Decision Algebra: Parameterized Specification of Decision Models

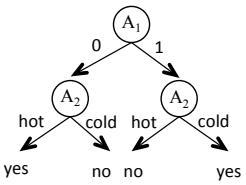
Antonina Danylenko[1], Wolf Zimmermann[2] and Welf Löwe[1]

[1] Linnaeus University, Software Technology Group,
351 95 Växjö, Sweden
{antonina.danylenko,welf.lowe}@lnu.se
[2] Martin-Luther-Universität Halle Wittenberg, Institut für Informatik,
061 20 Halle(Saale), Germany
{wolf.zimmermann}@informatik.uni-halle.de

Introduction. Processing decision information to adjust and alter applications' behaviour is a constitutive part in different fields of Computer Science, such as Data Mining, Software Engineering, Artificial Intelligence, etc. In general, decision information is an information that is used to deduce the relationship between a certain context and a certain decision. It is usually represented by a decision model (or classifier), which is a set of rules that determines a target decision based on the current context. Frequently used examples of decision models are decision tables, decision trees, support vector machines, etc.

Problem. In general, capturing decision information in decision models can have the problems like data replication (redundancy in the stored information) and model overfitting possibly leading to data fragmentation (the amount of information is too small to make a statistically significant decision) [1]. Thus, an important choice to be made is to select and implement a certain decision model for a particular problem domain. Issues such as model accuracy, capturing time, robustness, and scalability must be considered and can involve tradeoff between (1) the expressiveness of the model, i.e., the representation should be as close as possible to the initial captured data, (2) the memory required to keep this high-detailed decision model and (3) the time and initial data required in order to choose among the alternative decisions it can represent [2]. Moreover, developing or adjusting algorithms for processing this information might require adding new operations which, in general, have a negative impact on the problem domain memory requirements and performance.

Decision Algebra. Because of this variety of application domains with decision problems (each coming with different notations and tailored implementations) we consider it worthwhile to introduce a general algebraic specification, referred to as Decision Algebra, for describing an abstract data type that corresponds to a general decision model representation. Speaking of decision models we may have in mind decision models originated in Machine Learning filed, which keep different data (e.g. distributions, coefficients, probabilities) required for a correct decision making. Other examples may include decision models in Software Engineering, such as models representing program analysis results, software complexity models or service quality models.



spec *BinaryC* = *C*+
sorts *binaryC*
opns
$\sqcup$: *binaryC binaryC* → *binaryC*
yes: → *binaryC*
no: → *binaryC*

Fig. 1. Example of the Decision Tree *DT*

Every model has in general its own specification based on which different types of implementation are allowed. Therefore, it could be more effective to give one general specification for decision model, which can be parameterized, such that concrete instantiations may be obtained by suitable actualization of the parameter [3]. Such general algebraic specification for decision models can play an important conceptual role in software specifications where a decision process is involved, and development focused on functional programming.

Decision Algebra Specification. In our previous work [4] we introduced the Decision Algebra that was limited to decision trees and decision tables, where the specification was not parameterized and was particularly dedicated to certain decision model representation. The purpose of the current work is to present the Decision Algebra $DA(T, C, DF(T, C))$ as a parameterized specification that provides the representation of decision information as decision function *DF* along with a set of operations and equations. By replacing a formal parameter *DF* by an actual parameter specification we can obtain a new specification which defines a concrete decision model implementation.

spec $DA(T, C, DF(T, C)) = DF(T, C)+$
opns *evert: df int* → *df*; *approx: df int* → *df*; $apply^{(n)}$: *fun* $df^{(n)}$ → *df*
vars d_1, d_2 : *df*; *d*: $df^{(n)}$; *i* : *int*; *f* : *fun*
axioms $arity(approx(d_1, i)) \leq arity(d_1) = true$
$evert(d_1, i) \equiv d_1 = true$ $\quad arity(evert(d_1, i)) =^{int} arity(d_1) = true$
$apply^{(n)}(f, d) = eval^{(n)}\ (f, d)\ approx(d_1, i) \sqsubseteq d_1 = true$

A decision function $df^n : T \to C$, where $T = A_1 \times \ldots \times A_n$ is a tuple of contexts (attributes) that lead to a decision *C*, serves as a formal parameter specification which in turn is also parameterized by the specific decision information that is kept in a certain decision model, i.e. by concrete representation of tuple *T* and decision *C*.

spec $DF(T, C) = T+, C+, INT$
sorts *df, fun*
opns *decide: df t* → *c*; $\sqsubseteq$: *df df* → *bool*; $\equiv$: *df df* → *bool*;
arity: df → *int*[1]; $eval^{(n)}$: *fun* df^n → *df*
vars d_1, d_2 : *df*
axioms ($\forall$ *x: t, decide*(d_1, *x*) $\sqsubseteq$ *decide*(d_2, *x*) = *true*) ⇒ $\sqsubseteq$ (d_1, d_2) = *true*
($\forall$ *x* :*t, decide*(d_1, *x*) =^{c} *decide*(d_2, *x*) = *true*) ⇒ $\equiv$(d_1, d_2) = *true*

In this case the only requirement for parameter *C* is definition of partial order $\sqsubseteq$ over decisions $c_1, c_2 \in C$.

To specify parameter passing mechanism in this work we define a specification morphism that replaces a formal by an actual parameter specification and

[1] Note that *arity* operation specifies the number of attributes that influences the final decision

therefore results in a concrete decision model representation. For example, the decision tree model depicted in Figure 1 corresponds to the specific implementation $MyDT(T)$ of the decision function $DF(T, BinaryC)$ that is parameterized with the $BinaryC$ values representing the decision C.

spec $MyDT(T) = T+, BinaryC+, INT'+$
sorts $A_1, A_2, A_3, V_1, V_2, fun$
sub-sorts $A_1 \sqsubseteq df, A_2 \sqsubseteq df, A_3 \sqsubseteq df$
opns

tuple: $V_1 V_2 \rightarrow t$;	*merge:* $\rightarrow fun$;	$eval^{(2)}$*: fun df df* $\rightarrow df$;	*decide: t df* $\rightarrow$ *binaryC*;	
a_1*:* $A_2\ A_2 \rightarrow A_1$;	a_2*:* $A_3\ A_3 \rightarrow A_2$;	$\sqcup$*:* $A_1\ A_1 \rightarrow A_1$;	$\sqcup$*:* $A_1\ A_2 \rightarrow A_1$;	
leaf: $c \rightarrow A_3$;	*0:* $\rightarrow V_1$;	*1:* $\rightarrow V_1$;	*hot:* $\rightarrow V_2$;	
Π_1*:* $T \rightarrow V_1$;	Π_2*:* $T \rightarrow V_2$	*cold:* $\rightarrow V_2$;		

vars x_1, x_2, u_1, u_2*: df;* t_1, t_2*: t,* c_1, c_2*: binaryC*
axioms *(decide(t, leaf*$(c_1)) = c_1$*;* $eval^{(2)}(merge, x_1, x_2) = x_1 \sqcup x_2$
$(\Pi_1(t_1) = 0) \Rightarrow decide(t_1, a_1(x_1, x_2)) = decide(t_1, x_1)$
$(\Pi_1(t_1) = 1) \Rightarrow decide(t_1, a_1(x_1, x_2)) = decide(t_1, x_2)$
$(\Pi_2(t_1) = hot) \Rightarrow decide(t_1, a_1(x_1, x_2)) = decide(t_1, x_1)$
$(\Pi_1(t_1) = cold) \Rightarrow decide(t_1, a_1(x_1, x_2)) = decide(t_1, x_2)$
$a_1(x_1, x_2) \sqcup a_2(u_1, u_2) = a_1((x_1 \sqcup a_2(u_1, u_2)), (x_2 \sqcup a_2(u_1, u_2)))$
$a_2(x_1, x_2) \sqcup a_2(u_1, u_2) = a_2((x_1 \sqcup u_2), (x_2 \sqcup u_2))$
$leaf(c_1) \sqcup leaf(c_2) = leaf(c_1 \sqcup c_2)$; $approx(a_1(x_1, x_2), 1) = x_1 \sqcup x_2$
$approx(a_1(a_2(x_1, x_2), a_2(u_1, u_2)), 2) = a_1((x_1 \sqcup u_1), (x_2 \sqcup u_2))$

Outcome. Central idea of the Decision Algebra is the idea of a function, which computes a result that depends on the values of its inputs. This very much correlates with the ideas of functional programming that gives the clearest possible view of abstraction (in a function) and data abstraction (in an abstract data type) [5, 6]. Therefore, using our Decision Algebra, decision models can be directly implemented as a data type in functional languages, whereas in other languages one is forced to describe them by different specific data structures with a number of additional operations. Another strengths of having a such general algebraic specification for decision models is that we can define general functions which can be used in different applications and, therefore, construct the pattern of computation: transform every decision model in a same way and combine the decision models using same operator. Due to this generalization, insights can be gained at an abstract level or reused between application domains, paving the way for a deeper problem understanding and, possibly, for novel and more efficient algorithms for combining different decision models.

References

1. M. S. P.-N. Tan and V. Kumar, *Introduction to Data Mining*. Addison Wesley, 2005.
2. T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.
3. H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer-Verlag, 1985.
4. A. Danylenko, J. Lundberg, and W. Löwe, "Decisions: Algebra and Implementation," in *7th International Conference on Machine Learning and Data Mining (MLDM 2011)*, New-York/USA, August-September 2011.
5. M. Odersky, L. Spoon, and B. Venners, *Programming in Scala: A Comprehensive Step-by-step Guide*, 1st ed. USA: Artima Incorporation, 2008.
6. S. Thompson, *The Haskell: The Craft of Functional Programming*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.