A Model Management Imperative: Being Graphical Is Not Sufficient, You Have to Be Categorical

Zinovy Diskin[1,2(✉)], Tom Maibaum[1], and Krzysztof Czarnecki[2]

[1] NECSIS, McMaster University, Hamilton, Canada
{diskinz,maibaum}@mcmaster.ca
[2] University of Waterloo, Waterloo, Canada
{zdiskin,kczarnec}@gsd.uwaterloo.ca

Abstract. Graph-based modeling is both common in and fundamental for Model Driven Engineering (MDE). The paper argues that several important model management (MMt) scenarios require an essential extension of graphical models. We show that different versions of model merge and sync, including many-to-many correspondences between models, can be treated in a uniform, compact and well-defined mathematical way if we specify graphical models as directed graphs with associative arrow composition and identity loops, that is, as categories.

1 Introduction

Graph-based modeling is common in and fundamental for MDE. Such graphical models as Class and ER-diagrams in structural modeling, and Labeled Transition Systems (LTSs) and Sequence Diagrams in behavioral modeling, are real assets to software engineering. The goal of the paper is to argue that several important model management scenarios require an essential extension of graphical models. We show that different versions of model merge and sync (choice and parallel composition in the context of behavior modeling), including many-to-many correspondences between models, can be treated in a uniform, compact and well-defined mathematical way if we specify graphic models as directed graphs with associative arrow composition and identity loops, i.e., as categories.

This is *not* the first call for categories from the model management (MMt) domain. Formalizing model merge as colimit in the respective categories of models and mappings is well known [3,19,22,26], the category-theory based graph transformation framework for model transformations is in active development [14,17,18,21], and a broad categorical view of MMt can be found in [10,24]. In these papers, MMt operations are formalized as operations over (typed attributed) graphs, and MMt scenarios thus live within a suitable category of graphs. We can say that this work advocates the use of *categories-in-the-large*. In contrast, in the present paper we show that an accurate mathematical modeling of complex MMt scenarios, like merging LTSs modulo complex many-to-many correspondences between them, or synchronized parallel composition of LTSs,

© Springer International Publishing Switzerland 2015
G. Taentzer and F. Bordeleau (Eds.): ECMFA 2015, LNCS 9153, pp. 154–170, 2015.
DOI: 10.1007/978-3-319-21151-0_11

require models themselves to be considered as categories—we refer to this idea as to the use of *categories-in-the-small.*. Importantly, categories we need may be subject to non-trivial commutativity constraints and thus are *not free* categories generated by the respective graphs. That is, they are "real" categories rather than categorical completions of graphs. We thus show that a full realization of the known categories-in-the-large framework needs categories-in-the-small.

The paper makes three contributions to the literature. The first one is methodological as described above. The second is technical: we propose a novel extension of the classical notion of LTS based on ideas of the *enriched* category theory. Our LTSs as categories carry an additional structure of partial order between "parallel" (having the same source and target) transitions. Moreover, this order is actually a semi-lattice carrying OR (and perhaps partially defined AND) operations on transitions. Third, we explain how model merge and composition can be specified using (co)limits in a tutorial-like manner, and specially focus on adequacy of the mathematical models we build for the subject matter: constructs included into the models are motivated by the domain to be modelled rather than by mathematical completeness.

Our plan for the paper is as follows. In Sect. 2 we consider several scenarios of LTS merge, and motivate the necessity to include into the LTS formalism sequential and *or/and*-parallel composition of transitions. In Sect. 3 we show that adding to the LTS formalism identity (idling) loops allows us to treat synchronized parallel composition of LTSs via a limit operation. In these sections, considerations are accurate but not formal. In Sect. 4 we formally define an LTS as a functor from a category of states and transitions (enriched over the category of posets) to a category of labels (similarly enriched). This defines a category in which our merge-as-colimit and sync-as-limit scenarios are unravelled. Section 5 presents a summarizing discussion, and Section 6 concludes.

2 Model Merge and Colimit

We use *model merge* to refer to the following scenario. Several models expressing *local* views of the system are first *matched* by linking elements of the models that correspond to the same system element. Then models are integrated into a single *global* model, which includes all data from the local models but without redundancy, in accordance with the match. We will consider three typical cases of intermodel relationships and show that if the complexities of intermodel relationships are properly modeled, then the merge procedure as such is given by the same simple colimit operation. However, the universe in which the merge-as-colimit idea works well for complex intermodel correspondences is a category of graphs with a suitable *additional structure* of sequential and parallel arrow composition, i.e., the universe of enriched categories.

2.1 Getting Started: Simple Match and Merge

Figure 1 presents two consumption models. Model M_1 states that buying an apple is a wise way of spending your dollar, but first you need to work and earn it. Model M_2 suggests spending the dollar earned on buying a cake.

Suppose we want to merge the two models into an integrated model U without data redundancy and loss. For this, we first specify correspondences between the models by bidirectional links r_x ($x = 0, \$, w$) connecting (we also say *matching*) elements considered to be "the same" as shown in Fig. 1(a). Then we (disjointly) merge the two graphs and glue together matched elements (and only them). Such merge is easily performed "by hand", resulting in model $U = (M1 + M2)/R$ in Fig. 1(b) (read "merge of models $M1$ and $M2$ modulo correspondence $R = \{r_0, r_w, r_\$\}$").

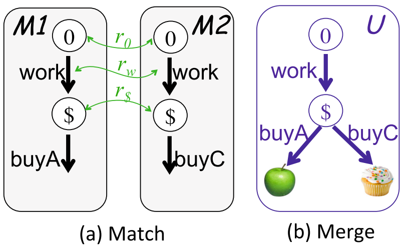

Fig. 1. Model-based merge (case 1)

The merge procedure can be formally specified as calculating the *colimit* of the *correspondence span* in the category of graphs as shown in Fig. 2. In this figure, the correspondence links r_x ($x = 0, \$, w$) from Fig. 1(a) are reified as, resp., states (for $x = 0, \$$) and transitions ($x = w$), which constitutes LTS R. The special nature of R's elements is formalized by two projection mappings r_i: $R \to M_i$, $i = 1, 2$.[1] Thus, matching models results in a correspondence span (in the sequel, *corr-span*) (r_1, R, r_2). Taking the colimit of the corr-span results in a model U together with two mappings u_i: $M_i \to U$, $i = 1, 2$, specifying how local models are embedded into the merge.

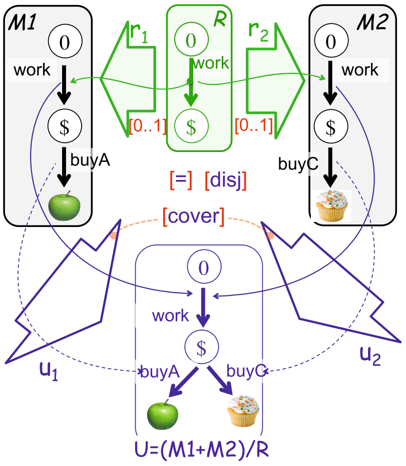

Fig. 2. Map-based merge (case 1)

For Fig. 2 and subsequently, we use the following shading/color schema. Given models and mappings are shaded, while algebraically computed ones are blank (and additionally blue with a color display; the blue color is assumed to recall mechanical computation). Although the elements of the corr-span are produced by the procedure of model matching, this procedure cannot be considered a formal algebraic operation taking two models and returning a corr-span between them. Indeed, correspondences are derived using various contextual and heuristic information *about* the models rather than immediately contained *in* them; moreover, model matching may require an input from the user. Thus, speaking algebraically, the corr-span is a given datum and hence shaded. However, its color is green rather than black to recall the special nature of the

[1] To avoid clutter in Fig. 2 and other our figures, in order to specify a mapping between graphs, we only show links between arrows and assume that their sources and targets are linked respectively, and those links are omitted.

heuristics-based model match quasi-operation. Constraints are shown in red; if a constraint is a postcondition of an operation, its name is typed in blue but enclosed by red brackets.

Thus, the colimit operation takes a span and produces a *cospan* (two mappings with a common target). Importantly, the colimit cospan can be uniquely defined (up to isomorphism) by a set of constraints (postconditions) $\{C1, ..., C4\}$ it must satisfy. Moreover, these constraints well correspond to natural requirements for model merging. Constraint $C1$ requires mappings u_1, u_2 to be total: nothing from models M_i is lost in the merge U. In our diagrams, all mappings are assumed to be total (and single-valued) by default, and the corresponding multiplicity constraints are omitted. Constraints $C2, C3$ specify properties of the quadruple (r_1, r_2, u_1, u_2). Constraint $C2$ is *commutativity* [=]: $x.r_1.u_1 = x.r_2.u_2$ for any element $x{\in}R$ (note the closed contours formed by links starting at R in Fig. 2), which ensures that elements linked by the corr-span are glued in U. On the other hand, constraint $C3$ (*disjointness*) requires *non*-linked elements *not* to be glued: if an element $x{\in}M_1$ is outside the range of mapping r_1, then element $x.u_1 \in U$ must be outside the range of u_2, and if $x.u_1 = x'.u_1$ then $x = x'$. Similar conditions are required for any $y{\in}M_2$ outside the range of r_2. Finally, constraint $C4$ states that two mappings u_i jointly *cover* the graph U, and hence every element in U has come from either $M1$, or $M2$, or both. It can be proven [8] that in the universe of graphs and graph mappings, given a corr-span (R, r_1, r_2), there is one and only one (up to isomorphism) cospan (U, u_1, u_2) satisfying the four constraints above. This cospan is called the *colimit* of the span, and it can be computed by a simple algorithm (see, e.g., [1] or [8]). Thus, the colimit operation in the category of graphs accurately captures the requirements for merge in the case of simple one-one correspondences between the models.

Our next goal is to analyse whether the simple pattern above works for more complex cases of intermodel relationships.

2.2 Complex Match and Merge via Derived Transitions

Practical intermodel relationships are often more complex than the one-to-one matches considered above. A simple *one-to-many* match is shown in Fig. 3(a). Model M_1 says you can convert a dollar into a smile by either buying and eating an apple, or by buying and eating a cake. Model M_2 is more abstract and says you can convert a dollar into a smile by firing either *healthyLife* or *happyLife* transitions, but is not specific about details of what should be done. Suppose we know that by *healthyLife* model M_2 actually means buying and eating an apple, so that transition *healthyLife* matches two transitions in M_1. Similarly, transition *happyLife* matches buying and eating a cake. Hence, corr-links in Fig. 3(a) must be accompanied with equations (E1)

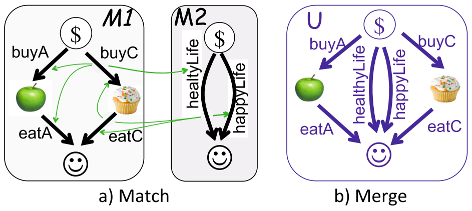

Fig. 3. Model-based merge (case 2)

healthyLife = *buyA*; *eatA* and (E2) *happyLife* = *buyC*; *eatC* specifying details of the relationships. Such constraints are often called *(correspondence) expressions* in the literature on model match and merge [2].

The merge can easily be produced by hand Fig. 3(b), but now LTS U as shown is underspecified: constraints (E1), (E2) are not shown. To take such expressions into account, the merge algorithm has to be more intelligent and actually more complicated than for one-to-one matching. Managing corr-expressions was declared as one of the big problems of model management in [2].

Let us see how the problem can be treated categorically. A key observation is that a one-to-many relationship is replaced by a one-to-one relationship to a respective *derived* transition, *buyA*; *eatA* for *healthyLife* and *buyC*; *eatC* for *happyLife*, as shown in Fig. 4. Derived transitions are shown by dashed arrows (blue with a color display), and the respective triangle diagrams are marked with symbol [;] referring to the operation of arrow composition. The left marker [;] says that the left unnamed transition is *buyA*; *eatA*, and analogously for the right [;].

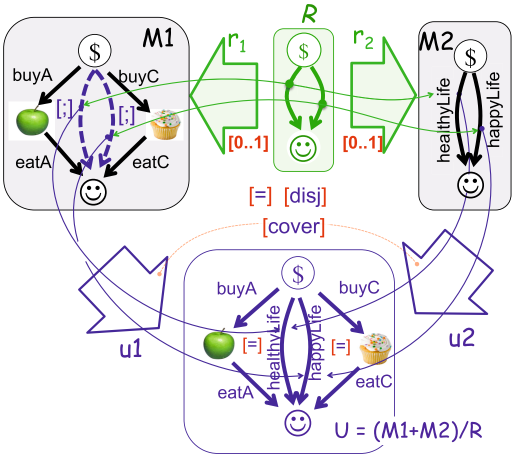

Fig. 4. Map-based merge (case 2)

Use of derived transitions allows us to reduce *one-to-many* to *one-to-one* matching, and then the simple merge algorithm described above can be directly applied and produces the result shown in the lower part of Fig. 4. The main merge principle of copying all data from original models into the merge is realized, and operation labels [;] are copied to U as well, but as derived transitions are named in U and hence appear there as basic rather than derived elements, their *definitions* in M_1 amount to equational *constraints* [=] in U. Thus, constraints $E1, E2$ specified above for the match, are now transferred to the merge. Note that disjointness [disj] of (r_1, r_2, u_1, u_2) amounts to injectivity of u_1 and u_2.

The same idea is applicable for other operations on transitions, e.g., their AND (parallel) or OR (choice) composition, as the example below demonstrates. Suppose that model M_2 has only one transition *beHappy*, which can mean either buying and eating an apple, or, perhaps, buying and eating a cake. With the naive

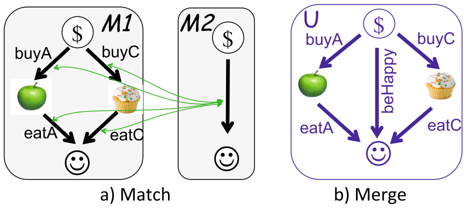

Fig. 5. Model-based merge (case 3)

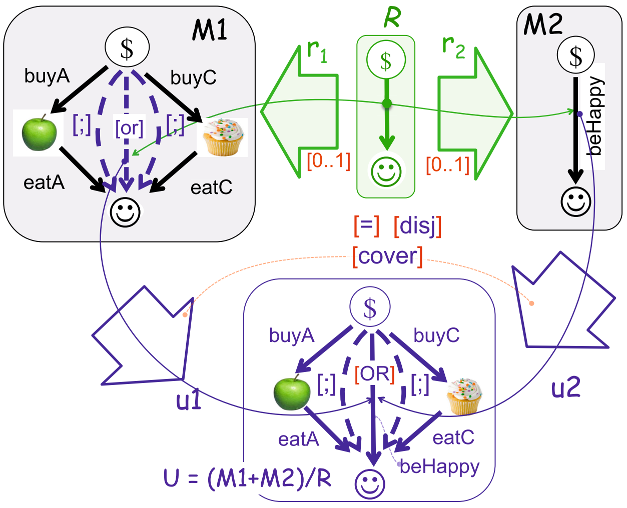

Fig. 6. Map-based merge (case 3_1)

expression-based approach, we would have a match in Fig. 5(a) accompanied with expression (E) specifying the relationship:
(E) $\quad beHappy = buyA; eatA$ or $buyC; eatC$.

We can specify this relationship by a one-to-one match as shown in Fig. 6. We first derive two composed transitions in model M_1 as above, and then apply to them the binary choice operation [or]. The result is the vertical transition in M_1, which is matched to the *beHappy*-transition in M_2. Now we apply the same simple colimit procedure as above, and produce model U as shown in the figure. As the middle transition now has a name, we replace its *definition* in model M_1 by a *constraint* [OR] specified by the expression E above. Similarly, if transition *beHappy* in M_2 assumes, in terms of model M_1, buying and eating both apples and cakes, we introduce a binary operation [and] on transitions, and match *beHappy* with derived transition *buyA;eatA*[and]*buyC;eatC*. The result is again a one-one match involving derived transitions.[2]

2.3 Non-injective Match and Merge

There is a different interpretation of the naive match in Fig. 5. We may assume that transition *beHappy* in M_2 is not a choice between *healthyLife* and *happyLife*, but rather their abstraction that simply does not distinguish between them. A direct way of modeling this interpretation is specified by the corr-span in Fig. 7, in which the right leg r_2 maps two transitions to the same target. The colimit

[2] To make operation [and] feasible, we need to interpret state \$ as providing enough money for the concurrent execution of all transition leaving the state, and similarly the smile-state means a holistic happiness rather than a number of "happiness units".

of this span is shown in the lower half of the figure; note that two different transitions in model M_1 are glued together in the merged model U.

At first sight, this gluing in the merge can seem bizarre, but let us consider the case in more detail. The match says that transition *beHappy* is "equal" to *buyA;eatA* and to *buyC;eatC* as well. Hence, *buyA;eatA*=*buyC;eatC*, so that the match reveals a new constraint on model M_1 not initially declared in model M_1.

This is not a rare situation: some properties of an object are only revealed (*emerge*) when this object is related to other objects (and this is what category theory is about). As model merge should preserve all input information, the constraint about M_1 stated by the corr-span is to be respected in the merged model U, and this is exactly what the colimit does. Note how accurately this situation is treated categorically: neither of the original models is changed, but everything needed is captured by a properly specified corr-span.

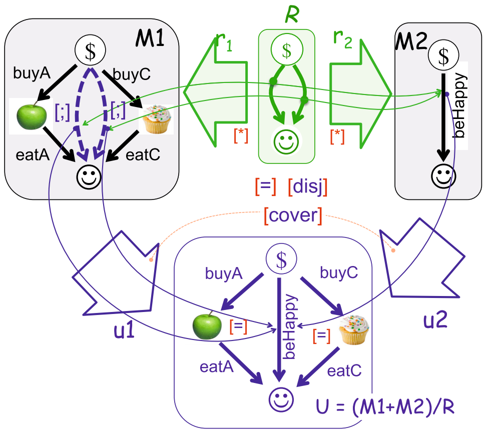

Fig. 7. Map-based merge (case 3_2)

3 Parallel Composition and Limits

Parallel composition of executable components (or behaviors) is a fundamental operation of behavior modeling. It assumes that several (local) components run simultaneously so that global states and transitions are tuples of local states and transitions. Some local transitions from different components may be synchronized, i.e., be always executed simultaneously (as a *handshake*). We will show that parallel composition can be specified categorically by an operation called *limit*, or else *synchronized product*, which places it into the realm of categorical methods and ideas. In particular, as limits are dual to colimits in some precise sense, model merge and parallel composition appear as dual scenarios; we will often call the former *additive*, and the latter *multiplicative* (or *parallel*) *merge*.

Suppose that the two models from Sect. 2.1 specify behaviors of two consumers acting in parallel (following a similar example in the textbook [20, p.42], we call them Ben and Bill). We assume that Ben and Bill work together on a joint project to earn their dollars, but buy their snacks separately and independently, i.e., concurrently. In our case, a concurrent composition can be easily constructed by hand, and is often described by diagrams like in Fig. 8 explaining the name *interleaving concurrency*.

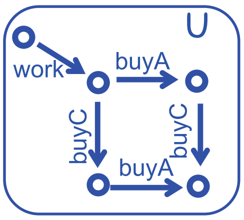

Fig. 8. Interleaving

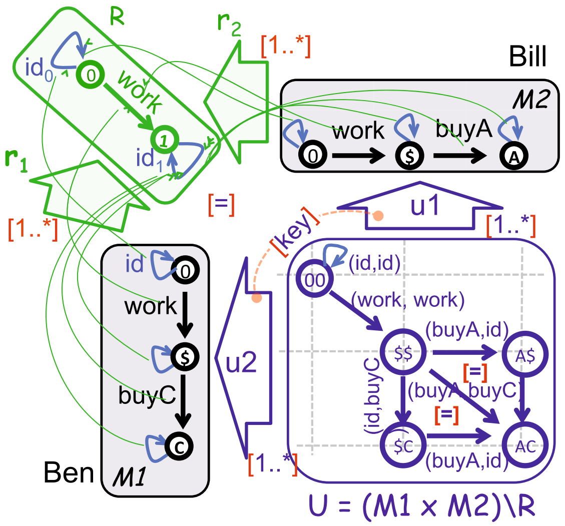

Fig. 9. Synchronized parallel composition

To consider the example in more detail, we add to both models *idle-loop* transitions as shown in Fig. 9 (ignore links to and from the models for a moment). That is, every state in a LTS is endowed with a loop denoting the do-nothing transition from the state to itself. Independence of Ben's and Bill's buying can now be expressed precisely: irrespective of what Ben is doing (idling or buying), Bill can idle or buy. The joint behavior is shown by LTS U in the right-lower corner of Fig. 9 (Expression $U = (M_1 \times M_2) \setminus R$ refers to U as a product of M_1 and M_2 modulo correspondence R, which we will consider later.) States in U are pairs of local states, and transitions are pairs of local transitions. Horizontal and vertical transitions in U are those in which one component is idling, whereas diagonal transitions combine two non-idle actions (and idle loops in the product are pairs of idle loops, only one such loop is shown). Thus, idle transitions are fundamental for modeling parallel composition. Note also that both triangle diagrams are declared to be commutative: from the global viewpoint, it does not matter how the system transitions from two dollars to two snacks—all paths are equal. This commutativity appears to be a precise formal counterpart of Ben's and Bill's independence in buying snacks.

As U's elements are pairs, each of them is supplied with a pair of links to the respective elements in M_1 and M_2. To avoid clutter, these links are not shown but can be easily restored as horizontal and vertical projections of U's elements; they constitute a pair of mappings $u_i\colon U \to M_i$, and the constraint [key] states that any element $e \in U$ is uniquely identified by pair $[u_1(e), u_2(e)]$. Both mappings $u_i\colon U \to M_i$ are surjective. This captures simultaneity of the parallel composition: any local element (state or transition) must become a component of a global element.

So far, we have built the composed behavior U by reasonable "physical" considerations. Remarkably, the same LTS U can be automatically computed by an operation called *limit*, if synchronization between the models is properly specified. This specification is given by mappings $r_i\colon M_i \to R$, $(i = 1, 2)$ from the local models to a common model R representing the *global (synchronized)* view of the behavior. We call the triple (r_1, R, r_2) a *synchronizing cospan*. The global view should evidently contain a global transition *work*, composed of two local instances of *work* acting in parallel (note the corresponding links in the r_i). Further, there are two global idle loops. The first, id_0, is a pair of local idle transitions, shown by the respective links in the cospan (r_1, r_2). The second, id_1, is much more interesting. It is the common target of several local transitions from both sides, meaning that, from the global viewpoint, the differences between the respective states are non-essential, and all actions are globally considered as idling, i.e., changes they cause are abstracted away in the global view.

Now the desired result of synchronized composition can be formally defined by a simple formula: $U \overset{\text{def}}{=} \{(e_1, e_2) \in M_1 \times M_2 \colon\ r_1(e_1) = r_2(e_2)\}$, which gives exactly the LTS U in Fig. 9 motivated "physically"; mappings u_1, u_2 are canonic projections. Note the commutativity of square (r_1, r_2, u_1, u_2): paired local transitions should be globally indistinguishable. The operation producing span (U, u_1, u_2) specified by the formula above from a cospan (R, r_1, r_2) is called the *limit* (in the category of LTSs). We denote the result by expression $U = (M_1 \times M_2) \setminus R$ (read "product of M_i modulo sync-cospan R"), and also call it the *synchronized product*. Thus, the MMt operation of LTS parallel merge can be formally defined as the LTS sync-product, but the latter requires having in the LTS formalism sequential composition of transitions and idle transitions.

4 LTSs: From Graphs to Categories to Enriched Categories

We formalize constructions described above in a categorical way. We first motivate state labels and make labeling a *functor* from a category of transitions to a category of labels; then we specify model merge and parallel composition as operations over functors (Sect. 4.1 and 4.2). We then consider how to model OR- and AND-composition of transitions as operations over arrows, and come to the notion of an *enriched LTS* (Sect. 4.3). We assume that elementary notions of category theory are known to the reader (see, e.g., [1]), but to fix notation and terminology, we provide several basic definitions in the footnotes.

4.1 Labeling as a Functor

A *(directed) graph* G comprises a set $G^\bullet$ of *nodes*, a set $\overrightarrow{G}$ of *arrows*, and two functions, $\mathsf{so}\colon \overrightarrow{G} \to G^\bullet$ and $\mathsf{ta}\colon \overrightarrow{G} \to G^\bullet$. Given nodes X, Y, we write $a\colon X \to Y$ if $X = \mathsf{so}(a)$ and $Y = \mathsf{ta}(a)$, and denote the set of all arrows from X to Y by $\overrightarrow{G}(X, Y)$. We write $e \in G$ to say that $e \in G^\bullet \cup \overrightarrow{G}$ is an element of graph G.

A classical LTS is a graph T whose nodes are called *states* and arrows are *transitions*; the latter are labeled via a function $\lambda\colon \overrightarrow{T} \to L$ into a predefined set L of *(action) labels*. As our examples of LTS merge and sync showed, we need to have in the LTS formalism sequential composition of transitions and idle transitions. Moreover, in the behavioral modeling context, associativity of composing transitions, and identity equations for idling, are very natural conditions.

In other words, the transition graph should be a category.[3] However, labels form a set and, thinking categorically, mapping a category (even a graph) to a set is not natural. Hence, we assume that states also have labels. Indeed, as a rule, applying a transition to a state requires the latter to satisfy some pre-conditions, and the result of transition execution satisfies some post-conditions. These pre- and post-conditions can be encoded by *state labels*, which together with transition labels form a graph, even a category, and labeling should be compatible with those categorical structures. Thus, an LTS becomes a triple $M = (T, \lambda, L)$ with T a category of (states and) transitions, L a category of labels, and labeling $\lambda\colon T \to L$ a mapping of categories, i.e., a functor.[4] For example, the category of labels for our model of buying and eating apples and cakes could consist of three nodes \$, *snack*, and ☺ (see model $L1$ in Fig. 10); two arrows *buy*: \$→*snack* and *eat*: *snack*→☺, their composition *life*, and three idle loops (omitted in Fig. 10).

Fig. 10 also shows an accurate specification of labeling: in model $T1$, elements' names before colons refer to states and transitions, while names after colons refer to their labels. The latter can be considered as types, and the former as their instances. Importantly, labeling preserves arrow composition and idle loops. Similarly, for model $M2$, the category of labels could be taken to be a single arrow *life*: \$ →☺ (plus two idle loops), while the transition category has two non-trivial arrows *healthy* and *happy*. Note that the classical LTS notion is subsumed if we require the set $L^\bullet$ of state labels to be a singleton: then there is only one state label, and a transition can be always applied to a state.

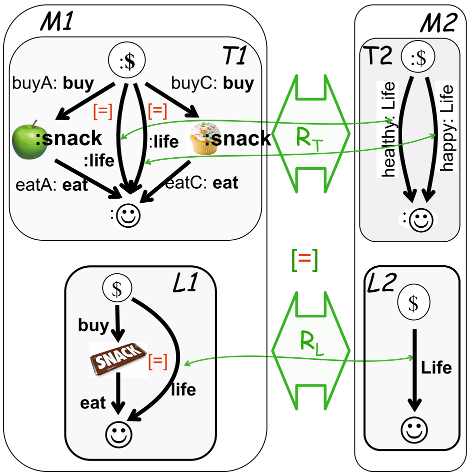

Fig. 10. Labeling

An important property of the transition category $T1$ of model $M1$ in Fig. 1 is that it is *freely* generated by the respective base graph by adding to it all idle loops and all possible arrow compositions (subject to associativity and identity equations). This is how we specified this category in Fig. 4: the basic transitions

[3] A *category* is a graph with (i) an associative arrow composition and (ii) identity loops. Point (i) means that for any pair (a, b) of consecutive arrows, an arrow $a; b$ is defined, and $(a; b); c = a; (b; c)$. Point (ii) means that every node $X \in G^\bullet$ is assigned with *identity* arrow $\mathsf{id}_X\colon X \to X$, and $\mathsf{id}_X; a = a = a; \mathsf{id}_Y$ for any $a\colon X \to Y$.

[4] Given categories C_1, C_2, a *functor* $f\colon C_1 \to C_2$ is a graph morphism that preserves composition and identity loops. That is, $f(a; b) = f(a); f(b)$ for any pair of consecutive arrows a, b in C_1, and $f(\mathsf{id}_X) = \mathsf{id}_{f(X)}$ for any node X in C_1. In its turn, being a graph *morphism* provides preservation of incidence between nodes and arrows: $\mathsf{so}(f(a)) = f(\mathsf{so}(a))$ and $\mathsf{ta}(f(a)) = f(\mathsf{ta}(a))$ for any arrow $a \in C_1$.

are shown by black solid arrows while derived composed transitions are dashed (and blue with a color display); idle loops were omitted.

Importantly, even if local transition categories are freely generated, LTS merge can result in categories with non-trivial equations. Consider, for instance, our example in Fig. 7: even if we ignore the name of the middle transition in the merged LTS, and hence can remove it from the model (it can be restored by composition if needed), the commutativity constraint $buyA;eatA = buyC;eatC$ must be kept. Thus, the universe of LTS merge is the universe of "real" categories rather than graphs and freely generated categories. This discussion motivates the following definition.

Definition 1. An *LTS* is a triple $M = (T, \lambda, L)$ with T and L categories of transitions and labels resp., and $\lambda\colon T \to L$ a functor. □

4.2 Model Management for LTSs

Our examples in Sections 2 and 3 show that the notion of LTS morphism is fundamental for LTS model management. As an LTS is a two-layer structure, their morphisms are also two-layered.

Definition 2. An *LTS morphism* $m\colon M \to M'$ is a pair of functors, $m_T\colon T \to T'$ and $m_L\colon L \to L'$, which commute with labeling: $m_T;\lambda' = \lambda; m_L$. □

For example, matching LTSs results in two corr-spans, between labels and between transitions, such that commutativity between matches and labeling holds as shown in Fig. 10, where we use bidirectional arrows as a succinct notation for triples (left link, corr-object, right link). Our diagrams in Sections 2 and 3 were not quite accurate by leaving labels implicit. An accurate description would be to leave diagrams as is, but provide the possibility to zoom-in and reveal the two-layer picture shown in Fig. 10. Correspondingly, operations over LTSs we considered are also two-layered: they begin with a two-layer (parallel) match followed by a two-layer (parallel) merge. As we argued, it makes sense to formally define these operations as colimit (for merge) and limit (for parallel merge/composition). After that, category theory provides the respective machinery roughly sketched below.

Let Cat denote the category of all (small) categories. Then, according to our formal definitions of LTSs and their morphisms, the category of all LTSs is nothing but the arrow category $\mathsf{Cat}^{\to}$ over Cat: its objects are Cat-arrows, i.e., functors, and their morphisms are commutative squares described in Definition 2. It is well known that such an arrow category has all (two-layer) limits and colimits with well known simple algorithms computing them [1]. (A simple description of these algorithms can be found in our TR [8].)

4.3 Parallel Composition of Transitions

The most immediate way to formalize OR- and AND-parallel composition motivated in Sect. 2 is to enrich the set of transitions $T(X, Y)$ for any pair of states X, Y with a partial order $\sqsubseteq_{XY}$ (we will often omit the subindex), and define binary OR and AND operations as taking the least upper bound (LUB) and the greatest lower bound (GLB) of two transitions w.r.t. $\sqsubseteq$. A natural "physical" interpretation of relation $\sqsubseteq$ is that stating $t1 \sqsubseteq t2$ means that transition $t1$ is a specialization of transition $t2$, for example, stating *happy* $\sqsubseteq$ *healthy* (both of type *Life*) means that *happy:Life* assumes *healthy:Life* (but needs more than that). Particularly, firing transition $t1$: $X \to Y$ automatically implies transition $t2$: $X \to Y$ (but not necessarily the converse). Categorically, specialization can be seen as a special *2-arrow* between transitions (*1-arrows*); in structural modeling, such arrows are often called *isA*-arrows. Now OR-composition of transitions $t1, t2$: $X \to Y$ is defined to be their LUB: firing any of ti automatically means firing $t1 \vee t2$. Similarly, we could define AND-composition of transitions (true concurrency) as taking their GLB wrt. $\sqsubseteq$: firing $t1 \wedge t2$ means firing both $t1$ and $t2$. This operation makes perfect sense in our context if we interpret state \$ as having as much money as necessary for buying all snacks, and correspondingly state $\smile$ as happiness provided by eating any number of snacks.

There is an essential difference between OR and AND compositions. It is reasonable to assume that any two transitions can be OR-composed—this is an explication of the fact that choice is inherited in the notion of LTS. In contrast, AND-composition (which means the concurrent execution) may exist for some transitions, but may not exist for others (which cannot run concurrently).

An immediate formalization of these ideas is provided by the notion of a category *enriched* over the category of posets Pos.[5]

Definition 3. An *OR-enriched* LTS is a triple (T, λ, L) with T and L categories of transitions and labels resp. enriched over Pos^+, and λ: $T \to L$ an Pos^+-enriched functor. The notion of an *(OR,AND)-enriched* LTS is defined similarly.

Specifically if we have transitions $t1$ and $t2$ with labels $l1$ and $l2$ resp., then OR-composition of $t1$ and $t2$ must be labeled by OR-composition of $i1$ and $l2$; and similarly for AND (because we require functor λ to be enriched). Note that the category of labels can be trivially enriched: $t1 \sqsubseteq t2$ iff $t1 = t2$. Then enrichedness of the labeling functor λ implies that only transitions with the same label can be OR- or AND-composed. In this way enriched labeling brings a strict type discipline to modeling with LTSs.

Colimits of enriched LTSs are computed componentwise: objects, arrows, 2-arrows. For instance, if in our merge scenario in Fig. 4, model $M2$ would have a 2-arrow from happyLife to healthyLife (i.e., we state *happy* $\sqsubseteq$ *healthy* over label *Life*), then the merged LTS would also have this 2-arrow, which together with commutativity constraints implies *buyC;eatC* $\sqsubseteq$ *buyA;eatA*. This is yet another

[5] A category C is *enriched* over Pos if for any two objects $X, Y \in C^\bullet$, collection of all arrows $C(X, Y)$ from X to Y is a poset. Moreover, arrow composition $\_;\_ : C(X, Y) \times C(Y, Z) \to C(X, Z)$ is a poset morphism (i.e., an order-preserving mapping) for all triples X, Y, Z. A functor f: $C_1 \to C_2$ between Pos-enriched categories is Pos-*enriched*, if f_{XY}: $C_1(X, Y) \to C_2(f(X), f(Y))$ is a poset morphism for all pairs X, Y. In a similar way we can define enrichedness over category Pos^+ of all posets with finite LUBs, or $\mathsf{Pos}^\times$ of all posets with finite GLBs, or category $\mathsf{Pos}^{+\times}$ of all lattices by requiring preservation of, resp., LUBs, GLBs, or both.

illustration of how one model ($M2$ in this case) imposes a constraint on another model ($M1$). Also note that even if local LTSs are freely enriched over Pos^+ (OR-compositions are added for all pairs of transitions), their colimit can be enriched non-freely. For example, in Fig. 6, the merged model U satisfies constraint $beHappy = buyA;eatA \vee buyC;eatC$ denoted by [OR].

5 Observations, Discussions, Future and Related Work

We will first discuss briefly several possible practical and theoretical applications of the framework, and then consider related work.

Universality of (co)limits and Specifications for MMt Tools. Categorical formalization of merge as colimit, and parallel composition as limit, reveals a remarkable duality between these two types of model management scenarios. The general patterns of the two operations are shown in Fig. 11. Colimit takes a corr-span and produces a cospan, while limit takes a sync-cospan and produces a span: the diagrams are mutually convertible by inverting directions of all arrows. Importantly, both operations can be uniquely defined by the respective post-conditions. We discussed them for colimit in Sect. 2.1, and similar conditions can be formulated for limit as well. In addition to constraints [=] and [key], we need to require the limit span to be *maximal* in some precise technical sense, which ensures that *all* globally indistinguishable pairs are collected in U. Dually, properties [cover] and [disj] provide *minimality* of the colimit cospan. Thus, the existence of (co)limits is a property of the universe of models and mappings rather than a user-defined superstructure (details can be found in any category theory textbook, e.g., [1]). In the categorical literature, defining (co)limits via their postconditions (minimality or maximality) is referred to as their *universality*.

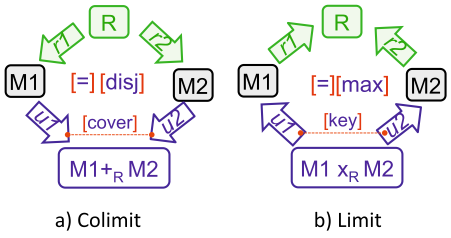

Fig. 11. Duality

Universality of (co)limits may be important for the proper design and use of MMt tools. As shown in [16], miscommunication between tool users and tool builders can be a major problem in MDE practice (see [9] for a detailed discussion). The possibility of defining model merge and parallel composition via (co)limit operations, which in their turn can be defined universally via postconditions, opens the door for specifying semantics of MMt tools in an unambiguous and precise way. We do not want to say that, e.g., any model merge scenario can be reduced to colimit, but colimit is a core concept of model merge to which specific merge scenarios and tools should be related. Considering a particular model merge scenario as either a particular case of an incomplete colimit, or the complete colimit, or a particular way of postprocessing the colimit (e.g., to resolve conflicts produced by the colimit), can guide the tool design, and facilitate communication between the tool users and tool builders. We plan a comparative study of model merge tools w.r.t. the colimit "yardstick" for future work.

MMt Colors and Tools. Either type of model merge consists of two stages: the green and the blue. The former is model match, which requires heuristics, contextual analysis, and often some user input. After models are matched and a corr-(co)span is produced, their merge is a routine and fully automatic (blue) operation (colimit or limit). The green and the blue stages of model merge are different both methodologically and technologically, and it makes sense to strictly separate concerns in the architecture of MMt tools. For example, in [6] we argue that schema and data integration algorithms mixing match and merge become unnecessarily complicated; the same is true for model merge tools. In [10], we show that the green vs. blue divide appears in other MMt scenarios, e.g., model transformation, and in [7] we argue that mixing green and blue leads to the inflexible architecture of model synchronization tools.

Modeling Concurrency. LTSs are considered to be well suited for modeling sequential behavior and choice, while their expressiveness for modeling concurrent behavior is limited and reduces concurrency to interleaving. Our categorical elaboration of LTSs brings new ideas to the subject. First, we distinguish between *external* concurrency via parallel composition of LTSs, and *internal* concurrency modeled with AND-compositions of transitions within the same LTS. In either case, commutativity is fundamental. Second, our notions of internal OR/AND compositions of transitions modify the understanding of what an LTS is. An LTS appears to be a signature of basic transitions from which different behaviors can be composed by applying OR and AND operations. In particular, an LTS enriched over the category $\mathsf{Pos}^{\times}$ (posets with GLBs) allows the concurrent execution of any finite set of transitions, but a more practical notion would be an LTS with only partially defined AND-composition. Applications of enriched LTSs for modeling true concurrency can be interesting future work.

Related Work. An LTS is a classical behaviour model defined as a transition relation $T \subset S{\times}L{\times}S$ over sets S of states and L of labels. Adding to the formalism composition of labels, thus making L a monoid or even a category, was considered by several authors: references and discussion can be found in [27]. Categorization of LTSs in these works is mainly motivated by mathematical reasons of unification and elegance. In contrast, we motivate categorical constructs by practical MMt scenarios and carefully discuss adequacy of our mathematical models. We are not aware of viewing commutativity between sequences of transitions as a major construct for the interleaving model of concurrency.

The notion of order-enriched LTS fits in the general paradigm of STS (*structured transition systems*) [5] stemming from an influential paper "Petri nets are monoids" (PNAM) [23], but there is an essential difference. For PNAM, applying AND to transitions assumes applying AND to their source and target states (to buy a one-dollar apple and a one-dollar cake, you need two dollars, and eating both snacks results in two smiles). The corresponding categorical structure is a monoidal rather than enriched category. The PNAM ideas were generalized in STSs by adding two different (but related) superstructures to LTSs: one for states and one for transitions, and the former is typically not poorer than the latter. In contrast, we do not assume any superstructure for states (other than that given by labeling). Also, the PNAM approach does not consider the OR-monoidal structure, which for LTSs is even more fundamental than AND. Categories combining AND- and OR-monoidal structures [4] were studied as categorical models of linear logic—a resource-sensitive version of propositional logic. Our order-enriched LTSs are not resource-sensitive.

Using colimits for modeling various operations of "putting widgets together" can be traced back to Goguen's pioneering work [13], and has often been used in computer science, databases, and ontology engineering; annotated references relevant to MDE can be found in [3,11]. Our use of colimits for merging behavioral models with complex correspondences via derived transitions is novel (usually more complicated methods are employed for such cases, see, e.g., [25]); its relation to semantic merge developed in [3] needs further research.

Using limits for synchronized parallel composition is less well known [15]; the closest to our setting is, probably, in [12], where they use limits for parallel composition of alphabets seen as pointed sets. But for us, an alphabet is a category, an LTS is a functor into this category, and we compose both alphabets (as types) and their "instances" (states and transitions). We are also not aware of the explicitly stated relationships between Model Management and Process Algebra: Merge/colimit is Choice, while Synchronization/limit is Parallel composition.

6 Conclusion

We have analyzed merge and parallel merge of LTSs. The main observation is that the merge procedures as such can be relatively simple (and formalized by the categorical operations of (co)limit), if correspondences between LTSs are properly specified using *derived* transitions. Operations needed for transition derivation are (i) adding idle-loop transitions for all states, (ii) sequential composition of transitions, and (iii) their OR- and AND-parallel composition. Adding (i) and (ii) to the LTS formalisms makes LTSs (mappings between) categories, and adding (iii) enriches them with a 2-arrow structure.

Although in the paper we only considered one class of models, LTSs, constructs we used should be applicable in a much wider context. Indeed, LTSs can be seen as a typical behavioral model and a benchmark for behavior modeling. As for structural modeling, we show in [8] that ideas and constructs we have considered in the paper are directly applicable for structural modeling with class diagrams as well. After all, the very nature of category theory, designed for a proper unification and generalization of different mathematical structures, facilitates "technology transfer" from LTSs to broad model universes.

Acknowledgments. We are grateful to anonymous reviewers for useful comments and suggestions. Financial support was provided by the Automotive Partnership Canada via sponsoring the NECSIS project.

References

1. Barr, M., Wells, C.: Category theory for computing science. Prentice Hall (1995)
2. Bernstein, P.A.: Applying model management to classical meta data problems. In: CIDR (2003)
3. Chechik, M., Nejati, S., Sabetzadeh, M.: A relationship-based approach to model integration. ISSE **8**(1), 3–18 (2012)
4. Cockett, J.R.B., Koslowski, J., Seely, R.A.G.: Introduction to linear bicategories. Mathematical Structures in Computer Science **10**(2), 165–203 (2000)
5. Corradini, A., Montanari, U.: An algebraic semantics for structured transition systems and its applications to logic programs. Theor. Comput. Sci. **103**(1), 51–106 (1992)
6. Diskin, Z., Easterbrook, S., Miller, R.: Integrating schema integration frameworks, algebraically. Tech. Rep. CSRG-583, University of Toronto (2008) http://ftp.cs.toronto.edu/pub/reports/csrg/583/TR-583-schemaIntegr.pdf
7. Diskin, Z., Xiong, Y., Czarnecki, K., Ehrig, H., Hermann, F., Orejas, F.: From State- to Delta-Based Bidirectional Model Transformations: The Symmetric Case. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 304–318. Springer, Heidelberg (2011)
8. Diskin, Z.: Towards category theory foundations for model management. Tech. Rep. GSDLab-TR 2014–03-03, University of Waterloo (2014). http://gsd.uwaterloo.ca/node/566
9. Diskin, Z., Gholizadeh, H., Wider, A., Czarnecki, K.: A Three-Dimensional Taxonomy for Bidirectional Model Synchronization. J. of Systems and Software (2015), to appear
10. Diskin, Z., Kokaly, S., Maibaum, T.: Mapping-aware megamodeling: design patterns and laws. In: Erwig, M., Paige, R.F., Van Wyk, E. (eds.) SLE 2013. LNCS, vol. 8225, pp. 322–343. Springer, Heidelberg (2013)
11. Diskin, Z., Xiong, Y., Czarnecki, K.: Specifying overlaps of heterogeneous models for global consistency checking. In: Dingel, J., Solberg, A. (eds.) MODELS 2010. LNCS, vol. 6627, pp. 165–179. Springer, Heidelberg (2011)
12. Fiadeiro, J.L., Costa, J.F., Sernadas, A., Maibaum, T.S.E.: Process semantics of temporal logic specifications. In: Bidoit, M., Choppy, C. (eds.) COMPASS/ADT. Lecture Notes in Computer Science, vol. 655, pp. 236–253. Springer, Heidelberg (1991)
13. Goguen, J.A.: A categorical manifesto. Mathematical Structures in Computer Science **1**(1), 49–67 (1991)
14. Golas, U., Lambers, L., Ehrig, H., Giese, H.: Toward bridging the gap between formal foundations and current practice for triple graph grammars. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) ICGT 2012. LNCS, vol. 7562, pp. 141–155. Springer, Heidelberg (2012)
15. Große-Rhode, M.: Semantic Integration of Heterogeneous Software Specifications. Monographs in Theoretical Computer Science. An EATCS Series, Springer (2004)
16. Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of mde in industry. In: ICSE, pp. 471–480. IEEE, ACM (2011)
17. Lambers, L., Hildebrandt, S., Giese, H., Orejas, F.: Attribute handling for bidirectional model transformations. ECEASST 49 (2012)
18. Lauder, M., Anjorin, A., Varró, G., Schürr, A.: Bidirectional model transformation with precedence triple graph grammars. In: Vallecillo, A., Tolvanen, J.-P., Kindler, E., Störrle, H., Kolovos, D. (eds.) ECMFA 2012. LNCS, vol. 7349, pp. 287–302. Springer, Heidelberg (2012)

19. Liang, H., Diskin, Z., Dingel, J., Posse, E.: A general approach for scenario integration. In: MODELS, pp. 204–218 (2008)
20. Magee, J., Kramer, J.: Concurrency: state models and Java programs. Wiley (1999)
21. Mantz, F., Taentzer, G., Lamo, Y.: Well-formed model co-evolution with customizable model migration. ECEASST 58 (2013)
22. Marchand, J., Combemale, B., Baudry, B.: A categorical model of model merging. In: Modeling in Software Engineering, pp. 70–76. MISE, ICSE Workshop (2012)
23. Meseguer, J., Montanari, U.: Petri nets are monoids. Inf. Comput. **88**(2), 105–155 (1990)
24. Muller, P., Fondement, F., Baudry, B., Combemale, B.: Modeling modeling modeling. Software and System Modeling **11**(3), 347–359 (2012)
25. Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S.M., Zave, P.: Matching and merging of statecharts specifications. In: ICSE (2007)
26. Sabetzadeh, M., Easterbrook, S.: An algebraic framework for merging incomplete and inconsistent views. In: 13th Int. Conference on Requirement Engineering (2005)
27. Sobociński, P.: Relational presheaves as labelled transition systems. In: Pattinson, D., Schröder, L. (eds.) CMCS 2012. LNCS, vol. 7399, pp. 40–50. Springer, Heidelberg (2012)