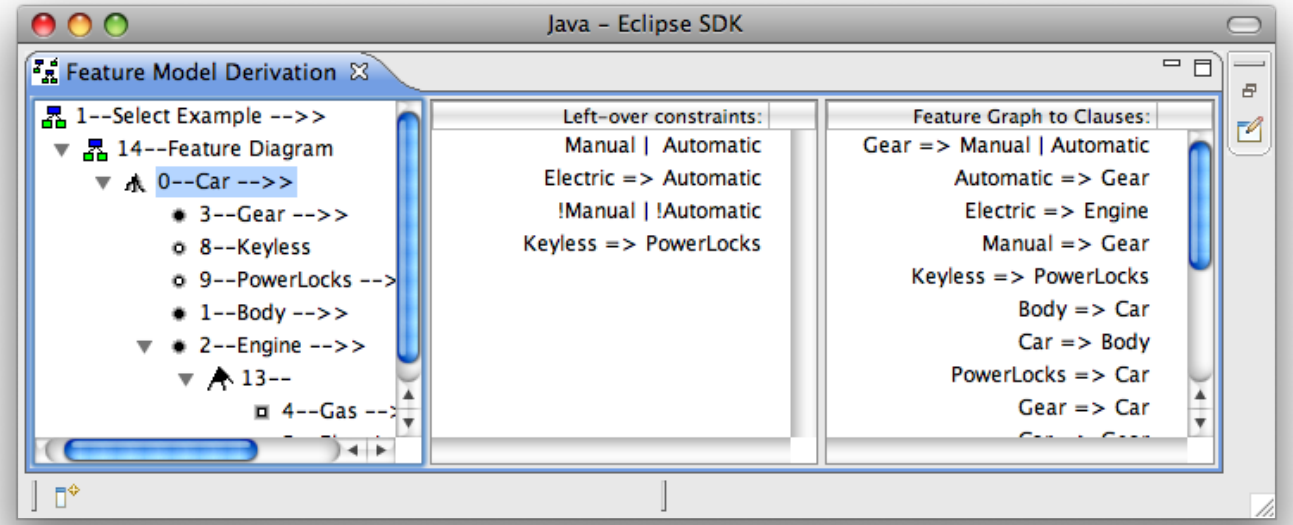
Practically Speaking



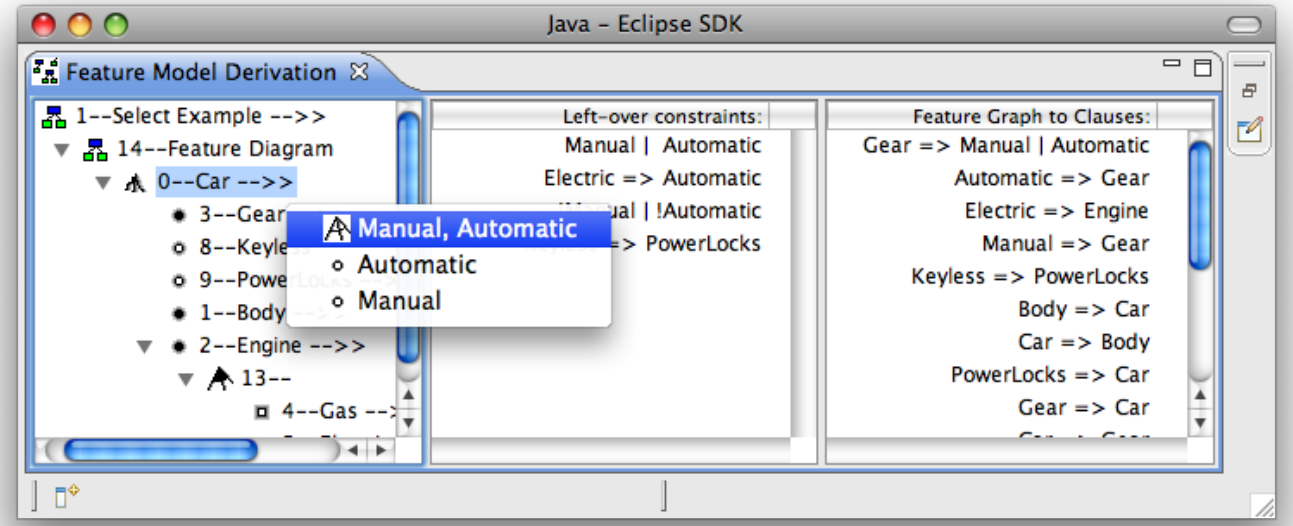
Practically **S**peaking



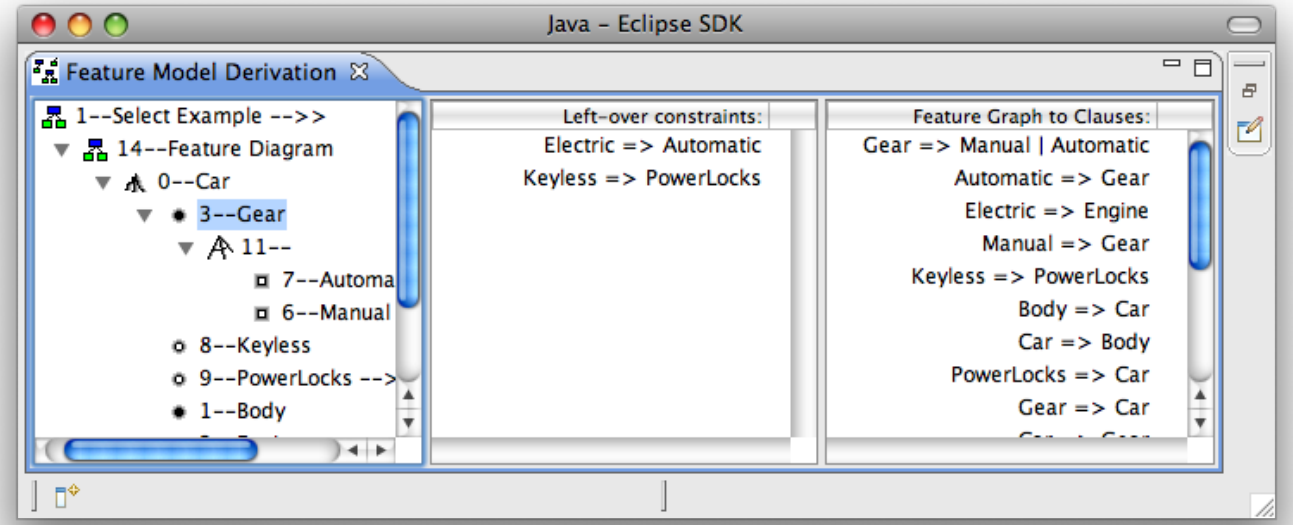
Practically Speaking



Practically Speaking



Practically Speaking



Summary and **C**hallenges **A**head

- **glossary**

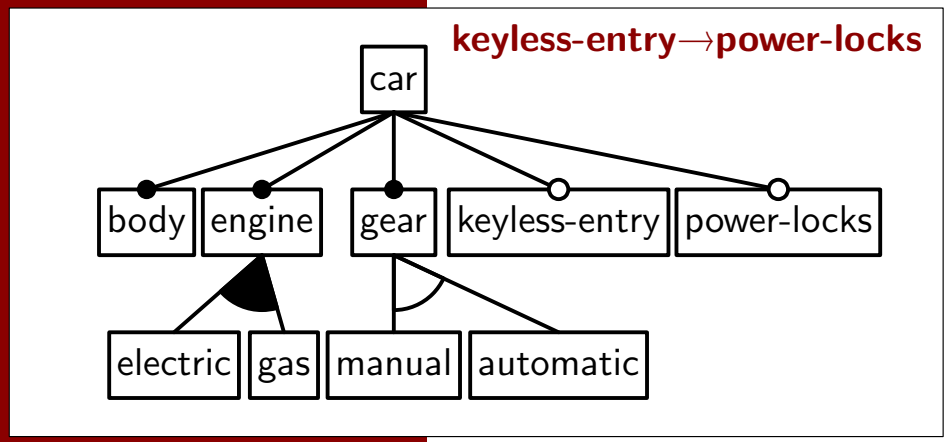
  soundness-preserving derivation
  completeness-preserving derivation
  semantics-preserving derivation

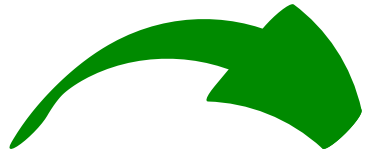
  validity of advice
  exhaustiveness of advice

- Work this out for a rich subset of ECORE models, not only for Feature Models

Q&A

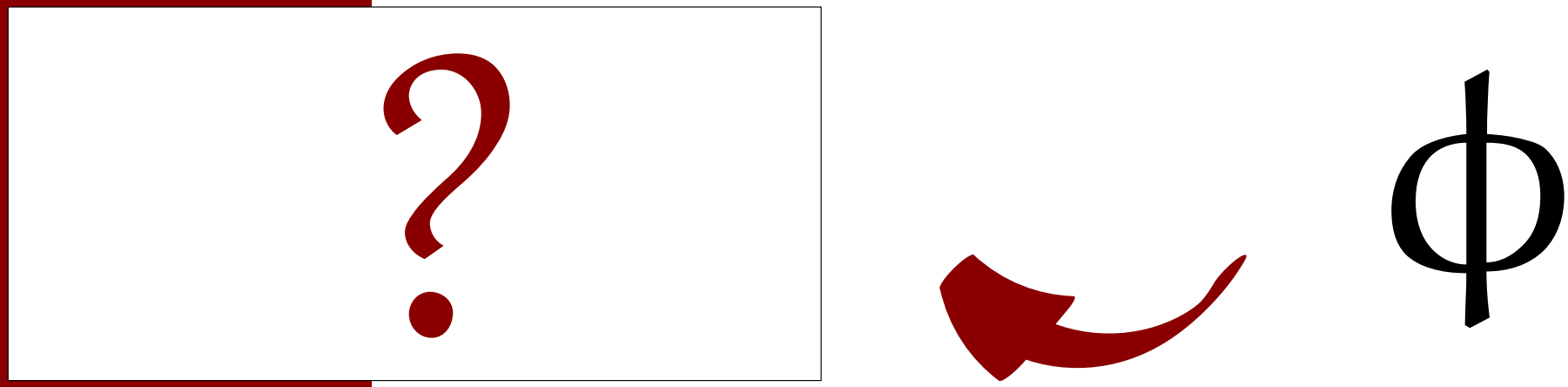
Semantics



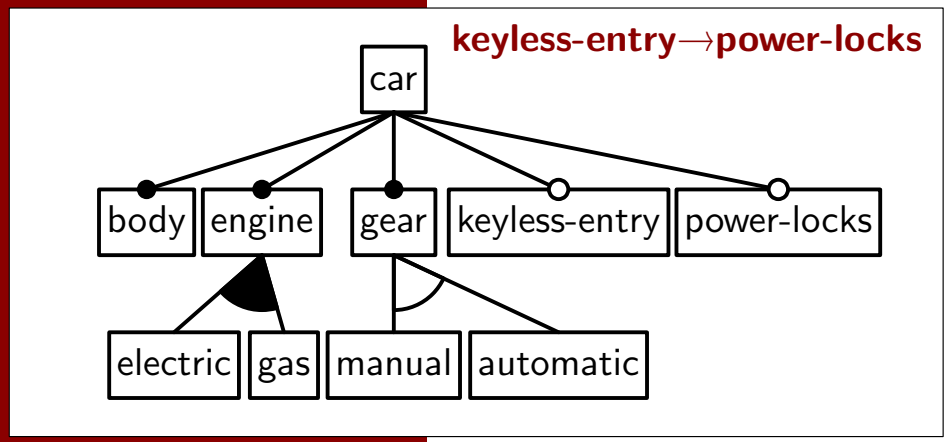


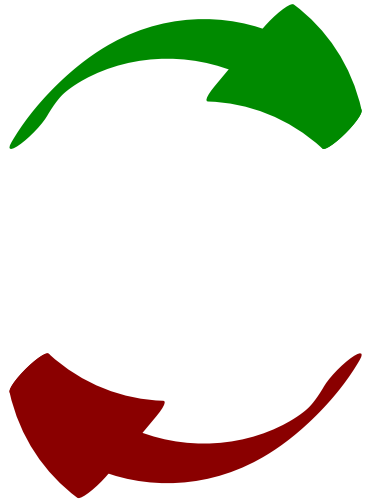
ϕ

Reverse **E**ngineering **S**yntax



Reverse **E**ngineering **S**yntax





ϕ

Computing Valid-Operations for FMs

VALID-OPERATIONS(M, n, ϕ) : operation-set

$\triangleright$ M is a (partially constructed) feature diagram
$\triangleright$ n is a node in M, present iff M is nonempty

1 **if** M is empty
2 **then return** $\{\text{Root}(r) \mid \text{for each feature } c.\ \phi, M \models c \rightarrow r\}$
3 $solitary \leftarrow \{\text{Mandatory}(n, c) \mid \phi, M \models c \leftrightarrow n$ and c not instantiated in $M\}$
$\cup \{\text{Optional}(n, c) \mid \phi, M \models c \rightarrow n$ and c not instantiated in $M\}$
4 $groups \leftarrow \{\text{OrGroup}(n, m_1 \ldots m_k) \mid n \leftrightarrow \phi, M \models \bigvee_{i \in 1 \ldots k} m_i$, $k > 1$ and all m_i are not instantiated in $M\}$
$\cup \{\text{XorGroup}(n, m_1 \ldots m_k) \mid \phi, M \models n \leftrightarrow \bigvee_{i \in 1 \ldots k} m_i$ and $\bigwedge_{i \neq j} \neg(m_i \wedge m_j)$, $k > 1$ and all m_i are not instantiated in $M\}$
5 $refine \leftarrow \{\text{RefineOR-group}(n, m_1, \ldots, m_k) \mid \{m_1, \ldots, m_k\}$ is an or-group of p in M and $\phi, M \models \bigwedge_{i \neq j} \neg(m_i \wedge m_j)\}$
$\cup \{\text{RefineOptional}(n) \mid n$ is an optional child of p in M, and $\phi, M \models n \rightarrow p\}$
6 **return** $solitary \cup groups \cup refine$