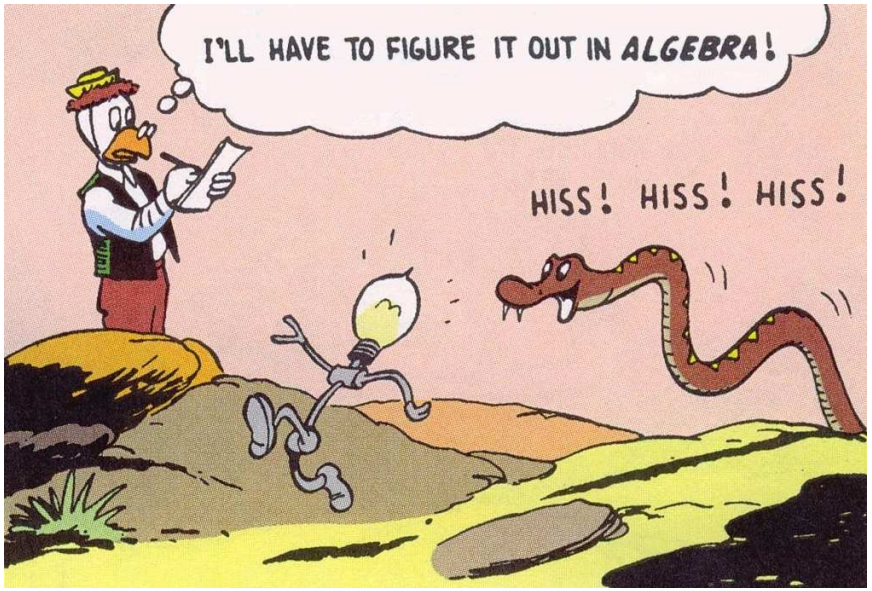
A Survey of Product Family Algebra

Peter Höfner, Ridha Khedri, Bernhard Möller



WG 2.1 Meeting Iceland 2011

1 Basic Definitions

terminology:

- *feature*: elementary basic unit
- *product*: composition of elementary features
- *product family* (or product line): collection of products

Definition

a *product family algebra* consists of

S	set of (abstract) product families
$a + b$	union of product families a, b – "or"
	associative, commutative, idempotent
0	empty product family, neutral w.r.t. $+$
$a \cdot b$	all possible combinations of a-products with b-products – "and"
	associative, commutative, distributive over $+$
1	family containing only the empty product with no features, neutral w.r.t. $\cdot$

commutative idempotent semiring

commutativity of $\cdot$: count only multiplicity of basic features, forget about their order

each family algebra term $\approx$ grammar of the family's products $\approx$ or-and tree

notions of *products* and *features* can be made precise in terms of $+/\cdot$ -irreducibility

Example

small company with family of two product lines:

- *DVD Players*
- *MP3 Players*

products share common features:

- audio equaliser (a_eq)
- dolby surround (dbs)

products may have mandatory and optional features

- all DVD players must have feature *play DVD* (p_dvd)
- some can *play mp3* (p_mp3)
- others cannot
- some MP3 players can *record mp3* (r_mp3)
- others cannot

algebraic description of optionality of a feature (or, more generally, a list of products)

$$\mathtt{opt}[p_1, \ldots, p_n] \stackrel{\text{def}}{=} (1 + p_1) \cdot \ \cdots \ \cdot (1 + p_n)$$

Example (continued)

above product lines:

$$\mathtt{dvd_player} = \mathtt{p_dvd} \cdot \mathtt{opt}[\mathtt{p_mp3}] \cdot \mathtt{a_eq} \cdot \mathtt{dbs}$$

$$\mathtt{mp3_player} = \mathtt{p_mp3} \cdot \mathtt{opt}[\mathtt{r_mp3}] \cdot \mathtt{a_eq} \cdot \mathtt{dbs}$$

whole product family:

$$\mathtt{dvd_player} + \mathtt{mp3_player} =$$

$$(\mathtt{p_dvd} \cdot \mathtt{opt}[\mathtt{p_mp3}] + \mathtt{p_mp3} \cdot \mathtt{opt}[\mathtt{r_mp3}]) \cdot \mathtt{a_eq} \cdot \mathtt{dbs}$$

commonality factored out using the distributive laws

approach allows also describing product families of product families:

since the basic features in a product family algebra can be anything, they might just as well be product families of the next lower level

algebra *feature-generated* if every element is a finite sum of finite products of features

commonality of a sum $p_1 + \cdots + p_n$ of products:

$$\gcd(p_1, \ldots, p_n)$$

particular feature-generated algebra (*bag model*):

- product families: finite sets of finite bags (multisets) of basic features
- $+$ is set union
- $\cdot$ is bag union

one concrete bag model:

- number the elementary features by prime numbers
- products: arbitrary natural numbers
- prime factor decomposition shows the multiplicity of each feature in a product
- product families: sets of products

coarser algebra (*set model*)

- product families: finite sets of finite sets of basic features
- forgets multiplicity of basic features in a product

both models can be enriched by counting or cost information
(or other relevant "measures")

2 Refinement

motivation for next definition:

Example (continued)

an "older" product line of DVD Players has no capability for dolby surround

$$\texttt{old_dvd_player} = \texttt{p_dvd} \cdot \texttt{opt}[\texttt{p_mp3}] \cdot \texttt{a_eq}$$

each product of `dvd_player` must have at least the same features as an `old_dvd_player`

hence we call `dvd_player` a *refinement* of `old_dvd_player`

for the formal definition we need the inclusion ordering

$$a \leq b \iff_{df} a + b = b$$

Definition

the refinement relation is

$$a \sqsubseteq b \iff_{df} \exists\, c : a \leq b \cdot c$$

informally, $a \sqsubseteq b$, i.e., a *refines* b, means that every product in a has (at least) all the features of some product in b, but possibly additional ones

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

$\sqsubseteq$ is a preorder, i.e., reflexive and transitive

further properties

$$a \leq b \implies a \sqsubseteq b$$

$$a \cdot b \sqsubseteq b$$

$$a \sqsubseteq a + b$$

$$a \sqsubseteq b \implies a + c \sqsubseteq b + c$$

$$a \sqsubseteq b \implies a \cdot c \sqsubseteq b \cdot c$$

$$a \sqsubseteq 0 \iff a \leq 0$$

$$0 \sqsubseteq a \sqsubseteq 1$$

3 Requirements: Implications and Exclusions

how to express integration constraints?

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"

expressed in feature algebra using the *requirement relation*

Definition

for elements a, b, c, d and product p in a feature-generated algebra, define inductively

$$a \xrightarrow{p} b \quad \Longleftrightarrow_{df} \quad (p \sqsubseteq a \implies p \sqsubseteq b)$$

$$\text{where } p \sqsubseteq c \iff_{df} \exists\, d : c \leq p \cdot d$$

$$a \xrightarrow{c+d} b \quad \Longleftrightarrow_{df} \quad a \xrightarrow{c} b \wedge a \xrightarrow{d} b$$

- 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 a preorder

assume a and b are products

- then $a \xrightarrow{e} b$ coincides with $a \xrightarrow{e} \mathrm{lcm}(a, b)$
- in the bag model, $\mathrm{lcm}(p, q)$ is the "smallest" bag refined by p and q

Example

assume features `wheel` and `axis`

- $\mathrm{lcm}(\mathtt{wheel}^4 \cdot \mathtt{axis}\,, \mathtt{wheel}^3 \cdot \mathtt{axis}^2) = \mathtt{wheel}^4 \cdot \mathtt{axis}^2$
- $\mathtt{wheel}^2 \xrightarrow{a} \mathtt{axis}$ expresses that in product line a two wheels need an axis
- compare this with the lcm for numbers

Example

assume that we have car components (features)

axis,engine,speed_indicator,steering_wheel,wheel

- engine $\xrightarrow{\text{car}}$ speed_indicator guarantees that every motor car also has a speed indicator
- wheel $\cdot$ wheel $\xrightarrow{\text{car}}$ steering_wheel and engine $\xrightarrow{\text{car}}$ steering_wheel require at least one steering wheel if the car has at least two wheels or one engine
- more than one steering wheel is excluded by (steering_wheel) $\cdot$ (steering_wheel) $\xrightarrow{\text{car}} 0$
- $1 \xrightarrow{\text{car}}$ engine requires that each car has (at least) one engine

further useful properties:

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

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

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

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

(e) if p is a product, then $b \xrightarrow{p} c \implies b + d \xrightarrow{p} c + d$

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

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

- global mutual exclusion of products p and q can be expressed by the additional axiom $p \cdot q = 0$
- practically, expressing exclusion using $\xrightarrow{a}$ is more suitable
- allows excluding a feature combination only within a

4 Multi-View Reconciliation

can algebraically 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

- conflict resolution between views is performed without modification of the initial views (*separation of concerns*)
- hence each view can be specified independently of the others
- we have implemented the algebra and a function for performing the view reconciliation in *Haskell*

5 Adding Semantics

so far, everything was more or less just "syntactic":

basic features were just names without a meaning

still we could

- formulate feature exclusion
- feature implication
- perform basic transformations like
- factoring out commonality
- multiplying out into sum-of products form
- hence count the number of products in a family
- and do algebraic view reconciliation

now we show one way (by far not the only one) how to give a "semantic" interpretation of the algebraic structure

basis: guarded commands

$$\text{if } B_1 \rightarrow C_1 \,[]\, \cdots \,[]\, B_n \rightarrow C_n \text{ fi}$$

or action systems

$$\text{do } B_1 \rightarrow C_1 \,[]\, \cdots \,[]\, B_n \rightarrow C_n \text{ od}$$

relationally based general correctness semantics

Definition

A *command* over a state space Σ is a pair (R, P) where $R \subseteq \Sigma \times \Sigma$ is a transition relation and P is a subset of Σ for which normal termination is guaranteed.

Example

$$\begin{aligned}
\mathsf{skip} &\stackrel{\text{def}}{=} (I, \Sigma) \\
\mathsf{abort} &\stackrel{\text{def}}{=} (\emptyset, \emptyset) \\
\mathsf{fail} &\stackrel{\text{def}}{=} (\emptyset, \Sigma) \\
\mathsf{chaos} &\stackrel{\text{def}}{=} (\Sigma \times \Sigma, \emptyset)
\end{aligned}$$

under a variant of the usual refinement relation the (equivalence classes of) commands form a lattice (they don't under demonic semantics)

join (arbitrary choice)

$$(R, P) \,[]\, (S, Q) \;=\; (R \cup S, P \cap Q)$$

meet (consistent choice)

$$(R, P) \sqcap (S, Q) \;=\; ((R \cap S) \cup (\overline{P}{\downarrow}S) \cup (\overline{Q}{\downarrow}R), P \cup Q)$$

where $\downarrow$ denotes restriction

in particular, both operations induce commutative monoids with fail and chaos as their neutral elements

Definition

A *(formal) scenario* over a state space Σ is a pair (C_s, C_e), where C_s and C_e are two domain-disjoint commands on Σ, called the *command of the system* and the *command of the environment*, respectively.

the environment will include the user of the system etc.

Example

$$
\begin{aligned}
& \texttt{openAccAtBranch}_s \\
\stackrel{\text{def}}{=} \; & \Big(\texttt{cstmerEligOpnAcc} \wedge \texttt{fldIdNum} = \texttt{idNum} \\
& \wedge \texttt{fldIdType} = \texttt{idDocType} \wedge \neg\texttt{newCstmer} \\
& \wedge \neg\texttt{acctCreated} \wedge \neg\texttt{prvlgesAccepted} \\
& \wedge \neg\texttt{feesPayed} \wedge \neg\texttt{crdIssued} \\
& \longrightarrow \texttt{fldCsrmerName} := \texttt{getCstmerName}(\texttt{idNum}) \; ; \\
& \quad \texttt{fldCstmerAddress} := \texttt{getCstmerAddress}(\texttt{idNum}) \Big) \\
\; [\!] \; & \Big(\texttt{cstmerEligOpnAcc} \wedge \texttt{fldIdNum} = \texttt{idNum} \\
& \wedge \texttt{fldIdType} = \texttt{idDocType} \wedge \neg\texttt{newCstmer} \\
& \wedge \texttt{fldCsrmerName} = \texttt{getCstmerName}(\texttt{idNum}) \\
& \wedge \texttt{fldCstmerAddress} = \texttt{getCstmerAddress}(\texttt{idNum}) \\
& \longrightarrow \texttt{acctprivileges} := \texttt{personalized} \; ; \\
& \quad \texttt{outputMssge} := \texttt{msgeAccptPrvlges?} \Big) \\
\; [\!] \; & \ldots
\end{aligned}
$$

Example

$$
\begin{aligned}
& \texttt{openAccAtBranch}_e \\
\stackrel{\text{def}}{=} \; & \Big(\texttt{cstmerEligOpnAcc} \wedge \neg(\texttt{fldIdNum} = \texttt{idNum}) \\
& \wedge \neg(\texttt{fldIdType} = \texttt{idDocType}) \wedge \neg\texttt{acctCreated} \\
& \wedge \neg\texttt{prvlgesAccepted} \wedge \neg\texttt{feesPayed} \wedge \neg\texttt{crdIssued} \\
& \longrightarrow \texttt{fldIdNum} := \texttt{idNum} \; ; \\
& \quad \texttt{fldIdType} := \texttt{idDocType} \Big) \\
& [\!] \; \Big(\texttt{cstmerEligOpnAcc} \wedge \texttt{fldIdNum} = \texttt{idNum} \\
& \wedge \texttt{fldIdType} = \texttt{idDocType} \wedge \texttt{newCstmer} \\
& \wedge \neg\texttt{acctCreated} \wedge \neg\texttt{prvlgesAccepted} \\
& \wedge \neg\texttt{feesPayed} \wedge \neg\texttt{crdIssued} \\
& \longrightarrow \texttt{fldCsrmerName} := \texttt{csrmerName} \; ; \\
& \quad \texttt{fldCstmerAddress} := \texttt{csrmerAddress} \Big) \\
& [\!] \; \ldots
\end{aligned}
$$

now we define a composition on scenarios

Definition

Let $S_C \stackrel{\text{def}}{=} (C_s, C_e)$ and $S_D \stackrel{\text{def}}{=} (D_s, D_e)$ be two scenarios on a common state space. The scenario $S_C \cdot S_D$ is defined by

$$S_C \cdot S_D \stackrel{\text{def}}{=} (C_s \sqcap D_s, C_e \,[]\, D_e)$$

by the properties of $\sqcap$ and $[]$ this induces a commutative monoid with neutral element $1 \stackrel{\text{def}}{=} (\mathsf{chaos}, \mathsf{fail})$

now we can link up to our earlier theory:

Theorem 5.1 *Let* M *be a set of scenarios that is closed under* $\cdot$ *and contains* 1*. Then the structure* $(\mathcal{P}(\mathsf{M}), \cup, \emptyset, \cdot, \{1\})$ *is a product family algebra.*

in a similar way one can define a product family algebra for abstract machines á la B (recent MS Thesis by A. Zelend)

in practice the subclass of *feasible* commands is more realistic

closed under $[]$ but generally not under $\sqcap$

however, the check of the condition for closure is partially mechanizable (tool SCENATOR at McMaster)

first form of feature interaction detection in our approach

6 Conclusion and Future Work

main characteristics of our approach:

- very simple but effective mathematical background
- axiomatics purely first-order (theory of commands can be abstracted to semirings as well)
- hence automated reasoning (e.g., using Prover9/Mace4) is possible and has been done
- prototypical implementation of some useful models of product family algebra in *Haskell*

outlook:

- view commands in scenarios as initial implementations
- link with pre-/postcondition style reasoning for verification
- transform them into efficient form using the usual refinement laws
- extend the treatment of feature interaction
- investigate further semantic algebras
- carry the same ideas over to *feature algebra* (with Apel, Kästner, Lengauer) and maybe to *OFMs* (with Batory, Höfner, Smith)

References

Peter Höfner, Ridha Khedri, Bernhard Möller: An Algebra of Product Families. Software and Systems Modeling 10, 161-182 (2009/2011)

Peter Höfner, Ridha Khedri, Bernhard Möller: Supplementing Product Families with Behaviour. Int J. Software Informatics 5, 245-266 (2011)