Interactive Model Derivation with External Constraints

Mikoláš Janota
University College Dublin, Ireland

Victoria Kuzina
Andrzej Wąsowski
IT University of Copenhagen, Denmark
Interactive Model Derivation

Possible Operations

Analysis

Model

Perform

Edit Operation

Choice

Designer
Interactive Model Derivation

Possible Operations

Model

Analysis

Perform

Edit Operation

Choice

Designer
Interactive Model Derivation

Possible Operations

Analysis

Model

Perform

Edit Operation

Choice

Designer
Interactive Model Derivation

- Model
  - Analysis
  - Possible Operations

- Perform
- Edit Operation
- Choice
- Designer
Model Carving
Model Carving
Model Carving
Model Carving
Classification of Derivation

- Soundness-preserving derivation seen in instance derivation
Classification of Derivation

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

- *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-Preserving Derivation

\[ \phi \]

model $\rightarrow$ an instance
Soundness-Preserving Derivation

\[ \phi \llbracket M_0 \rrbracket \]

model $\rightarrow$ an instance
Soundness-Preserving Derivation

\[ \phi \quad [\mathcal{M}_0] \subset [\mathcal{M}_1] \]

model → an instance
Soundness-Preserving Derivation

model → an instance
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. PC

2. PC → hub-1

3. PC → hub-1
   device → hub-2
   socket → hub-2

4. PC → hub-1
   device → printer-1
   socket → hub-2
Deriving a USB

1. PC
2. PC → hub-1
3. device → hub-2
4. socket → hub-2
5. printer-1

PC → hub-1
socket → device
printer-2
socket → device
socket → device
PC → hub-1
Deriving a USB

1. PC
2. PC → hub-1
3. PC → hub-1
4. PC → hub-1
5. PC → hub-1

- device
- socket
- device
- socket
- device
- socket
- device
- socket
- device
- socket
- device
- socket
- device
- keyboard
Properties of Derivation

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

- **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 Derivation

- **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 Derivation

- **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-Preserving Derivation

\[ \phi \rightarrow [M_0] \rightarrow [M_1] \rightarrow \ldots \rightarrow [M_n] \]

model → an instance
Completeness-Preserving Derivation
Completeness-Preserving Derivation

\[ \mathcal{L}_0 \]

\[ \phi \]
Completeness-Preserving Derivation

$[\mathcal{L}_0] \quad [\mathcal{L}_1] \quad \phi$
Completeness-Preserving Derivation
Feature Model Example

- **Root Feature**: car
- **Mandatory Sub-feature**: power-locks, car-body, gearshift, engine
- **Optional Sub-feature**: manual, automatic, gas, electric

**Additional Constraint**: electric → automatic

- **or-group**: manual, automatic
- **xor-group**: gas, electric
additional constraint:

electric → automatic
Practically Speaking

Feature Model approximating a constraint
Practically Speaking

Feature Model approximating a constraint
Practically Speaking

Feature Model approximating a constraint
Practically Speaking

Feature Model approximating a constraint
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.
Completeness-Preserving Derivation
Completeness-Preserving Derivation
Completeness-Preserving Derivation

model → an instance

meta-model → model
Semantics-Preserving Derivation

semantics-preserving =
completeness-preserving
+ soundness-preserving
Semantics-Preserving Derivation

semantics-preserving = completeness-preserving + soundness-preserving

Example: any refactoring
Semantics-Preserving Derivation

semantics-preserving = completeness-preserving + soundness-preserving

Example: any refactoring
Example: feature model derivation
the model comprises two components
feature diagram ($\mathcal{M}$)
additional constraint ($\psi$)
overall semantics must be preserved
“$\mathcal{M} + \psi = \mathcal{M}' + \psi'$”
the model comprises two components

feature diagram \((M)\)

additional constraint \((\psi)\)

overall semantics must be preserved

\[
\text{“} M + \psi = M' + \psi' \text{“}
\]

or

\[
(M, \psi) \xrightarrow{\phi} (M', \psi')
\]
Practically Speaking
Practically Speaking
Practically Speaking
Practically Speaking
Practically Speaking
Practically Speaking
Summary and Challenges Ahead

- **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
Semantics

- car
- body
- engine
- gear
- keyless-entry
- power-locks
- electric
- gas
- manual
- automatic

keyless-entry → power-locks

φ
Reverse Engineering Syntax
Reverse Engineering Syntax

[Diagram showing a network of car parts with arrows indicating 'keyless-entry' and 'power-locks']

\[\phi\]
Computing Valid-Operations for FMs

\textbf{VALID-OPERATIONS}(M, n, \phi) : operation-set

\begin{itemize}
  \item \( M \) is a (partially constructed) feature diagram
  \item \( n \) is a node in \( M \), present iff \( M \) is nonempty
\end{itemize}

\begin{enumerate}
  \item \textbf{if} \( M \) \textbf{is empty}
  \item \textbf{then return} \( \{\text{Root}(r) \mid \text{for each feature } c. \ \phi, M \models c \rightarrow r\} \)
  \item \textbf{solitary} \leftarrow \{\text{Mandatory}(n, c) \mid \phi, M \models c \leftrightarrow n \}
      \quad \text{and } c \text{ not instantiated in } M \}
      \cup \{\text{Optional}(n, c) \mid \phi, M \models c \rightarrow n \}
      \quad \text{and } c \text{ not instantiated in } M \}
  \item \textbf{groups} \leftarrow \{\text{OrGroup}(n, m_1 \ldots m_k) \mid n \leftrightarrow \phi, M \models \bigvee_{i \in 1 \ldots k} m_i, \}
      \quad k > 1 \text{ and all } m_i \text{ 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 \}
      \quad \text{and } \bigwedge_{i \neq j} (m_i \land m_j), \}
      \quad k > 1 \text{ and all } m_i \text{ are not instantiated in } M \}
  \item \textbf{refine} \leftarrow \{\text{RefineOR-group}(n, m_1, \ldots, m_k) \mid \}
      \quad \{m_1, \ldots, m_k\} \text{ is an or-group of } p \text{ in } M \}
      \text{ and } \phi, M \models \bigwedge_{i \neq j} (m_i \land m_j) \}
      \cup \{\text{RefineOptional}(n) \mid n \text{ is an optional child of } p \text{ in } M, \}
      \quad \text{ and } \phi, M \models n \rightarrow p \}
  \item \textbf{return} \text{solitary} \cup \text{groups} \cup \text{refine}
\end{enumerate}