Guarded Transitions in Evolving Specifications

Dusko Pavlovic* and Douglas R. Smith**

Kestrel Institute, Palo Alto, California 94304 USA

Abstract. We represent state machines in the category of specifications, where assignment statements correspond exactly to interpretations between theories [7, 8]. However, the guards on an assignment require a special construction. In this paper we raise guards to the same level as assignments by treating each as a distinct category over a shared set of objects. A guarded assignment is represented as a pair of arrows, a guard arrow and an assignment arrow. We give a general construction for combining arrows over a factorization system, and show its specialization to the category of specifications. This construction allows us to define the fine structure of state machine morphisms with respect to guards. Guards define the flow of control in a computation, and how they may be translated under refinement is central to the formal treatment of safety, liveness, concurrency, and determinism.

1 Introduction

In previous work [8] we introduced *Evolving Specifications* (abbreviated to especs) as a framework for specifying, composing and refining behavior. The point of such a framework is, at the very least, to help us cross the path from ideas to running code. Programming languages are designed to support us at the final sections of that path. On one hand, *especs are evolving specifications*: diagrams of specs, displaying how the conditions, satisfied by the variables of computation, change from state to state. On the other hand, *especs are specification-carrying programs*: pieces of code, given with some global requirements and invariants, as well as annotated with some local conditions, *state descriptions*, satisfied at some states of computation and not at others. They can be construed as formalized comments, or Floyd-Hoare annotations, but made into the first-class citizens of code, i.e. available at runtime.

While such global and local specifications of the intent of computation are *hard to reconstruct* if the design records have been lost or thrown away, they are *easy to verify* if the design records are carried with the code.

* Supported from the DARPA project "Specification-Carrying Software", contract number F30602-00-C-0209, and the ONR project "Game Theoretic Framework for Reasoning about Security", contract number N00014-01-C-0454.

** Supported from the DARPA project "Specification-Carrying Software" contract number F30602-00-C-0209.

1.1 State machines and algebraic specifications

Originally, state machines were introduced and studied (by Turing, Moore, Mealy, and many others) as abstract, mathematical models of computers. More recently, though, software engineering tasks reached the levels where *practical* reasoning in terms of state machines has become indispensable: designing reactive, hybrid, embedded systems seems unthinkable without the various state modeling tools and languages, like Esterel, or Statecharts. Verifying high assurance systems by model checking is based on such state machine models. Moreover, one could argue that the whole discipline of object oriented programming is essentially a method for efficient management of state in software constructs.

However, there seems to be a conceptual gap between state machines as theoretical versus practical devices. A notable effort towards bridging this gap are Gurevich's Abstract State Machines [5]: on one hand, they are a foundational paradigm of computation, explicitly compared with Turing machines; on the other hand, they have been used to present practically useful programming languages, capturing semantical features of C, Java, and others. However, the absence of powerful typing and structuring (abstraction, encapsulation, composition...) mechanisms makes them unsuitable for the development and management of large software systems.

We wish to investigate a representation of state machines in a framework for large-scale software specification development ("from-specs-to-code"). Previous work at Kestrel Institute has implemented the Specware/Designware framework for the development of functional programs that is based on a category of higher-order logical specifications, composition by colimit, and refinement by diagram morphisms [11, 12]. The current work builds on and extends this framework with behavioral specifications (especs), representing state machines as diagrams of specifications, and again using composition by colimit and refinement by diagram morphism. Related approaches to representing behavior in terms of a category of specifications include [2, 6].

The goal is to build a *practical* software development tool, geared towards large, complex systems, with reactive, distributed, hybrid, embedded features, and with high assurance, performance, reliability, or security requirements, all on a clean and simple semantical foundation.

1.2 Evolving Specifications

There are four key ideas underlying our representation of state machines as evolving specifications (especs). Together they reveal an intimate connection between behavior and the category of logical specifications. The first three are due to Gurevich [5].

1. *A state is a model* – A state of computation can be viewed as a snapshot of the abstract computer performing the computation. The state has a set of named stores with values that have certain properties.
2. *A state transition is a finite model change* – A transition rewrites the stored values in the state.

3. *An abstract state is a theory* – Not all properties of a state are relevant, and it is common to group states into abstract states that are models of a theory. The theory presents the structure (sorts, variables, operations), plus the axioms that describe common properties (i.e. invariants). We can treat states as static, mathematical models of a global theory thy_A, and then all transitions correspond to model morphisms. Extensions of the global theory thy_A provide local theories for more refined abstract states, introducing local variables and local properties/invariants.
4. *An abstract transition is an interpretation between theories* – Just as we abstractly describe a class of states/models as a theory, we abstractly describe a class of transitions as an interpretation between theories [7, 8]. To see this, consider the correctness of an assignment statement relative to a precondition P and a postcondition Q; i.e. a Hoare triple $P\ \{x\ :=\ e\}\ Q$. If we consider the initial and final states as characterized by theories thy_{pre} and thy_{post} with theorems P and Q respectively, then the triple is valid iff $Q[e/x]$ is a theorem in thy_{pre}. That is, the triple is valid iff the symbol map $\{\ x\ \mapsto\ e\}$ is an interpretation from thy_{post} to thy_{pre}. Note that interpretation goes in the *opposite* direction from the state transition.

The basic idea of especs is to use specifications (finite presentations of a theory) as state descriptions, and to use interpretations to represent transitions between state descriptions.

The idea that abstract states and abstract transitions correspond to specs and interpretations suggests that state machines are diagrams over Spec^{op}. Furthermore, state machines are composed via colimits, and state machines are refined via diagram morphisms [8].

1.3 Guards as Arrows

What's missing from the picture above is the treatment of guards on transitions. Interpretations between theories correspond exactly to (parallel) assignment statements, so something extra is needed to capture guards in this framework.

Let K and L be two states and $K \xrightarrow{g \vdash a} L$ a transition, consisting of the guard, *viz* predicate g, and the update a. Intuitively, it can be understood as the command `if` g `then` a, executed at the state K, and leading into L by a — whenever the guard condition g is satisfied. More precisely, it is executed in two steps:

- at the state K, the condition g is evaluated;
- if it is satisfied, the update a is performed.

Every guarded update $K \xrightarrow{g \vdash a} L$ thus factors in the form

$$(g \vdash a) = (g \vdash \mathrm{id}) \cdot (\top \vdash a)$$

where $g \vdash \mathrm{id}$ is a guard with a trivial update (with the identity id mapping all symbols of K to themselves), whereas $\top \vdash a$ is the unguarded update (with the condition $\top$ always satisfied).

Going a step further, to compose two guarded commands

$$(g_1 \vdash a_1) \cdot (g_2 \vdash a_2) = (g_1 \vdash \mathrm{id}) \cdot (\top \vdash a_1) \cdot (g_2 \vdash \mathrm{id}) \cdot (\top \vdash a_2)$$

suggests the need for a switching operator to interchange the positions of a_1 and g_2 to obtain

$$(g_1 \vdash \mathrm{id}) \cdot (g_2' \vdash \mathrm{id}) \cdot (\top \vdash a_1') \cdot (\top \vdash a_2) = g_1 \cdot g_2' \vdash a_1' \cdot a_2$$

This gives rise to the following

Task: Given two classes of morphisms $\mathbb{G}$ and $\mathbb{A}$ over the same class of objects $\mathcal{S}$, construct the category $\mathcal{S}_{\mathbb{G}\vdash\mathbb{A}}$, where the morphisms will be the composites $g \vdash a$ of the elements of the two classes, which will be recovered as components of a factorization system.

The remainder of the paper introduces a general mathematical construction for $\mathcal{S}_{\mathbb{G}\vdash\mathbb{A}}$, and shows how to treat guarded transitions as a special case.

2 Construction

2.1 Simple form

Let $\mathbb{G}$ and $\mathbb{A}$ be two categories over the same class of objects $\mathcal{S}$. We want to form the category $\mathcal{S}_\vdash = \mathcal{S}_{\mathbb{G}\vdash\mathbb{A}}$ with a factorization system where the $\mathbb{G}$-arrows will be the abstract epis and the $\mathbb{A}$-arrows the abstract monics. This means that $\mathbb{G}$ and $\mathbb{A}$ will appear as lluf subcategories of $\mathcal{S}_\vdash$, and every $\mathcal{S}_\vdash$-morphism will factorize as a composite of an $\mathbb{G}$-morphism followed by an $\mathbb{A}$-morphism, orthogonal to each other in the usual sense [1, 3].

The requirements induce the definition

$$\begin{aligned} |\mathcal{S}_{\mathbb{G}\vdash\mathbb{A}}| &= \mathcal{S} \\ \mathcal{S}_{\mathbb{G}\vdash\mathbb{A}}(K, L) &= \sum_{X \in \mathcal{S}} \mathbb{G}(K, X) \times \mathbb{A}(X, L) \end{aligned}$$

Notation. An arrow $\mathcal{S}_\vdash$, which is a triple $\langle X, g, a\rangle$, will usually be written in the form $(g \vdash a)$. The components $g \in \mathbb{G}$ and $a \in \mathbb{A}$ will sometimes be called guard and action, respectively.

Conversely, given an arrow $f \in \mathcal{S}_\vdash$, we'll denote by $f^\bullet$ a representative of the $\mathbb{G}$-component, and by ${}^\bullet f$ a representative of the $\mathbb{A}$-component. In summary,

$$\begin{aligned} f = (g \vdash a) &\text{ means that} \\ f^\bullet = g &\text{ and} \\ {}^\bullet f = a & \end{aligned}$$

Towards the definition of the composition

$$\cdot : \mathcal{S}_{\vdash}(A,B) \times \mathcal{S}_{\vdash}(B,C) \longrightarrow \mathcal{S}_{\vdash}(A,C)$$

note that its naturality with respect to the pre-composition in $\mathbb{G}$ and the post-composition in $\mathbb{A}$ means that it extends the composition in these two categories, i.e.

$$g' \cdot (g \vdash a) = (g' \cdot g) \vdash a$$
$$(g \vdash a) \cdot a' = g \vdash (a \cdot a')$$

For the moment, these equations can be viewed as notational conventions. Later, when the composition is defined, they will be derivable as properties. Similarly, the left-hand side of

$$g' \cdot f \cdot a' = (g' \cdot f^{\bullet}) \vdash \ ({}^{\bullet}f \cdot a') \tag{1}$$

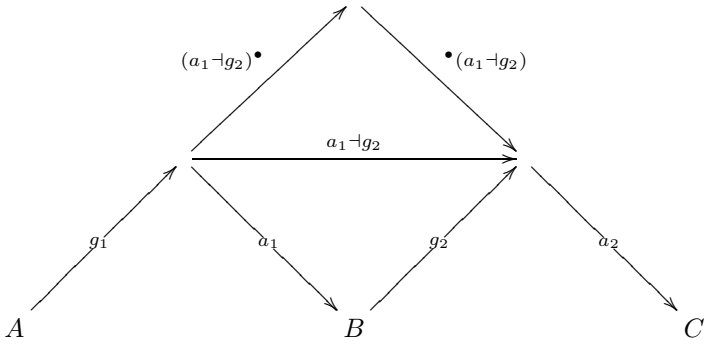
is just a convenient abbreviation of the right-hand side.

The composition can now be defined in the form

$$(g_1 \vdash a_1) \cdot (g_2 \vdash a_2) = g_1 \cdot \ (a_1 \dashv g_2) \cdot a_2$$

which is the abbreviated form of

$$(g_1 \vdash a_1) \cdot (g_2 \vdash a_2) = g_1 \cdot \ (a_1 \dashv g_2)^{\bullet} \ \vdash \ {}^{\bullet}(a_1 \dashv g_2) \cdot a_2 \tag{2}$$

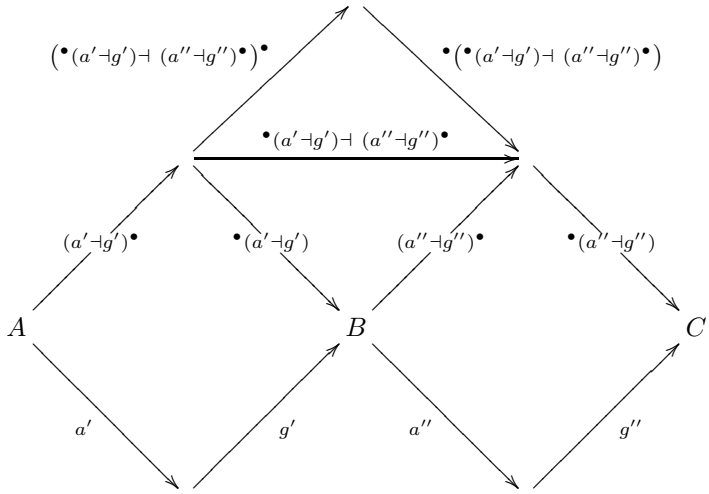


where the middle component comes from the family of *switching* functors

$$\dashv^{AB} : \textstyle\sum_{Z \in \mathcal{S}} \mathbb{A}(A,Z) \times \mathbb{G}(Z,B) \longrightarrow \mathcal{S}_{\vdash}(A,B)$$

natural for the pre-composition with the $\mathbb{A}$-morphisms to A and for the post-composition with the $\mathbb{G}$-morphisms out of B. Moreover, the switching is required to satisfy the following equations (simplified by (1))

$$a \dashv \mathrm{id} = \mathrm{id} \vdash a$$
$$\mathrm{id} \dashv g = g \vdash \mathrm{id}$$
$$a' \dashv g' \cdot (a'' \dashv g'')^{\bullet} = (a' \dashv g')^{\bullet} \ \cdot \ ({}^{\bullet}(a' \dashv g') \dashv (a'' \dashv g'')^{\bullet})$$
$${}^{\bullet}(a' \dashv g') \cdot a'' \dashv g'' = ({}^{\bullet}(a' \dashv g') \dashv (a'' \dashv g'')^{\bullet}) \ \cdot \ {}^{\bullet}(a'' \dashv g'')$$



The first two of these equations ensure that (id $\vdash$ id) play the role of the identities in $\mathcal{S}_\vdash$. The last two allow us to prove that the composition is associative.[1]

$$
\begin{aligned}
((g_1 \vdash a_1)\cdot(g_2 \vdash a_2))\cdot(g_3 \vdash a_3) &\overset{(2)}{=} \Big(g_1 \cdot\ (a_1 \dashv g_2)^\bullet \ \vdash\ {}^\bullet(a_1 \dashv g_2)\cdot a_2\Big)\cdot(g_3 \vdash a_3)\\
&\overset{(2)}{=} \begin{cases} g_1 \cdot\ (a_1 \dashv g_2)^\bullet \cdot ({}^\bullet(a_1 \dashv g_2)\cdot a_2 \dashv g_3)^\bullet \\ \vdash\ {}^\bullet({}^\bullet(a_1 \dashv g_2)\cdot a_2 \dashv g_3)\cdot a_3) \end{cases}\\
&= \begin{cases} g_1 \cdot\ (a_1 \dashv g_2)^\bullet \cdot ({}^\bullet(a_1 \dashv g_2)\dashv (a_2 \dashv g_3)^\bullet)^\bullet \\ \vdash\ {}^\bullet({}^\bullet(a_1 \dashv g_2)\dashv (a_2 \dashv g_3)^\bullet)\cdot (a_2 \dashv g_3)^\bullet \cdot a_3) \end{cases}\\
&= \begin{cases} g_1 \cdot (a_1 \dashv g_2 \cdot (a_2 \dashv g_3)^\bullet)^\bullet \\ \vdash\ {}^\bullet(a_1 \dashv g_2 \cdot (a_2 \dashv g_3)^\bullet)\cdot (a_2 \dashv g_3)^\bullet \cdot a_3) \end{cases}\\
&= (g_1 \vdash a_1)\cdot\Big(g_2 \cdot (a_2 \dashv g_3)^\bullet \vdash\ {}^\bullet(a_2 \dashv g_3)\cdot a_3\Big)\\
&= (g_1 \vdash a_1)\cdot((g_2 \vdash a_2)\cdot(g_3 \vdash a_3))
\end{aligned}
$$

Proposition 1. *Given the categories $\mathbb{G}$ and $\mathbb{A}$ over the object class $\mathcal{S}$, the category $\mathcal{S}_\vdash$ is universal for*

- *categories $\mathcal{K}$, given with*
- *a factorization*[2] *$(\mathbb{E}, \mathbb{M})$, and*
- *the functors*

$$G : \mathbb{G} \to \mathbb{E} \hookrightarrow \mathcal{K} \text{ and}$$
$$A : \mathbb{A} \to \mathbb{M} \hookrightarrow \mathcal{K}$$

that coincide on the objects

[1] The abbreviated form of (2) does not seem particularly useful here.

[2] For simplicity, we are taking an inessentially weaker notion of factorization than e.g. [3], omitting the requirement that both families contain all isomorphisms. Respecting this requirement would amount to an additional step of saturating the families.

– *and such that for all composable actions a and guards g*

$$A(a) \cdot G(g) = G(a \dashv g)^{\bullet} \cdot A^{\bullet}(a \dashv g)$$

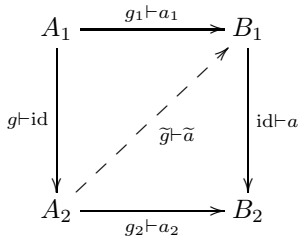
Proof. The category $\mathcal{S}_{\vdash}$ satisfies the above conditions. Indeed, the classes of arrows

$$\mathbb{E} = \{g \vdash \mathrm{id} \mid g \in \mathbb{G}\}$$
$$\mathbb{M} = \{\mathrm{id} \vdash a \mid a \in \mathbb{A}\}$$

form a factorization. This means that every $\mathcal{S}_{\vdash}$-arrow decomposes

$$(g \vdash a) = (g \vdash \mathrm{id}) \cdot (\mathrm{id} \vdash a)$$

The two families are orthogonal because every commutative square



can be filled with a unique diagonal, making both triangles commute. Indeed, the commutativity of the square means that

$$g_1 = g \cdot g_2 \text{ and}$$
$$a_1 \cdot a = a_2$$

so that we can simply take

$$\widetilde{g} = g_2 \text{ and}$$
$$\widetilde{a} = a_1$$

The switch functors are

$$(a \dashv g) = (g \vdash a)$$

so that the required condition holds

$$\begin{aligned} A(a) \cdot G(g) &= (\mathrm{id} \vdash a) \cdot (g \vdash \mathrm{id}) \\ &= (g \vdash a) \\ &= (a \dashv g) \\ &= G\,(a \dashv g)^{\bullet} \cdot A^{\bullet}\,(a \dashv g) \end{aligned}$$

The universality of $\mathcal{S}_{\vdash}$ now boils down to the fact that the functors $G : \mathbb{G} \to \mathbb{E} \hookrightarrow \mathcal{K}$ and $A : \mathbb{A} \to \mathbb{M} \hookrightarrow \mathcal{K}$ extend to a unique functor $H : \mathcal{S}_{\vdash} \longrightarrow \mathcal{K}$,

preserving the factorizations. The object part of this functor is already completely determined by the object parts of G and A, which coincide. Since the factorization is required to be preserved, the arrow part will be

$$H(g \vdash a) = G(g) \cdot A(a)$$

The functoriality follows from the assumptions:

$$\begin{aligned} H(g_1 \vdash a_1) \cdot H(g_2 \vdash a_2) &= G(g_1) \cdot A(a_1) \cdot G(g_2) \cdot A(a_2) \\ &= G(g_1) \cdot G(a_1 \dashv g_2)^\bullet \cdot A\,{}^\bullet(a_1 \dashv g_2) \cdot A(a_2) \\ &= G\left(g_1 \cdot (a_1 \dashv g_2)^\bullet\right) \cdot A\ \left({}^\bullet(a_1 \dashv g_2) \cdot a_2\right) \\ &= H\left((g_1 \vdash a_1) \cdot (g_2 \vdash a_2)\right) \end{aligned}$$

2.2 Examples

Adjoining a monoid of guards. Let $\langle \Gamma, \otimes, \top \rangle$ be a monoid, $\mathcal{C}$ an arbitrary category, and let

$$\mathcal{C} \times \Gamma \xrightarrow{\otimes} \mathcal{C}$$

be a monoid action, also denoted $\otimes$, by abuse of notation. For every $g \in \Gamma$ there is thus a functor $(-) \otimes g : \mathcal{C} \longrightarrow \mathcal{C}$, satisfying

$$\begin{aligned} A \otimes \top &= A \\ (A \otimes g_1) \otimes g_2 &= A \otimes (g_1 \otimes g_2) \end{aligned}$$

We want to adjoin the elements of Γ to $\mathcal{C}$ as abstract epis, while keeping the original $\mathcal{C}$-arrows as the abstract monics. So take $\mathbb{A} = \mathcal{C}$, and define the hom-sets of $\mathbb{G}$ by

$$\mathbb{G}(K, L) = \{g \in \Gamma \mid K \otimes g = L\}$$

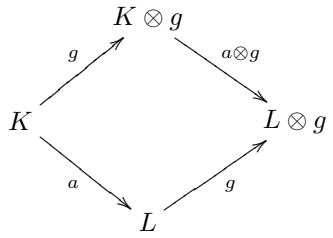
The composition is clearly induced from Γ: if $K \otimes g_1 = L$ and $L \otimes g_2 = M$, then $K \otimes (g_1 \otimes g_2) = M$ makes $g_1 \otimes g_2$ into an arrow from K to M.

The $\mathcal{C}_\vdash$ construction now becomes

$$\mathcal{C}_\vdash(K, L) = \sum_{g \in \Gamma} \mathcal{C}(K \otimes g, L)$$

The switch functors are

$$(a \dashv g) = (g \vdash a \otimes g)$$



so that the composition is

$$(g_1 \vdash a_1) \cdot (g_2 \vdash a_2) = g_1 \otimes g_2 \vdash (a_1 \otimes g_2) \cdot a_2$$

Objects as guards. An interesting special case of the above arises when $\mathcal{C}$ is a strict monoidal category[3]. Its object class $|\mathcal{C}|$ is then a monoid, which comes with the action

$$\mathcal{C} \times |\mathcal{C}| \xrightarrow{\otimes} \mathcal{C}$$

The category $\mathcal{C}_\vdash$ is formed as above, with the objects of $\mathcal{C}$ as the guards.

If $\mathcal{C}$ is the category $\mathcal{N}$ of von Neumann natural numbers, i.e. $n = \{0, 1, 2, \ldots, n-1\}$ with all functions as morphisms. This is, of course, equivalent to the category of finite sets, but the monoidal structures induced by the products, or the coproducts, can be made strict, using the arithmetic operations. The finiteness does not matter either, so the category $\mathcal{O}$ of all ordinals would do just as well, but looks bigger.

With the cartesian products as the monoidal structure, i.e. $\otimes = \times$ and $\top = 1$, a guarded morphism $(g \vdash a) : k \longrightarrow \ell$ is a function $a : k \times g \longrightarrow \ell$. The numbers g, k, ℓ can be thought of as arities; besides the inputs of arity k, the function a thus also requires the inputs from the guard g. Its composite with $(h \vdash b) : \ell \longrightarrow m$ just accumulates the arguments $x : g$ and $y : h$

$$(g \vdash a) \cdot (h \vdash b)(u, x, y) = a(b(u, x), y)$$

If the monoidal structure is the coproducts, i.e. $\otimes = +$ and $\top = 0$, then $(g \vdash a) : k \longrightarrow \ell$ is a pair of functions $[a_0, a_1] : k + g \longrightarrow \ell$. The composite with $(h \vdash b) : \ell \longrightarrow m$, i.e. $[b_0, b_1] : \ell + h \longrightarrow m$ is the triple

$$(g \vdash a) \cdot (h \vdash b) = [a_0 \cdot b_0, a_1 \cdot b_0, b_1]$$

On the other hand, if we use the monoid $\langle \mathcal{N}, +, 0 \rangle$, a guarded morphism $(g \vdash a) : k \longrightarrow \ell$ will be a function $a : k \longrightarrow \ell + g$, which besides the outputs of arity ℓ may yield some outputs of arity g.

2.3 Full construction

Some natural examples require a more general construction. For instance, starting from a category $\mathcal{S}$ with a factorization system $(\mathbb{E}, \mathbb{M})$ and taking $\mathbb{G} = \mathbb{E}$ and $\mathbb{A} = \mathbb{M}$, one would expect to get $\mathcal{S} \simeq \mathcal{S}_\vdash$. However, one gets a category where the morphisms are the *chosen* factorizations, which is essentially larger than the original one (it may not be locally small).

Moreover, looking back at the motivating example $\mathcal{S} = \mathsf{Spec}$, one might like to develop the guard category using the *specification morphisms*, rather than interpretations as the class of updates. This suggests the task of generating from

[3] In fact, premonoidal [9] is enough.

$\mathcal{S}$, $\mathbb{A}$ and $\mathbb{G}$ a category where the elements of a chosen class of morphisms $\mathbb{D}$ will boil down to isomorphisms. In the case when $\mathcal{S}$ is Spec, the class $\mathbb{D}$ are the definitional extension.

So suppose now that we are given a class $\mathcal{S}$ and three categories over it, $\mathbb{A}$, $\mathbb{D}$ and $\mathbb{G}$, such that $\mathbb{D}$ is a subcategory of both $\mathbb{A}$ and $\mathbb{G}$. So we have the faithful functors $\mathbb{D} \hookrightarrow \mathbb{A}$ and $\mathbb{D} \hookrightarrow \mathbb{G}$, both bijective on objects.

For every pair of objects $K, L \in \mathcal{S}$ there is a diagram

$$\begin{aligned} \mathbb{D} \times \mathbb{D}^{op} &\longrightarrow \mathsf{Set} \\ \langle X, Y \rangle &\longmapsto \mathbb{G}(K, X) \times \mathbb{A}(Y, L) \end{aligned}$$

The coend $\int_{X \in \mathbb{D}} \mathbb{G}(K, X) \times \mathbb{A}(X, L)$ of this diagram [10, IX.6] identifies elements of $\mathbb{G}(K, X) \times \mathbb{A}(X, L)$ and $\mathbb{G}(K, Y) \times \mathbb{A}(Y, L)$ along the spans

$$\begin{array}{ccc} & \mathbb{G}(K, X) \times \mathbb{A}(Y, L) & \\ \swarrow & & \searrow \\ \mathbb{G}(K, X) \times \mathbb{A}(X, L) & & \mathbb{G}(K, Y) \times \mathbb{A}(Y, L) \end{array}$$

induced by post-composing with $d \in \mathbb{D}(X, Y)$ in $\mathbb{G}$ and by pre-composing in $\mathbb{A}$.

The category $\mathcal{S}_{\vdash} = \mathcal{S}_{\mathbb{G} \vdash \mathbb{A}}^{\mathbb{D}}$ is now defined

$$\begin{aligned} |\mathcal{S}_{\vdash}| &= \mathcal{S} \\ \mathcal{S}_{\vdash}(K, L) &= \int_{X \in \mathbb{D}} \mathbb{G}(K, X) \times \mathbb{A}(X, L) \end{aligned}$$

If the $\mathbb{D}$-arrows support the right calculus of fractions in $\mathbb{G}$ and the left calculus of fractions in $\mathbb{A}$, the coends can be simplified by the usual equivalence relations [4, 1]. In any case, the composition follows from the universal property of the coends, and this general construction yields the following universal property of $\mathcal{S}_{\vdash}$.

Proposition 2. *Given the categories*

$$\begin{array}{ccc} & \mathbb{D} & \\ \swarrow & & \searrow \\ \mathbb{G} & & \mathbb{A} \end{array}$$

over the same object class $\mathcal{S}$, the category $\mathcal{S}_{\vdash}$ is universal for

- *categories $\mathcal{K}$, given with*
- *a factorization $(\mathbb{E}, \mathbb{M})$, and*
- *the functors*

$$\begin{aligned} &G : \mathbb{G} \to \mathbb{E} \hookrightarrow \mathcal{K} \text{ and} \\ &A : \mathbb{A} \to \mathbb{M} \hookrightarrow \mathcal{K} \\ &D : \mathbb{D} \to \mathbb{I} \hookrightarrow \mathcal{K} \end{aligned}$$

where $\mathbb{I}$ *is groupoid of isomorphisms of* $\mathcal{K}$*. These three functors coincide on the objects, and for all composable actions* a *and guards* g *holds*

– *satisfy*

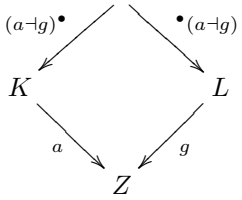
$$A(a) \cdot G(g) = G(a \dashv g)^{\bullet} \cdot A\,{}^{\bullet}(a \dashv g)$$

Examples

Decompositions. Trivially, the construction can be applied to $\mathbb{A} = \mathbb{G} = \mathbb{D} = \mathcal{C}$, for any category $\mathcal{C}$. The switch functors map a triple $\langle Z, f, g\rangle$ from $\sum_{X \in \mathcal{C}} \mathbb{A}(K, X) \times \mathbb{G}(X, L)$ to $f \cdot g$ in $\mathcal{C}(K, L) \cong \int_{X \in \mathbb{D}} \mathbb{G}(K, X) \times \mathbb{A}(X, L)$.

Note that ignoring $\mathbb{D}$ and applying the construction in the simple form, with the trivial switch operation, yields a bicategory, associative only up to arbitrary $\mathcal{C}$-morphisms.

Spans. If $\mathcal{C}$ is a category with pullbacks, we can take $\mathbb{G} = \mathcal{C}^{op}$, $\mathbb{A} = \mathcal{C}$ and $\mathbb{D} = \mathbb{I}$ the groupoid of all isomorphisms in $\mathcal{C}$. An abstract epi from K to L is now an ordinary $\mathcal{C}$-map from L to K. The switch functors can now be defined using pullbacks:



The above construction yields the category of matrices in $\mathcal{C}$, in contrast with the *bi*category of spans, obtained by the earlier simple construction. E.g., for $\mathcal{C} = \mathsf{Set}$, the morphisms of this category are ordinal matrices, with the matrix multiplication as the composition.

Opspans. Dually, when $\mathcal{C}$ is a category with pushouts, we can take $\mathbb{G} = \mathcal{C}$ and $\mathbb{A} = \mathcal{C}^{op}$, and get a "strictified" version of the bicategory of opspans as $\mathcal{C}_{\vdash}$. In contrast with the span composition, which boils down to the matrix composition, the opspan composition corresponds to the equivalence relation unions.

Factorization systems Given a category $\mathcal{C}$ with a factorization system $(\mathbb{E}, \mathbb{M})$, we can now recover $\mathcal{C} = \mathcal{C}_{\vdash}$ by taking $\mathbb{D}$ to be the isomorphism groupoid $\mathbb{I}$ of $\mathcal{C}$. This gives an equivalence of the 2-category of factorization systems, with the 2-category of the triples $\langle \mathbb{E}, \mathbb{M}, \mathbb{I} \rangle$.

3 Adjoining guards to Spec

The general construction can now be specialized to obtain a suitable category of guarded transitions in state machines. We start from the class $\mathcal{S} = |\mathsf{Spec}|$,

and want to build the category of guarded transitions from the categories $\mathbb{A}$ of unguarded transitions, and $\mathbb{G}$ of guards. The unguarded transitions will be the interpretations backwards, i.e.

$$\mathbb{A} = \mathsf{Spec}^{op}$$

But what to take as the guards? Probably the simplest idea is to take *all interpretations*:

$$\mathbb{G} = \mathsf{Spec}$$

With the switch functors induced by the pushouts, the resulting category of guarded interpretations is just the category of opspans in Spec. A guarded transition $K \xrightarrow{g \vdash f} L$ is thus just an opspan $K \xrightarrow{g} M \xleftarrow{f} L$ in Spec. The guard M is a separate spec, given with the interpretations g and f of K and L respectively, *and* possibly including some private variables, sorts and operations, equally invisible to K and to L.

As semantics of transitions, the opspans have the obvious shortcoming that they are completely symmetric: given an $K \xrightarrow{g} M \xleftarrow{f} L$, we have no way to tell whether it is a transition from K to L, or from L to K, since the guard M is symmetrically related, and can be equally unrelated to both. The opspan semantics is thus *not full*: not all opspans can be reasonably interpreted as transitions.

Intuitively, one might want to think of the guarded transition $K \xrightarrow{\Phi \vdash f} L$ as the command `if` Φ `then` f, executed at the state K, and leading into L by f — whenever the guard condition Φ is satisfied. The above symmetry is broken by the assumption that the guard Φ is evaluated at state K, and not L, or some third state. This is why, in general, such a transition is *irreversible*: once the state K is overwritten, it cannot in general be recovered, nor can Φ be re-evaluated.

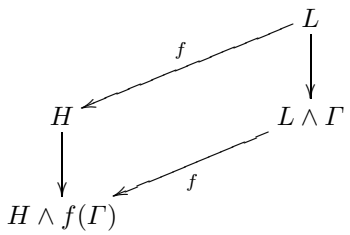
Following this intuition, that the guard Φ is evaluated at K, we restrict the opspan semantics by the requirement that Φ is expressible in the language $\mathcal{L}_K$ of K:

$$\mathbb{G}(K, M) = \{\Phi \in \mathcal{L}_K \mid K \wedge \Phi = M\}$$

where $K \wedge \Phi$ denotes the spec that extends K with the axiom Φ. Indeed, it seems reasonable to require that all variables of Φ must be known and determined at the state K, where this condition is evaluated, causing the transition to fire, or not.

The switching functors are given by

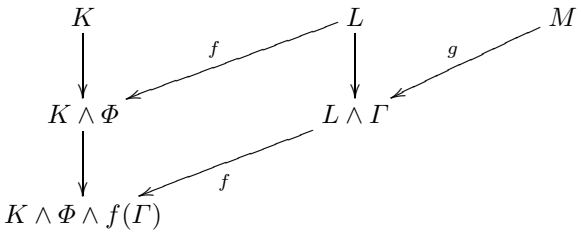
$$f \dashv \Gamma = f(\Gamma) \vdash f$$



so that arrow composition is given by the rule

$$\frac{K \xrightarrow{\Phi \vdash f} L \quad L \xrightarrow{\Gamma \vdash g} M}{K \xrightarrow{\Phi \wedge f(\Gamma) \vdash f \cdot g} M}$$

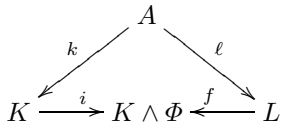
or diagrammatically



In summary, the category $\mathsf{Spec}_{\vdash}$ of specifications and guarded transitions will have specifications as its objects, while the hom-sets will be

$$\mathsf{Spec}_{\vdash}(K, L) = \{(\Phi \vdash f) \mid \Phi \in \mathcal{L}_K \text{ and } f \in \mathsf{Spec}(L, K \wedge \Phi)\}$$

In general, for a fixed spec A, we shall define $A/\mathsf{Spec}_{\vdash}$ to be the category with the specs inheriting A (*viz* the extensions $A \xrightarrow{k} K$) as objects[4], arrows $\Phi \vdash f$ have the form



and the hom-sets will be

$$A/\mathsf{Spec}_{\vdash}(k, \ell) = \{(\Phi \vdash f) \mid \Phi \in \mathcal{L}_K \text{ and } f \in \mathsf{Spec}(L, K \wedge \Phi) \text{ and } k \cdot i = \ell \cdot f\}$$

for $\mathbb{A} = A/\mathsf{Spec}$ and $\mathbb{G}(k, \ell) \;=\; \{\Phi \in \mathcal{L}_K \mid K \wedge \Phi = L\}$.

Remark. Note that the construction of $\mathsf{Spec}_{\vdash}$ is just a fibered form of adjoining objects as guards: for each language Σ the semilattice $\langle|\mathsf{Spec}_{\Sigma}|, \wedge, \top\rangle$ of all theories over Σ is adjoined to the fiber Spec_{Σ} as the monoid of guards.

[4] The state descriptions of a machine are not unrelated specs, but they share/inherit the common global spec, capturing whatever may be known about the global invariants and the intent of the program. The abstract states K and L thus come with the interpretations $k : A \longrightarrow K$ and $\ell : A \longrightarrow L$ of the globally visible signature and the invariants specified in A. The universe from which the states are drawn is thus the category A/Spec, of all specs inheriting A, rather than just Spec. While Spec is fibered over Lang by the functor mapping each spec K to its language $\mathcal{L}_K$, the category A/Spec is fibered over $\mathcal{L}_A/Lang$, mapping each interpretation $A \longrightarrow K$ to the underlying language translation.

The construction $A/\mathsf{Spec}_{\vdash}$ is, on the other hand, a fibered form of a slightly more general construction. The category A/Spec is fibered over the category $\mathcal{L}_A/\mathsf{Lang}$ of languages interpreting $\mathcal{L}_A$. But this time, each fiber $(A/\mathsf{Spec})_\sigma$ over $\sigma : \mathcal{L}_A \longrightarrow \Sigma$ is assigned not its own monoid of objects, but again just the semilattice of theories over Σ.

4 Concluding Remarks

One remarkable fact arising from the above presentation is that an elementary question arising from a basic computational model — the question of guards in evolving specifications — already leads to an apparently novel categorical construction, which may be of independant mathematical interest. The general construction, outlined in Section 2.3, corresponds to a *double* calculus of fractions, combining left and right fractions in a common framework, with new universal properties. The details and the instances of this new construction, which appears to lead to some interesting groups and algebras, remain to be investigated in a separate paper.

On the espec side, from which it originates, the construction now guides the development of a suitable category for guarded actions, allowing us to treat guarded transition systems (especs) as diagrams over Spec^{op}. In accord with the ideas and structures presented in [8], we can then compose especs using colimits, and we can refine systems of especs using diagram morphisms, providing a solid mathematical background for interleaving and combining bottom-up and top-down software development in a unified framework.

The conditions under which guards are mapped under refinement directly affect key behavioral properties such as nondeterminism, safety, and liveness. These will be explored in a subsequent paper. Other future directions for this work include adding timing constraints and exploring resource-bounded computation, and modeling hybrid embedded systems with especs.

References

1. Borceux, F. *Handbook of Categorical Algebra 1: Basic Category Theory*, vol. 50 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, Cambridge, 1994.
2. Errington, L. Notes on diagrams and state. Tech. rep., Kestrel Institute, 2000.
3. Freyd, P., and Kelly, G. M. Categories of continuous functors I. *Journal of Pure and Applied Algebra 2*, 3 (1972), 169–191.
4. Gabriel, P., and Zisman, M. *Calculus of Fractions and Homotopy Theory*, vol. 36 of *Ergebnisse der Mathematik und ihrer Grenzgebiete. New Series.* Springer-Verlag, New York, 1967.
5. Gurevich, Y. Evolving algebra 1993: Lipari guide. In *Specification and Validation Methods*, E. Boerger, Ed. Oxford University Press, 1995, pp. 9–36.
6. J.L.Fiadeiro, and T.Maibaum. Interconnecting formalisms: supporting modularity, reuse and incrementality. In *Proc. 3rd Symposium on the Foundations of Software Engineering* (1995), G. Kaiser, Ed., ACM Press, pp. 72–80.

7. Kutter, P. W. State transitions modeled as refinements. Tech. Rep. KES.U.96.6, Kestrel Institute, August 1996.
8. Pavlovic, D., and Smith, D. R. Composition and refinement of behavioral specifications. In *Proceedings of Automated Software Engineering Conference* (2001), IEEE Computer Society Press, pp. 157–165.
9. Power, J., and Robinson, E. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science 7*, 5 (1997), 453–468.
10. MacLane, S. *Categories for the Working Mathematician*, vol. 5 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 1971.
11. Smith, D. R. Mechanizing the development of software. In *Calculational System Design, Proceedings of the NATO Advanced Study Institute*, M. Broy and R. Steinbrueggen, Eds. IOS Press, Amsterdam, 1999, pp. 251–292.
12. Srinivas, Y. V., and Jüllig, R. Specware: Formal support for composing software. In *Proceedings of the Conference on Mathematics of Program Construction*, B. Moeller, Ed. LNCS 947, Springer-Verlag, Berlin, 1995, pp. 399–422.