An Algebra of Product Families

Peter Höfner

joint work with
R. Khedri, B. Möller

August, 2009

Introduction

Product lines and product families

- originally from hardware industry
- studying the commonality/variability
- allow several variants of products
- reduction of development and maintenance costs
- adoption to software development [Parnas76]
- found its way into the software development process
  Lucent Technologies: decrease in development time and costs 60% to 70% [WeissLai99]

Introduction

View reconciliation

captering all requirements is not possible in one model

- hardware *and* software
- components
- middleware

Introduction

Problems

- different notions
- no precise definition

Solutions

- an algebraic foundation
- sets of integration constraints
- link features in different views

Product Family Algebra

Terminology

- feature: elementary basic unit
- product: composition of elementary features
- product family: collection of products

Product Family Algebra

Example

small company with family of three product lines:

Product line	Mandatory	Optional	Commonalities
MP3 Player	– Play MP3 files (p_mp3)	– Record MP3 files (r_mp3)	– Audio equaliser (a_eq) – Video algorithms (v_alg) – Dolby surround (dbs)
DVD Player	– Play DVD	– Play music CD – View pictures from picture CD – Burn CD – Handle additional DVDs	
Hard Disk Recorder		– MP3 player – organise MP3 files	

Product Family Algebra

Definition

a product family algebra $(S, +, 0, \cdot, 1)$ is an idempotent and commutative semiring
its elements are called product families

S	$\leftrightarrow$	abstract product families
$a + b$	$\leftrightarrow$	union/choice of product families a, b
0	$\leftrightarrow$	empty product family
$a \cdot b$	$\leftrightarrow$	all possible combinations of a-products with b-products
1	$\leftrightarrow$	family containing only the empty product with no features

Product Family Algebra

Required axioms

- $(S, +, 0)$ commutative and idempotent monoid
- $(S, \cdot, 1)$ monoid
- $\cdot$ distributes over $+$
- 0 is an annihilator, i.e., $0 \cdot a = 0 = a \cdot 0$

Product Family Algebra

Example continued

MP3 players is described algebraically as

$$\texttt{mp3\_player} \;=\; \texttt{p\_mp3} \cdot (\texttt{r\_mp3} + 1) \cdot \texttt{a\_eq} \cdot \texttt{v\_alg} \cdot \texttt{dbs}$$

term $a + 1$ expresses optionality of a; abbreviated by $\texttt{opt}[a]$.
distributivity yields

$$\texttt{p\_mp3} \cdot \texttt{r\_mp3} \cdot \texttt{a\_eq} \cdot \texttt{v\_alg} \cdot \texttt{dbs} \quad + \quad \texttt{p\_mp3} \cdot \texttt{a\_eq} \cdot \texttt{v\_alg} \cdot \texttt{dbs}$$

$\Rightarrow$ mp3_player is a family consisting of exactly two products

Product Family Algebra

Concrete models

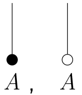
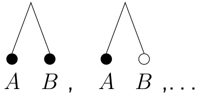
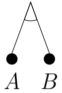
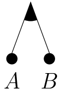
- bag model
  - product families: finite sets of finite bags (multisets) of basic features
  - $+$: set union
  - $\cdot$: bag union
  - products: singleton sets of finite bags
- set model
  - product families: finite sets of finite sets of basic features
  - forgets multiplicity of basic features in a product

Product Family Algebra

Definition
algebra feature-generated iff every element is a finite sum of finite products of features

representation in sum-of-products form corresponds to or/and trees of features (FODA)
can also be viewed as a commutative variant of the well known Backus-Naur form of grammars

Product Family Algebra

Base construct (feature diagram)	description	algebraic counterpart
 A , A	Mandatory and optional feature	A and $\text{opt}[A]$, resp.
 A B , A B ,. . .	Multiple Features	$A \cdot B$, $A \cdot \text{opt}[B]$
 A B	Alternative	$A + B$
 A B	Or-group	$A + B + A \cdot B$

Product Family Algebra

Principle of Family Induction

given a predicate $P(x)$ on feature-generated algebra S

- if P holds for 0 and all products (induction base)
- and is preserved by addition, i.e.,
  $P(b) \wedge P(c) \;\Rightarrow\; P(b+c)$ (induction step)
- then $\forall\, a \in S : P(a)$

soundness shown by straightforward induction on cardinality

Refinement

Example

Given the product family dvd_player

$$\texttt{dvd\_player} \quad = \quad \texttt{p\_dvd} \cdot \texttt{a\_eq} \cdot \texttt{v\_alg} \cdot \texttt{dbs} \cdot \texttt{opt}[\texttt{p\_mp3}]$$

an "older" product family of DVD players does not support dbs and v_alg

$$\texttt{old\_dvd\_player} \quad = \quad \texttt{p\_dvd} \cdot \texttt{opt}[\texttt{p\_mp3}] \cdot \texttt{a\_eq}$$

each product of dvd_player has at least the same features as a product of old_dvd_player

$$\texttt{dvd\_player} \quad \sqsubseteq \quad \texttt{old\_dvd\_player}$$

we call dvd_player a refinement of old_dvd_player

Refinement

Definition

inclusion ordering: $a \leq b \quad \Leftrightarrow_{df} \quad a + b = b$

refinement relation: $a \sqsubseteq b \quad \Leftrightarrow_{df} \quad \exists\, c : a \leq b \cdot c$

- for product p the relation $a \sqsubseteq p$ means that all products in a have p as a subproduct
- $\sqsubseteq$ is a preorder

Refinement

Theorem

some useful properties for arbitrary product families a, b *and product* p

(a) $a \leq b \Rightarrow a \sqsubseteq b$

(b) $a \cdot b \sqsubseteq b$

(c) $a \sqsubseteq a + b$

(d) $a \sqsubseteq b \Rightarrow a + c \sqsubseteq b + c$

(e) $a \sqsubseteq b \Rightarrow a \cdot c \sqsubseteq b \cdot c$

(f) $a \sqsubseteq 0 \Leftrightarrow a \leq 0$

(g) $0 \sqsubseteq a \sqsubseteq 1$

(h) $a + b \sqsubseteq c \Leftrightarrow a \sqsubseteq c \wedge b \sqsubseteq c$

(i) $p \sqsubseteq a + b \Leftrightarrow p \sqsubseteq a \vee p \sqsubseteq b$

A Haskell-Prototype

- checks the adequacy of our definitions
- implements the bag model
- features are simply encoded as strings;
  bags are represented as ordered lists
- normalise expressions into a sum-of-products-form
- bag model is isomorphic to natural number:
  (atomic) features correspond to a primes
  products correspond to natural numbers
  (allows efficient algorithms)

Requirements: Implications and Exclusions

a multi-view approach also needs integration constraints

- often they link presence of a feature in one view to that of another feature in the same or another view
- can link subproducts or subfamilies as well
- common informal formulations:

  "if a member of a product family has subproduct p_1
  it also must have subproduct p_2*"*

  "if a member of a product family has subproduct p_1
  it must not have subproduct p_2*"*

Requirements: Implications and Exclusions

Definition

the requirement relation is defined in family-induction style

$$
\begin{array}{rcl}
a \xrightarrow{0} b & \Leftrightarrow_{df} & \texttt{TRUE} \\
a \xrightarrow{p} b & \Leftrightarrow_{df} & (p \sqsubseteq a \Rightarrow p \sqsubseteq b) \\
a \xrightarrow{c+d} b & \Leftrightarrow_{df} & a \xrightarrow{c} b \wedge a \xrightarrow{d} b
\end{array}
$$

for elements a, b, c, d and product p of a feature-generated algebra

- informally, $a \xrightarrow{e} b$ means that if e has a then it also has b
  i.e., a implies b within e
- $\xrightarrow{e}$ is again a preorder
- $a \xrightarrow{e} b$ coincides with $a \xrightarrow{e} lcm(a, b)$
- in the bag model, $lcm(p, q)$ is the "smallest" bag refined by p and q

Requirements: Implications and Exclusions

Example

assume a vehicle built from the following features:
`speed_indicator`, `steering_wheel`, `wheel`, `axis`, `engine`,
`standard_transmission` and `automatic_transmission`

- $\mathtt{engine} \xrightarrow{\mathtt{car}} \mathtt{speed\_indicator}$:
  every motorised car has also a speed indicator
- $\mathtt{engine} \cdot \mathtt{wheel} \xrightarrow{\mathtt{car}} \mathtt{steering\_wheel}$:
  there is at least one steering wheel if the vehicle has at least one engine

Requirements: Implications and Exclusions

Example continued

- $(\texttt{steering\_wheel}) \cdot (\texttt{steering\_wheel}) \xrightarrow{\texttt{car}} 0$ only one steering wheel is allowed
- $\texttt{wheel}^{2n+1} \xrightarrow{\texttt{car}} \texttt{wheel}^{2n+2}$
  a car has to have an even number of wheels
- $\texttt{engine} \xrightarrow{\texttt{car}} \texttt{standard\_transmission} + \texttt{automatic\_transmission}$
  every motorised car has a standard transmission or an automatic one
- $1 \xrightarrow{\texttt{car}} \texttt{engine}$
  each car has (at least) one engine

Requirements: Implications and Exclusions

Theorem
some useful properties

(a) $b \xrightarrow{a} b + c$.

(b) $b \cdot c \xrightarrow{a} b$.

(c) $b \xrightarrow{a} c \Rightarrow b \xrightarrow{a} c + d$.

(d) $b \xrightarrow{a} d \Rightarrow b \cdot c \xrightarrow{a} d$.

(e) *If* p *is a product, then* $b \xrightarrow{p} c \Rightarrow b + d \xrightarrow{p} c + d$.

(f) $a \xrightarrow{e} b \wedge c \xrightarrow{e} d \Rightarrow a \cdot c \xrightarrow{e} b \wedge a \cdot c \xrightarrow{e} d$.

(g) $a + b \xrightarrow{e} c \Leftrightarrow a \xrightarrow{e} c \wedge b \xrightarrow{e} c$.

Multi-View Reconciliation

algebraically, this can be tackled as follows:

- take two product lines a and b and a set of implication clauses of the form $c \xrightarrow{a \cdot b} d$
- write a and b in sum-of-products form
- the term $a \cdot b$ denotes all possible combinations of products from a with products from b
- multiply out
- from the resulting sum remove all products violating the implication clauses
- this method is implemented in our prototype

Multi-View Reconciliation

Example

a company builds (simplified) computers

- basic computers have a hard disc and a screen
- a second screen can be added
- a printer and/or a scanner can be added
- it is possible to have more than one extension

abbreviations: $\mathtt{hd}$, $\mathtt{scr}$, $\mathtt{prn}$ and $\mathtt{scn}$

$$\mathtt{hw} = \mathtt{hd} \cdot \mathtt{scr} \cdot \mathtt{opt}[\mathtt{scn}, \mathtt{prn}, \mathtt{scr}]$$

Multi-View Reconciliation

Example continued

another company provides two different software packages

- drivers for hard disks, screens and printers
- drivers for hard disks, screens and scanners

$$\mathtt{sw} = \mathtt{hd\_drv} \cdot \mathtt{scr\_drv} \cdot \mathtt{prn\_drv} \ + \ \mathtt{hd\_drv} \cdot \mathtt{scr\_drv} \cdot \mathtt{scn\_drv}$$

Multi-View Reconciliation

Example continued

the multi-view reconciliation Problem asks for all products satisfying the following requirements

$$\begin{aligned} \texttt{hd} &\xrightarrow{\texttt{hw}\cdot\texttt{sw}} \texttt{hd\_drv} \\ \texttt{scr} &\xrightarrow{\texttt{hw}\cdot\texttt{sw}} \texttt{scr\_drv} \\ \texttt{prn} &\xrightarrow{\texttt{hw}\cdot\texttt{sw}} \texttt{prn\_drv} \\ \texttt{scn} &\xrightarrow{\texttt{hw}\cdot\texttt{sw}} \texttt{scn\_drv} \end{aligned}$$

each hardware component needs an appropriate driver

the above procedure determine all admissable products and elimates all inconsistent products

Multi-View Reconciliation

Example continued

some detailed observations about the result set:

- there is no machine with scanner and printer
  due to the fact that there is no software package with drivers for both components
- there are two different versions of the hardware product consisting of hard disk and screen(s) only;
  they offer software for scanners and printers, resp.
- such products can be seen as hardware with an upgrade option: the customer can add a hardware component without changing the software

Multi-View Reconciliation

- symmetrically to the combination of product lines, one can extract a *view* of a product family:
- simply project to the respective feature set F
- using a feature algebra homomorphism that sends all features outside F to the empty product 1

Conclusion

algebraic approach

- conflict resolution between views performed without modification on the initial views (*separation of concerns*)
- each view can be specified independently of the others
- very simple but effective mathematical background
- axiomatics purely first-order, hence automated reasoning (e.g., using Prover9) is possible and has been done
- prototypical implementation of some useful models of feature algebra in HASKELL

unmentioned work

- more examples and case studies
  e.g., product line of driver assisting systems with more than 40.000 products
- reduction of up to 75%

Outlook

- algebraic model of features is at a high level of abstraction
- from a software perspective, a feature could be a requirement scenario/use-case or a partial description of the functionality
- our current research aims at deriving concrete specifications of the members of a family from its abstract feature algebra specification and the concrete specifications of its basic features
- this step would join the ongoing research efforts for formal model driven software development techniques
- the feature algebra model of a family and the specifications of the family's features would be the initial models of the sought transformation into concrete form

