Preface

Ulrike Golas
Konrad-Zuse-Zentrum für Informationstechnik Berlin, Germany
golas@zib.de

Thomas Soboll
Universität Salzburg, Austria
tsoboll@cosy.sbg.ac.at

Category Theory is a well-known powerful mathematical modeling language with a wide area of applications in mathematics and computer science, including especially the semantical foundations of topics in software science and development. Categorical methods are already well established for the semantical foundation of type theory (cartesian closed categories), data type specification frameworks (institutions) and graph transformation (adhesive high level replacement categories).

It is the intention of the ACCAT Workshops on Applied and Computational Category Theory to bring together leading researchers in these areas with those in software science and development in order to transfer categorical concepts and theories in both directions. The workshops aims to represent a forum for researchers and practitioners who are interested in an exchange of ideas, notions, and techniques for different applications of category theory.

The originators of the ACCAT workshops were Hartmut Ehrig (Berlin) and Jochen Pfalzgraf (Salzburg); they started with the 1st ACCAT workshop as a satellite event of ETAPS 2006, the European Joint Conferences on Theory and Practice of Software. Since then, the workshop has been a yearly meeting at the ETAPS conferences.

The seventh ACCAT workshop on Applied and Computational Category Theory 2012 was held in Tallinn, Estonia on the 1st of April 2012 as a satellite event of ETAPS 2012. Topics relevant to the scope of the workshop included (but were not restricted to)

- General Modeling Aspects
- Categorical, Algebraic, Geometric, Topological Modeling Aspects
- Petri Nets, Process Algebras, Activity Networks
- Logical Modeling, Unifying Frameworks
- Coalgebraic Methods in Systems Theory
- Generalized Automata, Categorical Methods

In contrast to previous ones, this workshop consisted of three invited talks as well as four peer-reviewed presentations. This issue contains the full version of one of the invited talks as well as the submitted papers, which cover a wide range of applications of category theory, from model-driven engineering over transition systems in stochastic processes to transformations in M-adhesive categories.

We are grateful to the external referees for their excellent work in reviewing and selecting the submitted papers. Moreover, we would like to thank the Organizing Committee of the ETAPS conference, and in particular the general chair, Tarmo Uustalu, and the workshop chair, Keiko Nakata, for their successful organization.

24th of July, 2012

Ulrike Golas
Thomas Soboll

© U. Golas & T. Soboll
This work is licensed under the
Creative Commons Attribution License.
Category Theory and Model-Driven Engineering: From Formal Semantics to Design Patterns and Beyond

Zinovy Diskin\textsuperscript{1,2} Tom Maibaum\textsuperscript{1}

\textsuperscript{1}Network for Engineering of Complex Software-Intensive Systems for Automotive Systems (NECSIS), McMaster University, Canada
\textsuperscript{2}Generative Software Development Lab, University of Waterloo, Canada
diskinz@mcmaster.ca tom@maibaum.org

There is a hidden intrigue in the title. CT is one of the most abstract mathematical disciplines, sometimes nicknamed "abstract nonsense". MDE is a recent trend in software development, industrially supported by standards, tools, and the status of a new "silver bullet". Surprisingly, categorical patterns turn out to be directly applicable to mathematical modeling of structures appearing in everyday MDE practice. Model merging, transformation, synchronization, and other important model management scenarios can be seen as executions of categorical specifications.

Moreover, the paper aims to elucidate a claim that relationships between CT and MDE are more complex and richer than is normally assumed for "applied mathematics". CT provides a toolbox of design patterns and structural principles of real practical value for MDE. We will present examples of how an elementary categorical arrangement of a model management scenario reveals deficiencies in the architecture of modern tools automating the scenario.

Keywords: Model-driven engineering, mathematical modeling, category theory

1 Introduction

There are several well established applications of category theory (CT) in theoretical computer science; typical examples are programming language semantics and concurrency. Modern software engineering (SE) seems to be an essentially different domain, not obviously suitable for theoretical foundations based on abstract algebra. Too much in this domain appears to be ad hoc and empirical, and the rapid progress of open source and collaborative software development, service-oriented programming, and cloud computing far outpaces their theoretical support. Model driven (software) engineering (MDE) conforms to this description as well: the diversity of modeling languages and techniques successfully resists all attempts to classify them in a precise mathematical way, and model transformations and operations — MDE’s heart and soul — are an area of a diverse experimental activity based on surprisingly weak (if any) semantic foundations.

In this paper we claim that theoretical underpinning of modern SE could (and actually quite naturally) be based on CT. The chasm between SE and CT can be bridged, and MDE appears as a “golden cut”, in which an abstract view of SE realities and concrete interpretations of categorical abstractions merge together: SE $\rightarrow$ MDE $\leftarrow$ CT. The left leg of the cospan is extensively discussed in the MDE literature (see \cite{47} and references therein); prerequisites and challenges for building the right leg are discussed in the present paper. Moreover, we aim to elucidate a claim that relationships between CT and MDE are more complex and richer than is normally assumed for ”applied mathematics". CT provides a toolbox of design patterns and principles, whose added value goes beyond such typical applications of mathematics to SE as formal semantics for a language, or formal analysis and model checking.
Two aspects of the CT-MDE “marriage” are discussed in the paper. The first one is a standard argument about the applicability of a particular mathematical theory to a particular engineering discipline. To wit, there is a mathematical framework called CT, there is an engineering domain called MDE, and we will try to justify the claim that they make a good match, in the sense that concepts developed in the former are applicable for mathematical modeling of constructs developed in the latter. What makes this standard argument exciting is that the mathematical framework in question is known to be notoriously abstract, while the engineering domain is very agile and seemingly not suitable for abstract treatment. Nevertheless, the argument lies within the boundaries of yet another instance of the evergreen story of applying mathematics to engineering problems. Below we will refer to this perspective on the issue as Aspect A.

The second perspective (Aspect B) is less standard and even more interesting. It is essentially based on specific properties of categorical mathematics and on the observation that software engineering is a special kind of engineering. To wit, CT is much more than a collection of mathematical notions and techniques: CT has changed the very way we build mathematical models and reason about them; it can be seen as a toolbox of structural design patterns and the guiding principles of their application. This view on CT is sometimes called arrow thinking. On the other hand, SE, in general, and MDE, in particular, essentially depend on proper structuring of the universe of discourse into subuniverses, which in their turn are further structured and so on, which finally results in tool architectures and code modularization. Our experience and attempts to understand complex structures used in MDE have convinced us that general ideas of arrow thinking, and general patterns and intuitions of what a healthy structure should be, turn out to be useful and beneficial for such practical concerns as tool architecture and software design.

The paper is structured as follows. In Section 2 we present two very general A-type arguments that CT provides a “right” mathematical framework for SE. The second argument also gives strong prerequisites for the B-side of our story. Section 3.1 gives a brief outline of MDE, and Section 3.2 reveals a truly categorical nature of the cornerstone notions of multimodeling and intermodeling (another A-argument). In Section 4 we present two examples of categorical arrangement of model management scenarios: model merge and bidirectional update propagation. This choice is motivated by our research interests and the possibility to demonstrate the B-side of our story. In Section 5 we discuss and exemplify three ways of applying CT for MDE: understanding, design patterns for specific problems, and general design guidance on the level of tool architecture.
today manifests itself as Programming-in-the-world. The latter is characterized by a large, and growing, heterogeneity of modules to be composed and methods for their composition, and such essentially zmmodifvit-in-the-large modern technologies as service orientation, open source and collaborative software development, and cloud computing.

The lower part of Fig. 1 presents this picture in a very schematic way as a path from 1 to $M$ to $M^n$ with $M$ referring to multiplicity in different forms, and degree $n$ indicating the modern tendencies of growth in heterogeneity and complexity.

MDE could be seen as a reaction to this development, a way of taming the growth of $n$ in a systematic way. Indeed, until recently, software engineers may feel that they could live without mathematical models: just build the software by whatever means available, check and debug it, and keep doing this throughout the software’s life. (Note that the situation in classical (mechanical and electrical) engineering is essentially different: debugging, say, a bridge, would be a costly procedure, and classical engineers abandoned this approach long time ago.) But this gift of easily built systems afforded to SEs is rapidly degrading as the costs of this process and the liability from getting it wrong are both growing at an enormous rate. By slightly rephrasing Dijkstra, we may say that precise modeling and specification become a matter of death and life rather than luxury.

These considerations give us the vertical axis in Fig. 2 skipping the intermediate point. The horizontal axis represents the evolution of mathematics in a similar simplified way. Point 1 corresponds to the modern mathematics of mathematical structures in the sense of Bourbaki: what matters is operations and relations over mathematical objects rather than their internal structure. Skipped point $M$ corresponds to basic category theory: the internal structure of the entire mathematical structure is encapsulated, and mathematical studies focus on operations and relations over structures considered as holistic entities. The multitude of higher degree, $M^n$, refers to categorical facilities for reflection: enrichment, internalization, higher dimensions, which can be applied $\textit{ad infinitum}$, hence, $\infty$-degree.

This (over-simplified) schema gives us four points of Math $\times$ SE interaction. Interaction (1,1) turned
Category Theory and Model Driven Engineering

out to be quite successful, as evidenced by such theory-based practical achievements as compilers, model checking, and relational DB theory. As for the point \((1, M^n)\), examining the literature shows that attempts at building theoretical foundations for MDE based on classical 1-mathematics were not successful. A major reason seems to be clear: 1-mathematics does not provide an adequate machinery for specifying and reasoning about inter-structural relationships and operations, which are at the very heart of modern software development. This point may also explain the general skepticism that a modern software engineer, and an academic teaching software engineering, feel about the practicality of using mathematics for modern software design: unfortunately, the only mathematics they know is the classical mathematics of Bourbaki and Tarski.

On the other hand, we view several recent applications of categorical methods to MDE problems \([2, 5, 17, 37, 21, 45, 44, 22, 19, 43, 46]\) as promising theoretical attempts, with great potential for practical application. It provides a firm plus for the \((M^\infty, M^n)\)-point in the plane.

Moreover, as emphasized by Lawvere, the strength of CT based modeling goes beyond modeling multi-structural aspects of the mathematical universe, and a categorical view of a single mathematical structure can be quite beneficial too. This makes point \((M^\infty, 1)\) in the plane potentially interesting, and indeed, several successful applications at this point are listed in the figure.

### 2.2 Mathematical modeling of engineering artifacts: Round-tripping abstraction vs. waterfall based abstraction

Figure Fig. 3(a) shows a typical way of building mathematical models for mechanical and electrical engineering domains. Meta-mathematics (the discipline of modeling mathematical models) is not practically needed for engineering as such. The situation dramatically changes for software engineering. Indeed, category theory (CT) could be defined as a discipline for studying mathematical structures: how to specify, relate and manipulate them, and how to reason about them. In this definition, one can safely remove the adjective “mathematical” and consider CT as a mathematical theory of structures in a very broad sense. Then CT becomes directly applicable to SE as shown in Fig. 3(b). Moreover, CT has actually changed the way of building mathematical structures and thinking about them, and found extensive and deep applications in theoretical computer science. Hence, CT can be considered as a common theoretical framework for all modeling stages in the chain (and be placed at the center). In this way, CT provides a remarkable unification for modeling activities in SE.

The circular, non linear nature of the figure also illustrates an important point about the role of CT in SE. Because software artifacts are conceptual rather than physical entities, there is potential for feedback between SE and Mathematics in a way that is not possible in traditional scientific and engineering disciplines. Design patterns employed in SE can be, and have been, influenced by mathematical model of software and the way we develop them.
3 MDE and CT: an overall sketch

We will begin with a rough general schema of the MDE approach to building software (Section 3.1), and then will arrange this schema in categorical terms (Section 3.2).

3.1 MDE in a nutshell

The upper-left corner of Fig. 4 shows a general goal of software design: building software that correctly interacts with different subsystems of the world (shown by figures of different shapes). For example, software embedded in a car interacts with its mechanical, electrical and electronic subsystems, with the driver and passengers, and with other cars on the road in future car designs. These components interact between themselves, which is schematically shown by overlaps of the respective shapes. The lower-right corner of Fig. 4 shows software modularized in parallel to the physical world it should interact with. The passage from the left to the right is highly non-trivial, and this is what makes SE larger and more challenging than mere programming. An effective means to facilitate the transition is to use models — a system of syntactical objects (as a rule, diagrammatic) that serve as abstractions of the “world entities” as shown in the figure (note the links from pieces of World to the respective parts of Modelware). These abstractions are gradually developed and refined until finally transformed into code. The modelware universe actually consists of a series of “modelwares” — systems of requirement, analysis, and design models, with each consecutive member in the list refining the previous one, and in its own turn encompassing several internal refinement chains. Modelware development consumes intelligence and time, but still easier and more natural for a human than writing code; the latter is generated automatically. The main idea of MDE is that human intelligence should be used for building models rather than code.

Of course, models have been used for building software long before the MDE vision appeared in the market. That time, however, after the first version of a software product had been released, its maintenance and further evolution had been conducted mainly through code, so that models had quickly become outdated, degraded and finally became useless. In contrast, MDE assumes that maintenance and evolution should also go through models. No doubts that some changes in the real world are much easier to incorporate immediately in the code rather than via models, but then MDE prescribes to update the models to keep them in sync with code. In fact, code becomes just a specific model, whose only essential distinction from other models in the modelware universe is its final position in the refinement chain. Thus, the Modelware boundary in Fig. 4 should be extended to encompass the Software region too.
3.2 Modelware categorically

Consider a modelware snapshot in Fig. 4. Notice that models as such are separated whereas their referents are overlapped, that is, interact between themselves. This interaction is a fundamental feature of the real world, and to make the model universe adequate to the world, intermodel correspondences/relations must be precisely specified. (For example, the figure shows three binary relations, and one ternary relation visualized as a ternary span with a diamond head.) With reasonable modeling techniques, intermodel relations should be compatible with model structures. The modelware universe then appears as a collection of structured objects and structure-compatible mappings between them, that is, as a categorical phenomenon. In more detail, a rough categorical arrangement could be as follows.

The base universe. Models are multi-sorted structures whose theories are called metamodels. The latter can be seen as generalized sketches [39][20], that is, pairs $M = (G_M, C_M)$ with $G_M$ a graph (or, more generally, an object of an apriori fixed presheaf topos $G$), and $C_M$ a set of constraints (i.e., diagram predicates) declared over $G_M$. An instance of metamodel $M$ is a pair $A = (G_A, t_A)$ with $G_A$ another graph (an object in $G$) and $t_A : G_A \rightarrow G_M$ a mapping (arrow in $G$) to be thought of as typing, which satisfy the constraints, $A \models C_M$ (see [20] for details). An instance mapping $A \rightarrow B$ is a graph mapping $f : G_A \rightarrow G_B$ commuting with typing: $f \circ t_B = t_A$. This defines a category $\text{Mod}(M) \subset G/G_M$ of $M$-instances.

To deal with the heterogeneous situation of models over different metamodels, we first introduce metamodel morphisms $m : M \rightarrow N$ as sketch morphisms, i.e., graph mappings $m : G_M \rightarrow G_N$ compatible with constraints. This gives us a category of metamodels $\text{MMod}$. Now we can merge all categories $\text{Mod}(M)$ into one category $\text{Mod}$, whose objects are instances (= $G$-arrows) $t_A : G_A \rightarrow G_M(A)$, $t_B : G_B \rightarrow G_M(B)$ etc., each having its metamodel, and morphisms $f : A \rightarrow B$ are pairs $f_{\text{data}} : G_A \rightarrow G_B$, $f_{\text{meta}} : M(A) \rightarrow M(B)$ such that $f_{\text{data}} \circ t_B = t_A ; f_{\text{meta}}$, i.e., commutative squares in $G$. Thus, $\text{Mod}$ is a sub-category of the arrow category $G^{-\rightarrow}$.

It can be shown that pulling back a legal instance $t_B : G_B \rightarrow G_N$ of metamodel $N$ along a sketch morphism $m : M \rightarrow N$ results in a legal instance of $M$ [20]. We thus have a fibration $p : \text{Mod} \rightarrow \text{MMod}$, whose Cartesian lifting is given by pullbacks.

Intermodel relations and queries. A typical intermodeling situation is when an element of one model corresponds to an element that is not immediately present in another model, but can be derived from other elements of that model by a suitable operation (a query, in the database jargon) [19]. Query facilities can be modeled by a pair of monads $(Q_{\text{def}}, Q)$ over categories $\text{MMod}$ and $\text{Mod}$, resp. The first monad describes the syntax (query definitions), and the second one provides the semantics (query execution).

A fundamental property of queries is that the original data are not affected: queries compute new data but do not change the original. Mathematical modeling of this property results in a number of equations, which can be summarized by saying that monad $Q$ is $p$-Cartesian, i.e., the Cartesian and the monad structure work in sync. If can be shown [19] that a query language $(Q, Q_{\text{def}})$ gives rise to a fibration $p_Q : \text{Mod} \rightarrow \text{MMod}_{\text{def}}$ between the corresponding Kleisli categories. These Kleisli categories have immediate practical interpretations. Morphisms in $\text{MMod}_{\text{def}}$ are nothing but view definitions: they map elements of the source metamodel to queries against the target one. Correspondingly, morphisms in $\text{Mod}_Q$ are view executions composed from query execution mechanism followed by retyping. The fact that projection $p_Q$ is fibration implies that the view execution mechanism is compositional: execution of a composed view equals the composition of executions.

Now a correspondence between models $A, B$ over metamodels $M, N$ can be specified by data shown in Fig. 5; these data consist of three components. (1) span $(m : N \leftrightarrow MN, n : MN \Rightarrow N)$ (whose legs are Kleisli mappings) specifies a common view $MN$ between the two metamodels. (2) trapezoids (arrows
in \(\text{Mod}_Q\) are produced by \(p_Q\)-Cartesian “lifting”, i.e., by executing views \(m\) and \(n\) for models \(A\) and \(B\) resp., which results in models \(A\mid_m\) and \(B\mid_n\) (here and below we use the following notation: computed nodes are not framed, and computed arrows are dashed). (3) span \((p: A\mid_m \leftarrow AB, q: AB \rightarrow B\mid_n)\) specifies a correspondence between the views. Note that this span is an independent modelware component and cannot be derived from models \(A, B\).

Spans like in Fig. 5 integrate a collection of models into a holistic system, which we will refer to as a multimodel. Examples, details, and a precise definition of a multimodel’s consistency can be found in [21].

It is tempting to encapsulate spans in Fig. 5 as composable arrows and work with the corresponding (bi)categories of metamodels and models. Unfortunately, it would not work out because, in general, Kleisli categories are not closed under pullbacks, and it is not clear how to compose Kleisli spans. It is an important problem to overcome this obstacle and find a workable approach to Kleisli spans.

Until the problem above is solved, our working universe is the Kleisli category of heterogeneous models fibred over the Kleisli category of metamodels. This universe is a carrier of different operations and predicates over models, and a stage on which different modeling scenarios are played. Classification and specification of these operations and predicates, and their understanding in conventional mathematical terms, is a major task of building mathematical foundations for MDE. Algebraic patterns appear here quite naturally, and then model management scenarios can be seen as algebraic terms composed from diagram-algebra operations over models and model mappings. The next section provides examples of such algebraic arrangements.

4 Model management (MMt) and algebra: Two examples

We will consider two examples of algebraic modeling of MMt scenarios. A simple one — model merging, and a more complex and challenging — bidirectional update propagation (BX).

4.1 Model merge via colimit

Merging several interrelated models without data redundancy and loss is an important MDE scenario. Models are merged (virtually rather than physically) to check their consistency, or to extract an integrated information about the system. A general schema is shown in Fig. 6. Consider first the case of several homogeneous models \(A, B, C, \ldots\) to be merged. The first step is to specify correspondences/relations between models via Kleisli spans \(R_1, R_2, \ldots\), or perhaps direct mappings like \(r_3\). The intuition of merging without data loss and redundancy (duplication of correspondent data) is precisely captured by the universal property of colimits, that is, it is reasonable to define merge as the colimit of a diagram of models and model mappings specifying intermodel correspondences.

\footnote{Note, however, that a proper categorical treatment of these operations in terms of universal constructions can be not straightforward.}
If models are heterogeneous, their relations are specified as in Fig. 5. To merge, we first merge metamodels modulo metamodel spans. Then we can consider all models and heads of the correspondence spans as instances of the merged metamodel, and merge models by taking the colimit of the entire diagram in the category of instances of the merged metamodel.

An important feature of viewing model merge as described above is a clear separation of two stages of the merge process: (i) discovery and specifying intermodel correspondences (often called model matching), and (ii) merging models modulo these correspondences. The first stage is inherently heuristic and context dependent. It can be assisted by tools based on AI-technologies, but in general a user input is required for final adjustment of the match (and of course to define the heuristics used by the tool). The second stage is pure algebra (colimit) and can be performed automatically. The first step may heavily depend on the domain and the application, while the second one is domain and application independent. However, a majority of model merge tools combine the two stages into a holistic merge algorithm, which first somehow relates models based on a specification of conflicts between them, and then proceeds accordingly to merging. Such an approach complicates merge algorithms, and makes a taxonomy of conflicts their crucial component; typical examples are [49, 42].

The cause of this deficiency is that tool builders rely on a very simple notion of model matching, which amounts to linking the-same-semantics elements in the models to be matched. However, as discussed above in Section 3.2, for an element e in model A, the-same-semantics B-element e′ can only be indirectly present in B, i.e., e′ can be derived from other elements of B with a suitable operation (query) over B rather than being an immediate element of B. With complex (Kleisli) matching that allows one to link basic elements in one model with derived elements in another model, the algebraic nature of merge as such (via the colimit operation) can be restored. Indeed, it is shown in [9] that all conflicts considered in [42] can be managed via complex matching, that is, described via Kleisli spans with a suitable choice of queries, afterwards merge is computed via colimit.

### 4.2 Bidirectional update propagation (BX)

Keeping a system of models mutually consistent (model synchronization) is vital for model-driven engineering. In a typical scenario, given a pair of inter-related models, changes in either of them are to be propagated to the other to restore consistency. This setting is often referred to as bidirectional model transformation (BX) [6].

#### 4.2.1 BX via tile algebra

A simple BX-scenario is presented in Fig. 7(a). Two models, A and B, are interrelated by some correspondence specification r (think of a span in a suitable category, or an object in a suitable comma category, see [27] for examples). We will often refer to them as horizontal deltas between models. In addition, there is a notion of delta consistency (extensionally, a class of consistent deltas), and if r is consistent, we call models A and B synchronized.

Now suppose that (the state of) model B has changed: the updated (state of the) model is B′, and arrow b denotes the correspondence between B and B′ (a vertical delta). The reader may think of a span, whose head consists of unchanged elements and the legs are injections so that B′’s elements beyond the
Suppose that we can re-align models $A$ and $B'$ and compute new horizontal delta $r \ast b$ (think of a span composition). If this new delta is not consistent, we need to update model $A$ so that the updated model $A'$ would be in sync with $B'$. More accurately, we are looking for an update $a: A \leftrightarrow A'$ such that the triple $(A', r', B')$ is consistent. Of course, we want to find a minimal update $a$ (with the biggest head) that does the job.

Unfortunately, in a majority of practically interesting situations, the minimality condition is not strong enough to provide uniqueness of $a$. To achieve uniqueness, some update propagation policy is to be chosen, and then we have an algebraic operation $\text{bPpg}$ (‘b’ stands for ‘backward’), which, from a given a pair of arrows $(b, r)$ connected as shown in the figure, computes another pair $(a, r')$ connected with $(b, r)$ as shown in the figure. Thus, a propagation policy is algebraically modeled by a diagram operation of arity specified by the upper square in Fig. 7(a): shaded elements denote the input data, whereas blank ones are the output. Analogously, choosing a forward update propagation policy (from the $A$-side to the $B$-side) provides a forward operation $\text{fPpg}$ as shown by the lower square.

The entire scenario is a composition of two operations: a part of the input for operation application $2: \text{fPpg}$ is provided by the output of $1: \text{bPpg}$. In general, composition of diagram operations, i.e., operations acting upon configurations of arrows (diagrams), amounts to their tiling, as shown in the figure; then complex synchronization scenarios become tiled structures. Details, precise definitions and examples can be found in [15].

Different diagram operations involved in model synchronization are not independent and their interaction must satisfy certain conditions. These conditions capture the semantics of synchronization procedures, and their understanding is important for the user of synchronization tools: it helps to avoid surprises when automatic synchronization steps in. Fortunately, principal conditions (synchronization laws) can be formulated as universally valid equations between diagrammatic terms — a tile algebra counterpart of universal algebraic identities. In this way BX becomes based on an algebraic theory: a signature of diagram operations and a number of equational laws they must satisfy. The appendix presents one such theory — the notion of a symmetric delta lens, which is currently an area of active research from both a practical and a theoretical perspective.

### 4.2.2 BX: delta-based vs. state-based

As mentioned above, understanding the semantics of model synchronization procedures is important, both theoretically and practically. Synchronization tools are normally built on some underlying algebraic theory [28, 53, 40, 4, 1, 41, 30], and many such tools (the first five amongst those cited above) use algebraic theories based on state-based rather than delta-based operations. The state-based version of the propagation scenario in Fig. 7(a) is described in Fig. 7(b). The backward propagation operation takes models $A, B, B'$, computes necessary relations between them ($r$ and $b$ on the adjacent diagram), and then
computes an updated model \( A' \). The two-chevron symbol reminds us that the operation actually consists of two stages: model alignment (computing \( r \) and \( b \)) and update propagation as such.

The state-based frameworks, although they may look simpler, actually hides several serious deficiencies. Model alignment is a difficult task that requires contextual information about models. It can be facilitated by intelligent AI-based tools, or even be automated, but the user should have an option to step in and administer corrections. In this sense, model alignment is similar to model matching preceding model merge. \(^2\) Weaving alignment (delta discovery) into update (delta) propagation essentially complicates the semantics of the latter, and correspondingly complicates the algebraic theory. In addition, the user does not have an access to alignment results and cannot correct them.

Two other serious problems of the state-based frameworks and architectures are related to operation composition. The scenario described in Fig. 7(a) assumes that the model correspondence (delta) used for update propagation \( 2: f_{Ppg} \) is the delta computed by operation \( 1: b_{Ppg} \); this is explicitly specified in the tile algebra specification of the scenario. In contrast, the state-based framework cannot capture this requirement. A similar problem appears when we sequentially compose a BX program synchronizing models A and B and another program synchronizing models B and C: composition amounts to horizontal composition of propagation operations as shown in Fig. 8, and again continuity, \( b_1 = b_2 \), cannot be specified in the state-based framework. A detailed discussion of delta- vs. state-based synchronization can be found in \([22,10]\).

### 4.2.3 Assembling model transformations

Suppose \( M, N \) are two metamodels, and we need to transform \( M \)-instances (models) into \( N \)-ones. Such a transformation makes sense if metamodels are somehow related, and we suppose that their relationship is specified by a span \((m: M \Leftarrow MN, n: MN \Rightarrow N)\) (Fig. 9), whose legs are Kleisli mappings of the respective query monad.

Now \( N \)-translation of an \( M \)-model \( A \) can be done in two steps. First, view \( m \) is executed (via its Cartesian lifting actually going down in the figure), and we obtain Kleisl arrow \( \overrightarrow{m_A}: A \Leftarrow R \) (with \( R = A \uparrow m \)). Next we need to find an \( N \)-model \( B \) such that its view along \( n, B \downarrow n \), is equal to \( R \). In other words, given a view, we are looking for a source providing this view. There are many such sources, and to achieve uniqueness, we need to choose some policy. Afterwards, we compute model \( B \) related to \( A \) by span \( (\overrightarrow{m_A}, \overrightarrow{n_B}) \).

If model \( A \) is updated to \( A' \), it is reasonable to compute a corresponding update \( b: B \leftrightarrow B' \) rather than recompute \( B' \) from scratch (recall that models can contain thousands elements). Computing \( b \) again consists of two steps shown in the figure.

Operations \( \text{Get}^m \) and \( \text{Put}^n \) are similar to \( f_{Ppg} \) and \( b_{Ppg} \) considered above, but work in the asymmetric situation when mappings \( m \) and \( n \) are total (Kleisl) functions and hence view \( R \) contains nothing new wrt. \( M \) and \( N \). Because of asymmetry, operations \( \text{Get} \) (‘get’ the view update) and \( \text{Put} \) (‘put’ it back to

\(^2\) A difference is that model matching usually refers to relating independently developed models, while models to be aligned are often connected by a given transformation.
the source) are different. Get\textsuperscript{m} is uniquely determined by the view definition \( m \). Put\textsuperscript{n} needs, in addition to \( n \), some update propagation policy. After the latter is chosen, we can realize transformation from \( M \) to \( N \) incrementally by composition \( fPpg = \text{Get}^m; \text{Put}^n \) — this is an imprecise linear notation for tiling (composition of diagram operations) specified in Fig. 9.

Note that the initial transformation from \( M \) to \( N \) sending, first, an \( M \)-instance \( A \) to its view \( R = A|_m \), and then finding an \( N \)-instance \( B \in N \) such that \( B|_n = R \), can be also captured by Get and Put. For this, we need to postulate initial objects \( \Omega_M \) and \( \Omega_N \) in categories of \( M \)- and \( N \)-instances, so that for any \( A \) over \( M \) and \( B \) over \( N \) there are unique updates \( 0_A : \Omega_M \to A \) and \( 0_B : \Omega_N \to B \). Moreover, there is a unique span \( (m_\Omega : \Omega_M \Leftarrow \Omega_{MN}, n_\Omega : \Omega_{MN} \Rightarrow \Omega_N) \) relating these initial objects. Now, given a model \( A \), model \( B \) can be computed as \( B' \) in Fig. 9 with the upper span being \( (m_\Omega, n_\Omega) \), and the update \( a \) being \( 0_A : \Omega_M \to A \).

The backward transformation is defined similarly by swapping the roles of \( m \) and \( n \):

\[ \text{bPpg} = \text{Get}^m; \text{Put}^n. \]

The schema described above can be seen as a general pattern for defining model transformation declaratively with all benefits (and all pains) of having a precise specification before the implementation is approached (and must obey). Moreover, this schema can provide some semantic guarantees in the following way. Within the tile algebra framework, laws for operations Get and Put, and their interaction (invertibility), can be precisely specified \cite{22} (see also the discussion in Section 5.1); algebras of this theory are called delta lenses. Then we can deduce the laws for the composed operations \( fPpg \) and \( bPpg \) from the delta lens laws. Also, operations \( \text{Get}^m, \text{Put}^n \) can themselves be composed from smaller blocks, if the view \( m \) is composed: \( m = m_1; m_2; \ldots; m_k \), via sequential lens composition. In this way, a complex model transformation is assembled from elementary transformation blocks, and its important semantic properties are guaranteed. More examples and details can be found in \cite{15}.

5 Applying CT to MDE: Examples and Discussion.

We will try to exemplify and discuss three ways in which CT can be applied in MDE. The first one — gaining a deeper understanding of an engineering problem — is standard, and appears as a particular instantiation of the general case of CT’s employment in applied domains. The other two are specific to SE: structural patterns provided by categorical models of the software system to be built can directly influence the design. We will use models of BX as our main benchmark; other examples will be also used when appropriate.

5.1 Deeper understanding. As mentioned in Sect. 4.2 stating algebraic laws that BX procedures must obey is practically important as it provides semantic guaranties for synchronization procedures. Moreover, formulation of these laws should be semantically transparent and concise as the user of synchronization tools needs a clear understanding of propagation semantics. The original state-based theory of asymmetric BX \cite{28} considered two groups of laws: invertibility (or round-tripping) laws, GetPut and PutGet, and history ignorance, PutPut. Two former laws say that two propagation operations, Get and Put, are mutually inverse. The PutPut law says that if a complex update is decomposed into consecutive pieces, it can be propagated incrementally, one piece after the other. A two-sorted algebra comprising two operations, Get and Put, satisfying the laws, is called a well-behaved lens.

Even an immediate arrow-based generalization of lenses to delta lenses (treated in elementary terms via tile algebra \cite{15,22}) revealed that the GetPut law is a simple law of identity propagation, IdPut, rather than of round-tripping. The benefits of renaming GetPut as IdPut are not exhausted by clarification of semantics: as soon as we understand that the original GetPut is about identity propagation, we at
once ask what the real round-tripping law GetPut should be, and at once see that operation Put is not the inverse of Get. We only have the weaker 1.5-round-tripping GetPutGet law (or weak invertibility; see the Appendix, where the laws in question are named IdPpg and fbfPpg and bfbPpg). It is interesting (and remarkable) that papers [14, 31], in which symmetric lenses are studied in the state-based setting, mistakenly consider identity propagation laws as round-tripping laws, and correspondingly analyze a rather poor BX-structure without real round-tripping laws at all.

The tile algebra formulation of the PutPut law clarified its meaning as a composition preservation law [15, 22], but did not solve the enigmatic PutPut problem. The point is that PutPut does not hold in numerous practically interesting situations, but its entire removal from the list of BX laws is also not satisfactory, as it leaves propagation procedures without any constraints on their compositionality. The problem was solved, or at least essentially advanced, by a truly categorical analysis performed by Michael Johnson et al [35, 34]. They have shown that an asymmetric well-behaved lens is an algebra for some KZ monad, and PutPut is nothing but the basic associativity condition for this algebra. Hence, as Johnson and Rosebrugh write in [34], the status of the PutPut changes from being (a) “some law that may have arisen from some special applications and should be discarded immediately if it seems not to apply in a new application” to (b) a basic requirement of an otherwise adequate and general mathematical model. And indeed, Johnson and Rosebrugh have found a weaker — monotonic — version of PutPut (see Fig. [13] in the Appendix), which holds in a majority of practical applications, including those where the original (non-monotonic or mixed) PutPut fails. Hopefully, this categorical analysis can be generalized for the symmetric lens case, thus stating solid mathematical foundations for BX.

5.2 Design patterns for specific problems.

Recalling Figure 3, Figure [10] presents a rough illustration of how mathematical models can reshape our view of a domain or construct $X$. Building a well-structured mathematical model $M$ of $X$, and then reinterpreting it back to $X$, can change our view of the latter as schematically shown in the figure with the reshaped construct $X'$. Note the discrepancy between the reshaped $X'$ and model $M$: the upper-left block is missing from $X'$. If $X$ is a piece of reality (think of mathematical modeling of physical phenomena), this discrepancy means, most probably, that the model is not adequate (or, perhaps, some piece of $X$ is not observable). If $X$ is a piece of software, the discrepancy may point to a deficiency of the design, which can be fixed by redesigning the software. Even better to base software design on a well-structured model from the very beginning. Then we say that model $M$ provides a design pattern for $X$.

We have found several such cases in our work with categorical modeling of MDE-constructs. For example, the notion of a jointly-monic n-ary arrow span turns out to be crucial for modeling associations between object classes, and their correct implementation as well [17]. It is interesting to observe how a simple arrow arrangement allows one to clean the UML metamodel and essentially simplify notation [12, 8]. Another example is modeling intermodel mappings by Kleisli morphisms, which provide a universal pattern for model matching (a.k.a alignment) and greatly simplify model merge as discussed in Sect. 4.1. In addition, the Kleisli view of model mappings provides a design pattern for mapping composition — a problem considered to be difficult in the model management literature [3]. Sequential
composition of symmetric delta lenses is also not evident; considering such lenses as algebras whose carriers are profunctors (see Appendix) suggests a precise pattern to be checked (this work is now in progress). Decomposition of a model transformation into Cartesian lifting (view execution) followed by the inverse operation of Cartesian lifting completion (view updating) as described in Section 4.2.3 provides a useful guidance for model transformation design, known to be laborious and error-prone. In particular, it immediately provides bidirectionality.

The graph transformation community also developed several general patterns applicable to MDE (with models considered as typed attributed graphs, see [23] for details). In particular, an industrial standard for model transformation, QVT [41], was essentially influenced by triple-graph grammars (TGGs). Some applications of TGGs to model synchronization (and further references) can be found in [30].

5.3 Diagrammatic modeling culture and tool architecture.

The design patterns mentioned above are based on the respective categorical machinery (monads, fibrations, profunctors). A software engineer not familiar with these patterns would hardly recognize them in the arrays of implementation details. Even less probable is that he will abstract away his implementation concerns and reinvent such patterns from scratch; distillation of these structures by the CT community took a good amount of time. In contrast, simple arrow diagrams, like in Fig. 7(a) (see also the Appendix), do not actually need any knowledge of CT: all that is required is making intermodel relations explicit, and denoting them by arcs (directed or undirected) connecting the respective objects. To a lesser extent, this also holds for the model transformation decomposition in Fig. 9 and the model merge pattern in Fig. 6. We refer to a lesser extent because the former pattern still needs familiarity with the relations-are-spans idea, and the latter needs an understanding of what colimit is (but, seemingly, it should be enough to understand it roughly as some algebraic procedure of “merging things”).

The importance of mappings between models/software artifacts is now well recognized in many communities within SE, and graphical notations have been employed in SE for a long time. Nevertheless, a majority of model management tools neglect the primary status of model mappings: in their architecture, model matching and alignment are hidden inside (implementations of) algebraic routines, thus complicating both semantics and implementation of the latter; concerns are intricately mixed rather than separated. As all SE textbooks and authorities claim separation of concerns to be a fundamental principle of software design, an evident violation of the principle in the cases mentioned above is an empirical fact that puzzles us. It is not clear why a BX-tool designer working on tool architecture does not consider simple arrow diagrams like in Fig. 7(a), and prefers discrete diagrams (b). The latter are, of course, simpler but their simplicity is deceiving in an almost evident way.

The only explanation we have found is that understanding the deceiving simplicity of discrete diagrams (b), and, simultaneously, manageability of arrow diagrams (a), needs a special diagrammatic modeling culture that a software engineer normally does not possess. This is the culture of elementary arrow thinking, which covers the most basic aspects of manipulating and using arrow diagrams. It appears that even elementary arrow thinking habits are not cultivated in the current SE curriculum, the corresponding high-level specification patterns are missing from the software designer toolkit, and software is often structured and modularized according to the implementation rather than specification concerns.
6 Related work

First applications of CT in computer science, and the general claim of CT’s extreme usefulness for computer applications should be, of course, attributed to Joseph Goguen [29]. The shift from modeling semantics of computation (behavior) to modeling structures of software programs is emphasized by José Fiadeiro in the introduction to his book [36], where he refers to a common “social” nature of both domains. The ideas put forward by Fiadeiro were directly derived from joint work with Tom Maibaum on what has become known as component based design and software architecture [26, 27, 24]. A clear visualization of these ideas by Fig. 2 (with M standing for Fiadeiro’s “social”) seems to be new. The idea of round-tripping modeling chain Fig. 3 appears to be novel, its origin can be traced to [11].

Don Batory makes an explicit call to using CT in MDE in his invited lecture for MoDELS’2008 [2], but he employs the very basic categorical means, in fact, arrow composition only. In our paper we refer to much more advanced categorical means: sketches, fibrations, Cartesian monads, Kleisli categories.

Generalized sketches (graphs with diagram predicates) as a universal syntactical machinery for formalizing different kinds of models were proposed by Diskin et al., [18]. Their application to special MDE problems can be found in [17, 38] and in the work of Rutle et al., see [46], [43] and references therein. A specific kind of sketches, ER-sketches, is employed for a number of problems in the database context by Johnson et al. [32]. Considering models as typed attributed graphs with applications to MDE has been extensively put forward by the graph transformation (GT) community [23]; their work is much more operationally oriented than our concerns in the present paper. On the other hand, in contrast to the generalized sketches framework, constraints seem to be not the first-class citizens in the GT world.

The shift from functorial to fibrational semantics for sketches to capture the metamodeling foundations of MDE was proposed in [13] and formalized in [20]. This semantics is heavily used in [15], and in the work of Rutle et al. mentioned above. Comparison of the two semantic approaches, functorial and fibrational, and the challenges of proving their equivalence, are discussed in [52].

The idea of modeling query languages by monads, and metamodel (or data schema) mappings by Kleisli mappings, within the functorial semantics approach, was proposed in [16], and independently by Johnson and Rosebrugh in their work on ER-sketches [32]. Reformulation of the idea for fibrational semantics was developed and used for specifying important MDE constructs in [15, 21]. An accurate formalization via Cartesian monads can be found in [19].

Algebraic foundations for BX is now an area of active research. Basics of the state-based algebraic framework (lenses) were developed by Pierce with coauthors [28]; their application to MDE is due to Stevens [50]. Delta-lenses [22, 10] is a step towards categorical foundations, but they have been described in elementary terms using tile algebra [15]. A categorical approach to the view update problem has been developed by Johnson and Rosebrugh et al. [33]; and extended to categorical foundations for lenses based on KZ-monads in [35, 34]. The notion of symmetric delta lens in Appendix is new; it results from incorporating the monotonic PutPut-law idea of Johnson and Rosebrugh into the earlier notion of symmetric delta lens [10]. Assembling synchronization procedures from elementary blocks is discussed in [15].

7 Conclusion

The paper claims that category theory is a good choice for building mathematical foundations for MDE. We first discuss two very general prerequisites that concepts and structures developed in category theory have to be well applicable for mathematical modeling of MDE-constructs. We then exemplify the argu-
ments by sketching several categorical models, which range from general definitions of multimodeling and intermodeling to important model management scenarios of model merge and bidirectional update propagation. We briefly explain (and refer to other work for relevant details) that these categorical models provide useful design patterns and guidance for several problems considered to be difficult.

Moreover, even an elementary arrow arrangement of model merge and BX scenarios makes explicit a deficiency of the modern tools automating these scenarios. To wit: these tools’ architecture weaves rather than separates such different concerns as (i) model matching and alignment based on heuristics and contextual information, and (ii) relatively simple algebraic routines of merging and update propagation. This weaving complicates both semantics and implementation of the algebraic procedures, does not allow the user to correct alignment if necessary, and makes tools much less flexible. It appears that even simple arrow patterns, and the corresponding structural decisions, may not be evident for a modern software engine.

Introduction of CT courses into the SE curriculum, especially in the MDE context, would be the most natural approach to the problem: even elementary CT studies should cultivate arrow thinking, develop habits of diagrammatic reasoning and build a specific intuition of what is a healthy vs. ill-formed structure. We believe that such intuition, and the structural lessons one can learn from CT, are of direct relevance for many practical problems in MDE.

Acknowledgement. Ideas presented in the paper have been discussed at NECSIS seminars at McMaster and the University of Waterloo; we are grateful to all their participants, and especially to Michał Antkiewicz and Krzysztof Czarnecki, for useful feedback and stimulating criticism. We have also benefited greatly from discussions with Don Batory, Steve Easterbrook, José Fiadeiro, Michael Healy, Michael Johnson, Ralf Lämmel, Bran Selic and Uwe Wolter. Thanks also go to anonymous referees for comments.

Financial support was provided with the NECSIS project funded by the Automotive Partnership Canada.

References


A Appendix. Algebra of bidirectional update propagation

In Section 4.2 we considered operations of update propagation, but did not specify any laws they must satisfy. Such laws are crucial for capturing semantics, and the present section aims to specify algebraic laws for BX. We will do it in an elementary way using tile algebra (rather than categorically — it is non-trivial and left for a future work). We will begin with the notion of an alignment framework to formalize delta composition (* in Section 4.2), and then proceed to algebraic structures modeling BX — symmetric delta lenses. (Note that the lenses we will introduce here are different from those defined in [10].)
A definition of an alignment framework is given by the following data.

(i) Two categories with pullbacks, \( A \) and \( B \), called model spaces. We will consider spans in these categories up to their equivalence via a head isomorphism commuting with legs. That is, we will work with equivalence classes of spans, and the term ‘span’ will refer to an equivalence class of spans. Then span composition (via pullbacks) is strictly associative, and we have categories (rather than bicategories) of spans, \( \text{Span}_1(A) \) and \( \text{Span}_1(B) \). Their subcategories consisting of spans with injective legs will be denoted by \( A^* \) and \( B^* \) resp.

Such spans are to be thought of as (model) updates. They will be depicted by vertical bi-directional arrows, for example, \( a \) and \( b \) in the diagrams Fig. 11(a). We will assume that the upper node of such an arrow is its formal source, and the lower one is the target; the source is the the original (state of the) model, and the target is the updated model. Thus, model evolution is directed down.

A span whose upper leg is identity (nothing deleted) is an insert update; it will be denoted by unidirectional arrows going down. Dually, a span with identity lower leg is a delete update; it will be denoted by a unidirectional arrow going up (but the formal source of such an arrow is still the upper node).

(ii) For any two objects, \( A \in A_0 \) and \( B \in B_0 \), there is a set \( R(A,B) \) of correspondences (or corrs in short) from \( A \) to \( B \). Elements of \( R(A,B) \) will be depicted by bi-directional horizontal arrows, whose formal source is \( A \) and the target is \( B \).

Updates and corrs will also be called vertical and horizontal deltas, resp.

(iii) Two diagram operations over corrs and updates called forward and backward (re)alignment. Their arities are shown in Fig. 11(a) (output arrows are dashed). We will also write \( a * r \) for \( fAln(a,r) \) and \( r * b \) for \( bAln(b,r) \). We will often skip the prefix ‘re’ and say ‘alignment’ to ease terminology.

There are three laws regulating alignment. Identity updates do not actually need realignment:

\[ (\text{IdAln}) \quad \text{idA} * r = r = r * \text{idB} \]

for any corr \( r : A \leftrightarrow B \).

The result of applying a sequence of interleaving forward and backward alignments does not depend on the order of application as shown in Fig. 11(b):

\[ (\text{fAln-bAln}) \quad (a * r) * b = a * (r * b) \]

for any corr \( r \) and any updates \( a,b \).

We will call diagrams like those shown in Fig. 11(a,b) commutative if the arrow at the respective operation output is indeed equal to that one computed by the operation. For example, diagram (b) is commutative if \( r' = a * r * b \).
Each operation must satisfy the following laws.

1. **Stability** or **ldPpg law**: if nothing changes on one side, nothing happens on the other side as well, that is, identity mappings are propagated into identity mappings as shown by diagrams Fig. 12(b).

2. **Monotonicity**: Insert updates are propagated into inserts, and delete updates are propagated into

Finally, alignment is compositional: for any consecutive updates \( a: A \rightarrow A', a': A' \rightarrow A'', b: B \rightarrow B', b': B' \rightarrow B'' \), the following holds:

\[
(A\text{ln}A\text{ln}) \quad a' * (a * r) = (a; a') * r \quad \text{and} \quad (r * b) * b' = r * (b; b')
\]

where \( ; \) denotes sequential span composition.

**Definition 2** A symmetric delta lens (briefly, an sd-lens) is a triple \((\alpha, f\text{Ppg}, b\text{Ppg})\) with \(\alpha: A^* \times B^* \rightarrow \text{Set}\) an alignment framework, and \(f\text{Ppg}, b\text{Ppg}\) two diagram operations over corrs and updates (called forward and backward update propagation, resp.). The arities are specified in Fig. 12(a) with output arrows dashed and output nodes not framed. Sometimes we will use a linear notation and write \( b = a.f\text{Ppg}(r) \) and \( a = b.b\text{Ppg}(r) \) for the cases specified in the diagrams.

Each operation must satisfy the following laws.

- **Monotonic Compositionality** or \( \text{PpgPpg} \) law: composition of two consecutive inserts is propagated into composition of propagations as shown by the left diagram in Fig. 12(b) (to be read as follows: if the two squares are \( f\text{Ppg}\), then the outer rectangle is \( f\text{Ppg} \) as well). The right diagram specifies compositionality for deletes. The same laws are formulated for \( b\text{Ppg} \).

Note that we do not require compositionality for propagation of general span updates. The point is that interleaving inserts and deletes can annihilate, and lost information cannot be restored: see \[28, 22, 10\] for examples.

- **Commutativity**: Diagrams Fig. 12(a) must be commutative in the sense that \( a * r * b = a'b' \).

Finally, forward and backward propagation must be coordinated with each other by some invertibility law. Given a corr \( r: A \rightarrow B \), an update \( a: A \rightarrow A' \) is propagated into update \( b = a.f\text{Ppg}(r) \), which can be propagated back to update \( a' = b.b\text{Ppg}(r) \). For an ideal situation of strong invertibility, we should require \( a' = a \). Unfortunately, this does not hold in general because the \( A^* \)-specific part of the information is lost.

![Diagram](image-url)
in passing from \( a \) to \( b \), and cannot be restored \[10\]. However, it makes sense to require the following \textit{weak invertibility} specified in Fig. [14] which does hold in a majority of practically interesting situations, e.g., for \( \text{BX} \) determined by TGG-rules \[30\]. The law \( \text{fbfPpg} \) says that although \( a_1 = a.f\text{Ppg}(r).b\text{Ppg}(r) \neq a \), \( a_1 \) is equivalent to \( a \) in the sense that \( a_1.f\text{Ppg}(r) = a.f\text{Ppg}(r) \). Similarly for the \( \text{bfbPpg} \) law.

The notion of \( \text{sd-lens} \) is specified above in elementary terms using tile algebra. Its categorical underpinning is not evident, and we only present several brief remarks.

1) An alignment framework \( \alpha : \textbf{A}^\times \times \textbf{B}^\times \to \textbf{Set} \) can be seen as a profunctor, if \( \textbf{A} \)-arrows will be considered directed up (i.e., the formal source of update \( a \) in diagram Fig. [11](a) is \( a' \), and the target is \( A \)). Then alignment amounts to a functor \( \alpha : \textbf{A}^\times \times \textbf{B}^\times \to \textbf{Set} \), that is, a profunctor \( \alpha : \textbf{B}^\times \to \textbf{A}^\times \). Note that reversing arrows in \( \textbf{A}^\times \) actually changes the arity of operation \( \text{fAln} \): now its input is a pair \((a,r)\) with \( a \) an update and \( r \) a corr from the target of \( a \), and the output is a corr \( r' \) from the source of \( a \), that is, realignment goes back in time.

2) Recall that operations \( \text{fPpg} \) and \( \text{bPpg} \) are functorial wrt. injective arrows in \( \textbf{A}, \textbf{B} \), not wrt. arrows in \( \textbf{A}^\times, \textbf{B}^\times \). However, if we try to resort to \( \textbf{A}, \textbf{B} \) entirely and define alignment wrt. arrows in \( \textbf{A}, \textbf{B} \), then we will need two \( \text{fAln} \) operations with different arities for inserts and deletes, and two \( \text{bAln} \) operations with different arities for inserts and deletes. We will then have four functors \( \alpha_i : \textbf{A} \times \textbf{B} \to \textbf{Set} \) with \( i \) ranging over four-element set \( \{\text{insert,delete}\} \times \{\textbf{A}, \textbf{B}\} \).

3) The weak invertibility laws suggest that a Galois connection/adjunction is somehow hidden in \( \text{sd-lenses} \).

4) Working with chosen spans and pullbacks rather than with their equivalence classes provides a more constructive setting (given we assume the axiom of choice), but then associativity of span composition only holds up to chosen natural isomorphisms, and \( \textbf{A}^\times \) and \( \textbf{B}^\times \) have to be considered bicategories rather than categories.

All in all, we hope that the categorical analysis of asymmetric delta lenses developed by Johnson \textit{et al} [35][34] could be extended to capture the symmetric case too.
Bisimulation of Labeled State-to-Function Transition Systems of Stochastic Process Languages

D. Latella & M. Massink
CNR – Istituto di Scienza e Tecnologie dell’Informazione ‘A. Faedo’

E.P. de Vink *
Technische Universiteit Eindhoven and Centrum Wiskunde Informatica

Abstract Labeled state-to-function transition systems, FuTS for short, admit multiple transition schemes from states to functions of finite support over general semirings. As such they constitute a convenient modeling instrument to deal with stochastic process languages. In this paper, the notion of bisimulation induced by a FuTS is addressed from a coalgebraic point of view. A correspondence result is proven stating that FuTS-bisimulation coincides with the behavioral equivalence of the associated functor. As generic examples, the concrete existing equivalences for the core of the stochastic process algebras PEPA and IML are related to the bisimulation of specific FuTS, providing via the correspondence result coalgebraic justification of the equivalences of these calculi.

1 Introduction

Process description languages equipped with formal operational semantics are successful formalisms for modeling concurrent systems and analyzing their behavior. Typically, the operational semantics is defined by means of a labeled transition system following the SOS approach. The states of the transition systems are just process terms, while the labels of the transitions between states represent the possible actions and interactions. Process description languages often come equipped with process equivalences, so that system models can be compared according to specific behavioral relations.

In the last couple of decades, process languages have been enriched with quantitative information. Among these quantitative extensions, those allowing a stochastic representation of time, usually referred to as stochastic process algebras, have received particular attention. The main aim has been the integration of qualitative descriptions and quantitative analysis in a single mathematical framework by building on the combination of labeled transition systems and continuous-time Markov chains. The latter being one of the most successful approaches to modeling and analyzing the performance of computer systems and networks. An overview on stochastic process algebras, equivalences and related analysis techniques can be found in [14][1][3], for example. A common feature of many stochastic process algebras is that actions are enriched with the rates of exponentially distributed random variables that characterize their duration. Although exploiting the same class of distributions, the models and the techniques underlying the definition of the calculi turn out to be significantly different in many respects. A prominent difference concerns the modeling of the race condition by means of the choice operator, and its relationship to the issue of transition multiplicity. In the quantitative setting, multiplicities can make a crucial distinction between processes that are qualitatively equivalent. Several significantly different approaches have been proposed for handling transition multiplicity. The proposals range from multi-relations [17][13], to proved transition systems [23], to LTS with numbered transitions [14], to unique rate names [9], just to mention a few.

*Corresponding author, email evink@win.tue.nl.
In [7], Latella, Massink et al. have proposed a variant of LTS, called Rate Transition Systems (RTS). In LTS, a transition is a triple \((P, \alpha, P')\) where \(P\) and \(\alpha\) are the source state and the label of the transition, respectively, while \(P'\) is the target state reached from \(P\) via the transition. In RTS, a transition is a triple of the form \((P, \alpha, \mathcal{P})\). The first and second component are the source state and the label of the transition, as in LTS, while the third component \(\mathcal{P}\) is a continuation function which associates a non-negative real value to each state \(P'\). A non-zero value for the state \(P'\) represents the rate of the exponential distribution characterizing the time for the execution of the action represented by \(\alpha\), necessary to reach \(P'\) from \(P\) via the transition. If \(\mathcal{P}\) maps \(P'\) to 0, then state \(P'\) is not reachable from \(P\) via the transition. The use of continuation functions provides a clean and simple solution to the transition multiplicity problem and make RTS particularly suited for stochastic process algebra semantics. In order to provide a uniform account of the many stochastic process algebras proposed in the literature, in previous joint work of the first two authors [8] Labelled State-to-Function Transition Systems (FuTS) have been introduced as a natural generalization of RTS. In FuTS the co-domain of the continuation functions are arbitrary semirings, rather than just the non-negative reals. This provides increased flexibility while preserving basic properties of primitive operations like sum and multiplication.

In this paper we present a coalgebraic treatment of FuTS that allow multiple state-to-function transition relations involving arbitrary semirings. Given label sets \(L_i\) and semirings \(\mathcal{R}_i\), a FuTS takes the general format \(S = (S, (\mapsto_i)_{i=1}^n)\) with transition relations \(\mapsto_i \subseteq S \times L_i \times \mathcal{FS}(S, \mathcal{R}_i)\). Here, \(\mathcal{FS}(S, \mathcal{R}_i)\) are the sets of functions from \(S\) to \(\mathcal{R}_i\) of finite support, a subcollection of functions also occurring in other work combining coalgebra and quantitative modeling. We will associate to \(S\) the product of the functors \(\mathcal{FS}(-; \mathcal{R}_i)^{L_i}\). For this to work, we need the transition relations \(\mapsto_i\) to be total and deterministic for the coalgebraic modeling as a function. Maybe surprisingly, this isn’t a severe restriction at all in the presence of continuation functions: the zero-continuation \(\lambda s'.0\) expresses that no LTS-transition exists from the state \(s\) to any state \(s'\); if \(s\) allows a transition to some state \(s_1\) as well as a state \(s_2\), the continuation function will simply yield a non-zero value for \(s_1\) and for \(s_2\).

The notion of \(S\)-bisimulation that arises from a FuTS \(S\) is reinterpreted coalgebraically as the behavioral equivalence of a functor that is induced by \(S\), along the lines sketched above. Behavioral equivalence rather than coalgebraic bisimulation is targeted, since, dependent on the semirings involved, weak pullbacks may not be preserved and the construction of a mediating morphism for a coalgebraic bisimulation from a concrete one may fail for degenerate denominators. However, following a familiar argument, we show that the functor associated with a FuTS does possess a final coalgebra and therefore has an associated notion of behavioral equivalence indeed. It is noted, in the presence of a final coalgebra for FuTS a more general definition of behavioral equivalence based on cospans coincides [20]. A correspondence result is proven in this paper that shows that the concrete bisimulation of a FuTS, coincides with behavioral equivalence of its functor. Pivotal for its proof is the absence of multiplicities in the FuTS treatment of quantities.

Using the bridge established by the correspondence result, we continue by showing for two well-known stochastic process algebras, viz. Hillston’s PEPA [17] and Hermanns’s IML [13], that the respective standard notion of strong equivalence and strong bisimulation coincides with behavioral equivalence of the associated FuTS. This constitutes the main contribution of the paper. PEPA stands out as one of the prominent Markovian process algebras, while IML specifically provides separate prefix constructions for actions and for delays. The equivalences of PEPA and of IML are compared with the bisimulations of the respective FuTS as given by an alternative operational semantics involving the state-to-function scheme. In passing, the multiplicities have to be dealt with. Appropriate lemmas are provided relating the relation-based cumulative treatment with FuTS to the multirelation-based explicit treatment of PEPA and IML.
Related work on coalgebra includes [28, 19, 26], papers that also cover measures and congruence formats, a topic not touched upon here. For the discrete parts, regarding the correspondence of bisimulations, our work aligns with the approach of the papers mentioned. In this paper the bialgebraic perspective of SOS and bisimulation [27] is left implicit. An interesting direction of research combining coalgebra and quantities studies various types of weighted automata, including linear weighted automata, and associated notions of bisimulation and languages, as well as algorithms for these notions [6, 25, 5].

In particular, building on a result on bounded functors [11], it is shown in [5] for a functor involving functions of finite support over a field that the final coalgebra exists. Below, we have followed the scheme of [5] to obtain such a result for a functor induced by a FuTS. The notions of equivalence addressed in this paper, as often in coalgebraic treatments of process relations, are all strong bisimilarities.

The present paper is organized as follows: Section 2 briefly discusses some material on semirings and coalgebras. Labeled state-to-function transition systems and FuTS as well as the associated notion of bisimulation are provided in Section 3. The coalgebraic counterparts of FuTS and FuTS-bisimulation are defined in Section 4, where we also establish the correspondence with behavioral equivalence of the final coalgebra. In Section 5 the standard equivalence of PEPA is identified with the bisimulation of a FuTS and, hence, with behavioral equivalence. In Section 6 the same is done for the language of IMC where actions and delays are present on equal footing. Section 7 wraps up and discusses directions of future research. An appendix provides the proofs of a number of lemmas.

2 Preliminaries

A tuple \( R = (R, +, 0, *, 1) \) is called a semiring, if \((R, +, 0)\) is a commutative monoid with neutral element 0, \((R, *, 1)\) is a monoid with neutral element 1, * distributes over +, and \(0 * r = r * 0 = 0 \) for all \(r \in R\).

As examples of a semiring we will use the booleans \( \mathbb{B} = \{ \text{false, true} \} \) with disjunction as sum and conjunction as multiplication, and the non-negative reals \( \mathbb{R}_{\geq 0} \) with the standard operations. We will consider, for a semiring \( R \) and a function \( \varphi : X \to R \), countable sums \( \sum_{x \in X} \varphi(x) \) in \( R \), for \( X' \subseteq X \). For such a sum to exist we require \( \varphi \) to be of finite support, i.e. the support set \( \text{sp}(\varphi) = \{ x \in X \mid \varphi(x) \neq 0 \} \) is finite. Here, 0 is the neutral element of \( R \) with respect to +.

We use the notation \( FS(X, R) \) for the collection of all functions of finite support from the set \( X \) to the semiring \( R \). A construct \([x_1 \mapsto r_1, \ldots, x_n \mapsto r_n]\), with \( x_i \in X \), \( i = 1 \ldots n \) all distinct, \( r_i \in R \), \( i = 1 \ldots n \), denotes the mapping that assigns \( r_i \) to \( x_i \), \( i = 1 \ldots n \), and assigns 0 to all \( x \in X \) different from all \( x_j \). In particular [], or more precisely \([\varnothing]\), is the constant function \( x \mapsto 0 \) and \( X_1 = [x \mapsto 1] \) is the characteristic function on \( R \) for \( x \in X \). For \( \varphi \in FS(X, R) \), we write \( \oplus \varphi \) for the value \( \sum_{x \in X} \varphi(x) \) in \( R \). For \( \varphi, \psi \in FS(X, R) \), the function \( \varphi + \psi \) is the pointwise sum of \( \varphi \) and \( \psi \), i.e. \( (\varphi + \psi)(x) = \varphi(x) + \psi(x) \in R \).

Clearly, \( \varphi + \psi \) is of finite support as \( \varphi \) and \( \psi \) are. Given an injective operation \( |: X \times X \to X \), we define \( \varphi \mid \psi : X \to R \), by \( (\varphi \mid \psi)(x) = \varphi(x_1) * \psi(x_2) \) if \( x = x_1 \mid x_2 \) for some \( x_1, x_2 \in X \), and \( (\varphi \mid \psi)(x) = 0 \) otherwise. Again, \( \varphi \mid \psi \) is of finite support as \( \varphi \) and \( \psi \) are. This is used in the setting of syntactic processes \( P \) that may have the form \( P_1 \parallel_A P_2 \) for two processes \( P_1 \) and \( P_2 \) and a syntactic operator \( \parallel_A \).

Lemma 1. Let \( X \) be a set, \( R \) a semiring, and \( | \) an injective binary operation on \( X \). For \( \varphi, \psi \in FS(X, R) \) it holds that \( \oplus(\varphi + \psi) = \oplus \varphi + \oplus \psi \) and \( \oplus(\varphi \mid \psi) = (\oplus \varphi) * (\oplus \psi) \). \( \square \)

We recall some basic definitions from coalgebra. See e.g. [24] for more details. For a functor \( F : \text{Set} \to \text{Set} \) on the category \( \text{Set} \) of sets and functions, a coalgebra of \( F \) is a set \( X \) together with a mapping \( \alpha : X \to F(X) \). A homomorphism between two \( F \)-coalgebras \( (X, \alpha) \) and \( (Y, \beta) \) is a function \( f : X \to Y \) such that \( F(f) \circ \alpha = \beta \circ f \). An \( F \)-coalgebra \( (\Omega, \omega) \) is called final, if there exists, for every \( F \)-coalgebra
(X,α), a unique homomorphism \([\cdot]^{F}_{X} : (X,\alpha) \to (\Omega,\omega)\). Two elements \(x_1, x_2\) of a coalgebra \((X,\alpha)\) are called behavioral equivalent with respect to \(F\) if \(\|x_1\|^{F}_{X} = \|x_2\|^{F}_{X}\), notation \(x_1 \approx_{F} x_2\).

Using a characterization of \([12]\), a functor \(F\) on \(\mathbf{Set}\) is bounded, if there exist sets \(A\) and \(B\) and a surjective natural transformation \(\eta : A \times (\cdot)^B \Rightarrow F\). Here, \(A \times (\cdot)\) is the functor that maps a set \(X\) to the Cartesian product \(A \times X\) and maps a function \(f : X \to Y\) to the mapping \(A \times f : A \times X \to A \times Y\) with \((A \times f)(a,x) = (a,f(x))\), while \((\cdot)^B\) denotes the functor that maps a set \(X\) to the function space \(X^B\) of all functions from \(B\) to \(X\) and that maps a function \(f : X \to Y\) to the mapping \(f^B : X^B \to Y^B\) with \(f^B(\varphi)(b) = f(\varphi(b))\). For bounded functors we have the following result, see \([11]\) for a proof.

**Theorem 2.** If a functor \(F : \mathbf{Set} \to \mathbf{Set}\) is bounded, then its final coalgebra exists. \(\square\)

A number of proofs of results on process languages \(\mathcal{P}\) in this paper relies on so-called guarded recursion \([2]\). Typically, constants \(X\) are a syntactical ingredient in these languages. As usual, if \(X := P\), i.e. the constant \(X\) is declared to have the process \(P\) as its body, we require \(P\) to be prefix-guarded. Thus, any occurrence of a constant in the body \(P\) is in the scope of a prefix-construct of the language. Guarded recursion assumes the existence of a function \(c : \mathcal{P} \to \mathbb{N}\) such that \(c(P_1 \cdot P_2) > \max\{ c(P_1), c(P_2) \}\) for all syntactic operations \(\cdot\) of \(\mathcal{P}\), and moreover \(c(X) > c(P)\) if \(X := P\).

### 3 Labeled State-to-Function Transition Systems

The definition of a labeled state-to-function transition system, \(\text{FuTS}\) for short, involves a set of states \(S\) and one or more relations of states and functions from states into a semiring. For sums over arbitrary subsets of states to exist, the functions are assumed to be of finite support.

**Definition 1.** A FuTS \(S\), in full a labeled state-to-function transition system, over a number of label sets \(\mathcal{L}_i\) and semirings \(\mathcal{R}_i\), \(i = 1\ldots n\), is a tuple \(S = (S, (\mapsto)_1^n)\) such that \(\mapsto_i \subseteq S \times \mathcal{L}_i \times FS(S, \mathcal{R}_i)\) for \(i = 1\ldots n\).

As usual, we write \(s \mapsto_i v\) for \((s,\ell,v) \in \mapsto_i\). For a FuTS \(S = (S, (\mapsto)_1^n)\) the set \(S\) is called the set of states. We refer to each \(\mapsto_i\) as a state-to-function transition relation of \(S\) just as a transition relation of it. If for \(S\) we have that \(n = 1\), i.e. there is only one state-to-function transition relation \(\mapsto\), then \(S\) is called simple. A FuTS \(S\) is called total and deterministic if for each transition relation \(\mapsto_i \subseteq S \times \mathcal{L}_i \times FS(S, \mathcal{R}_i)\) involved and for all \(s \in S\), \(\ell \in \mathcal{L}_i\), we have \(s \mapsto^\ell_i v\) for exactly one \(v \in FS(S, \mathcal{R}_i)\). In such a situation, the zero-function \([\cdot]_{\mathcal{R}_i}\) plays a special role. A state-to-function transition \(s \mapsto^\ell_i v\) reflects the absence of a non-trivial transition \(s \mapsto^\ell_i v\) for \(v \neq [\cdot]_{\mathcal{R}_i}\). In the context of \(\text{LTS}\) one says that \(s\) has no \(\ell\)-transition. For the remainder of the paper, all FuTS we consider are assumed to be total and deterministic.

**Examples** For the modeling of CCS processes, we choose a set of actions \(\mathcal{A}\) as label set and the booleans \(\mathbb{B}\) as semiring. Consider the two CCS processes \(P = a.b.0 + a.c.0\) and \(Q = a.(b.0 + c.0)\), their representation as a FuTS is depicted in Figure 1. For process \(P\) we have \(P \mapsto^a [b.0 \mapsto \text{true}, c.0 \mapsto \text{true}]\), while the process \(Q\) we have \(Q \mapsto^a [b.0 + c.0 \mapsto \text{true}]\). So, FuTS are able to represent branching.

As another example of a simple FuTS, Figure 1 displays a FuTS over the action set \(\mathcal{A}\) and the semiring \(\mathbb{R}_{\geq 0}\) of the non-negative real numbers. The functions \(v_0\) to \(v_4\) used in the example have the property that \(\bigoplus v_i(s) = 1\), for \(i = 0\ldots 4\). Usually, such a FuTS over \(\mathbb{R}_{\geq 0}\) is called a (reactive) probabilistic transition system.

\(^{1}\)Definition \([1]\) slightly differs in formulation from the one in \([8]\).
Hence, we use the notation \( t \in S \subseteq \text{equivalence relation } R \) of bisimilarity for a \( \text{FuTS} \) function \( \theta \). Additional property that \( \text{FuTS} \) one could see a \( \text{FuTS} \) chains \([13, 16]\) using \( \theta \). It will be notationally convenient to consider a (total and deterministic) \( \text{FuTS} \) as a tuple \((S, \langle \theta_i \rangle_{i=1}^n)\) with transition functions \( \theta_i : S \rightarrow L_i \rightarrow \mathcal{FS}(S, R_i), \ i = 1 \ldots n \), rather than using the form \((S, \langle \theta_i \rangle_{i=1}^n)\) that occurs more frequent for concrete examples in the literature. Alternatively, using disjoint unions, one could see a \( \text{FuTS} \) represented by a function \( \theta' : S \rightarrow \bigoplus_{i=1}^n L_i \rightarrow \bigoplus_{i=1}^n \mathcal{FS}(S, R_i) \) satisfying the additional property that \( \theta'(s)(\ell) \in \mathcal{FS}(S, R_i) \) if \( \ell \in L_i \). As this fits less smoothly with the category-theoretical approach of Section\(^4\) we stick to the former format. Note, an interpretation of a \( \text{FuTS} \) as a function \( S \rightarrow \bigoplus_{i=1}^n (L_i \rightarrow \mathcal{FS}(S, R_i)) \) does not suit our purposes as the \( \text{IML} \) example above illustrates.

We will use the notation with transition functions \( \theta_i : S \rightarrow L_i \rightarrow \mathcal{FS}(S, R_i) \) to introduce the notion of bisimilarity for a \( \text{FuTS} \).

**Definition 2.** Let \( S = (S, \langle \theta_i \rangle_{i=1}^n) \) be a \( \text{FuTS} \) over the label sets \( L_i \) and semirings \( R_i, \ i = 1 \ldots n \). An equivalence relation \( R \subseteq S \times S \) is called an \( S \)-bisimulation if \( R(s_1, s_2) \) implies

\[
\sum_{t' \in [t]_R} \theta_i(s_1)(\ell)(t') = \sum_{t' \in [t]_R} \theta_i(s_2)(\ell)(t')
\]

for all \( t \in S, \ i = 1 \ldots n \) and \( \ell \in L_i \). Two elements \( s_1, s_2 \in S \) are called \( S \)-bisimilar if \( R(s_1, s_2) \) for some \( S \)-bisimulation \( R \) for \( S \). Notation \( s_1 \sim_R s_2 \).

We use the notation \([t]_R\) to denote the equivalence class of \( t \in S \) with respect to \( R \). Note that sums in equation \(1\) exist since the functions \( \theta_i(s_j)(\ell) \in \mathcal{FS}(S, R_i), \ i = 1 \ldots n, \ j = 1, 2, \) are of finite support. Hence, \( \theta_i(s_j)(\ell)(t') = 0 \in R_i \) for all but finitely many \( t' \in [t]_R \subseteq S \).

For the combined \( \text{FuTS} \) of the two CCS-processes of Figure\(^1\) the obvious equivalence relation relating \( b.0 \) and \( b.0 + c.0 \) is not a \( \text{FuTS} \)-bisimulation. Although \( \sum_{t' \in [0]_R} \theta(b.0)(b)(t') = \theta(b.0)(b)(0) = \text{true} \) and \( \sum_{t' \in [0]_R} \theta(b.0 + c.0)(b)(t') = \theta(b.0 + c.0)(b)(0) = \text{true} \), we have \( \sum_{t' \in [0]_R} \theta(b.0)(c)(t') = \text{false} \), while \( \sum_{t' \in [0]_R} \theta(b.0 + c.0)(c)(t') = \text{true} \), taking sums, i.e. disjunctions, in \( \mathbb{B} \).

### 4 \text{FuTS} co algebraically

In this section we will cast \( \text{FuTS} \) in the framework of coalgebras and prove a correspondence result of \( \text{FuTS} \)-bisimulation and behavioral equivalence for a suitable functor on \( \text{Set} \).
Let \( L \) be a set of labels and let \( R \) be a semiring. The functor \( V^L_R : \text{Set} \to \text{Set} \) assigns to a set \( X \) the function space \( FS(X, R)^L \) of all functions \( \varphi : L \to FS(X, R) \) and assigns to a function \( f : X \to Y \) the mapping \( V^L_R(f) : FS(X, R)^L \to FS(Y, R)^L \) where

\[
V^L_R(f)(\varphi)(\ell) = \sum_{x' \in f^{-1}(y)} \varphi(\ell)(x')
\]

for all \( \varphi \in FS(X, R)^L \), \( \ell \in L \) and \( y \in Y \).

Again we rely on \( \varphi(\ell) \in FS(X, R) \) having a finite support for the sum to exist and for \( V^L_R \) being well-defined. In fact, we have \( \text{spt}(V^L_R(f)(\varphi)(\ell)) = \{ f(x) \mid x \in \text{spt}(\varphi(\ell)) \} \).

As we aim to compare our notion of bisimulation for \( \text{FuTS} \) with behavioral equivalence for the functor \( V^L_R \), given a set of labels \( L \) and a semiring \( R \), we need to check that \( V^L_R \) possesses a final coalgebra. We follow the approach of \([S]\).

**Lemma 3.** Let \( L \) be a set of labels, \( R \) a semiring. Then the functor \( V^L_R \) on \( \text{Set} \) is bounded. \( \square \)

Working with total and deterministic \( \text{FuTS} \), we can interpret a \( \text{FuTS} \) \( S = (S, \theta_i)_{i=1}^n \) over the label sets \( L_i \) and semirings \( R_i \), \( i = 1 \ldots n \) as a product \( \theta_1 \times \cdots \times \theta_n : S \to \prod_{i=1}^n (L_i \to FS(S, R_i)) \) of functions \( \theta_i : S \to L_i \to FS(S, R_i) \). To push this idea a bit further, we want to consider the \( \text{FuTS} \) \( S = (S, \theta_i)_{i=1}^n \) as a coalgebra of a suitable product functor on \( \text{Set} \).

**Definition 4.** Let \( S = (S, \theta_i)_{i=1}^n \) be a \( \text{FuTS} \) over the label sets \( L_i \) and semirings \( R_i \), \( i = 1 \ldots n \). The functor \( V_S \) on \( \text{Set} \) is defined by \( V_S = \prod_{i=1}^n V^L_{R_i} = \prod_{i=1}^n FS(\cdot, R_i)^{L_i} \).

The point is, under conditions that are generally met, coalgebras come equipped with a natural notion of behavioral equivalence that can act as a reference for strong equivalences, in particular of bisimulation for \( \text{FuTS} \). Below, see Theorem\([S]\), we prove that \( S \)-bisimilarity as given by Definition\([S]\) coincides with behavioral equivalence for the functor \( V_S \) as given by Definition\([S]\) providing justification for the notion of equivalence defined on \( \text{FuTS} \).

For the notion of behavioral equivalence for the functor \( V_S \) obtained from \( S \) to be defined, we establish that it possesses a final coalgebra.

**Theorem 4.** The functor \( V_S \) has a final coalgebra.

**Proof.** By Lemma\([S]\) we have that each factor \( V^L_{R_i} \) of \( V_S \) is bounded, and hence possesses a final coalgebra \( \Omega_{V^L_{R_i}} \) by Theorem\([S]\). It follows that also \( V_S \) has a final coalgebra \( \Omega_S \). Writing \( [[\cdot]]_X^S \) for the final morphism of a \( V_S \)-coalgebra \( X \) into \( \Omega_S \), we have

\[
\Omega_S = \Omega_{V^L_{R_1}} \times \cdots \times \Omega_{V^L_{R_n}} \quad \text{and} \quad [[\cdot]]_X^S = [[\cdot]]_X^{V^L_{R_1}} \times \cdots \times [[\cdot]]_X^{V^L_{R_n}}
\]

as can be straightforwardly shown. \( \square \)

Since the functor \( V_S \) of a \( \text{FuTS} \) has a final coalgebra, we can speak of the behavioral equivalence \( \approx_S \) induced by \( V_S \). Next we establish, for a \( \text{FuTS} \) \( S \), the correspondence of \( S \)-bisimulation \( \sim_S \) as given by Definition\([S]\) and behavioral equivalence \( \approx_S \).

**Theorem 5.** Let \( S = (S, \theta_i)_{i=1}^n \) be a \( \text{FuTS} \) over the label sets \( L_i \) and semirings \( R_i \), \( i = 1 \ldots n \). Then \( s_1 \sim_S s_2 \iff s_1 \approx_S s_2 \), for all \( s_1, s_2 \in S \).
Proof. Let \( s_1, s_2 \in S \). We first prove \( s_1 \sim_S s_2 \Rightarrow s_1 \approx_S s_2 \). So, assume \( s_1 \sim_S s_2 \). Let \( R \subseteq S \times S \) be an \( S \)-bisimulation with \( R(s_1, s_2) \). Put \( \theta = \theta_1 \times \cdots \times \theta_n \). Note \((S, \theta)\) is a \( \mathcal{V}_S \)-coalgebra. We turn the collection of equivalence classes \( S/R \) into a \( \mathcal{V}_S \)-coalgebra \((S/R, \theta_R)\) by putting
\[
\theta_R^i([s]_R)(\ell)([t]_R) = \sum_{t \in [t]_R} \theta_i(s)(\ell)(t') \quad \text{and} \quad \theta_R = \theta_1^r \times \cdots \times \theta_n^r
\]
for \( s, t \in S \), \( \ell \in L_i \), \( i = 1 \ldots n \). This is well-defined since \( R \) is an \( S \)-bisimulation: if \( R(s, s') \) then we have \( \sum_{t \in [t]_R} \theta_i(s)(\ell)(t') = \sum_{t' \in [t']_R} \theta_i(s')(\ell)(t') \). The canonical mapping \( \varepsilon_R : S \to S/R \) is a \( \mathcal{V}_S \)-homomorphism: For \( i = 1 \ldots n \), \( \ell \in L_i \) and \( t \in S \), we have both
\[
\mathcal{F}(e_R, R_i)^{L_i}(\theta_i(s)(\ell)([t]_R)) = \sum_{t' \in [t]_R} \theta_i(s)(\ell)(t') \quad \text{and} \quad \theta_R^i([s]_R)(\ell)([t]_R) = \sum_{t \in [t]_R} \theta_i(s)(\ell)(t')
\]
Thus, \( \mathcal{F}(e_R, R_i)^{L_i} \circ \theta_i = \theta_R^i \circ \varepsilon_R \). Since \( \mathcal{V}_S(e_R) = \prod_{i=1}^n \mathcal{F}(e_R, R_i)^{L_i} \) it follows that \( e_R \) is a \( \mathcal{V}_S \)-homomorphism. Therefore, by uniqueness of a final morphism, we have \([\cdot : \ell]_S = [\cdot : \ell]_{S/R} \circ e_R \). In particular, \([s_1]_S\) is an \( S \)-bisimulation. Since the map \([\cdot : \ell]_S : (S, \theta) \to (\Omega_S, \omega_S) \) is a \( \mathcal{V}_S \)-homomorphism, the relation \( R_S \) with \( R_S(s', s'') \Leftrightarrow [s']_S = [s'']_S \) is an \( S \)-bisimulation: Suppose \( R_S(s', s'') \), i.e. \( s' \approx_S s'' \), for some \( s', s'' \in S \). Assume \( \theta_\Omega = \theta_1^r \times \cdots \times \theta_n^r \). Pick \( 1 \leq i \leq n \), \( \ell \in L_i \), \( t \in S \). Put \([t]_S = w \in \Omega_S \). Let \([t]_S \) denote the equivalence class of \( t \) in \( R_S \).
\[
\sum_{t \in [t]_S} \theta_i(s')(\ell)(t') = \sum_{t \in [t]_S} \theta_i(s'')(\ell)(t') \quad \text{and} \quad \theta_R^i([s]_R)(\ell)([t]_R) = \sum_{t \in [t]_R} \theta_i(s)(\ell)(t')
\]
Thus, if \( R_S(s', s'') \) then \( \sum_{t \in [t]_S} \theta_i(s')(\ell)(t') = \sum_{t \in [t]_S} \theta_i(s'')(\ell)(t') \) for all \( i = 1 \ldots n \), \( t \in S \), \( \ell \in L_i \) and \( R_S \) is an \( S \)-bisimulation. Since \([s_1]_S = [s_2]_S \), it follows that \( R_S(s_1, s_2) \). Thus \( R_S \) is an \( S \)-bisimulation relating \( s_1 \) and \( s_2 \). Conclusion, it holds that \( s_1 \approx_S s_2 \).

5 FuTS Semantics of PEPA

Next we will consider a significant fragment of the process algebra PEPA [17], including the parallel operator implementing the scheme of so-called minimal apparent rates, and provide a FuTS semantics for it. We will show that PEPA’s notion of equivalence \( \approx_{\text{pepa}} \), called strong equivalence in [17], fits with the bisimilarity \( \approx_{\text{pepa}} \) as arising from the FuTS semantics.

Definition 5. The set \( \mathcal{P}_{\text{PEPA}} \) of PEPA processes is given by the BNF
\[
P ::= \text{nil} | (a, \lambda).P | P + P | P || A.P | X
\]
where \( a \) ranges over the set of actions \( A \), \( \lambda \) over \( \mathbb{R}_{>0} \). \( A \) over the set of finite subsets of \( A \), and \( X \) over the set of constants \( X \).

PEPA, like many other stochastic process algebras (e.g. [15] [4]), couples actions and rates. The prefix \((a, \lambda)\) of the process \((a, \lambda).P\) expresses that the duration of the execution of the action \( a \in A \) is sampled from an exponential distribution of rate \( \lambda \). The parallel composition \( P || A.Q \) of a process \( P \) and a process \( Q \)
Figure 2: FuTS semantics for PEPA.

for a set of actions $A \subseteq \mathcal{A}$ allows for the independent, asynchronous execution of actions of $P$ and $Q$ not occurring in the subset $A$, on the one hand, and requires the simultaneous, synchronized execution of $P$ and $Q$ for the actions occurring in $A$, on the other hand. The FuTS-semantics of the fragment of PEPA that we consider here, is given by the SOS of Figure 2 on which we comment below.

Characteristic for the PEPA language is the choice to model parallel composition, or cooperation in the terminology of PEPA, scaled by the minimum of the so-called apparent rates. By doing so, PEPA’s strong equivalence becomes a congruence [17]. Intuitively, the apparent rate $r_a(P)$ of an action $a$ for a process $P$ is the sum of the rates of all possible $a$-executions for $P$. When considering the CSP-style parallel composition $P || A Q$, with cooperation set $A$, an action $a$ occurring in $A$ has to be performed by both $P$ and $Q$. The rate of such an execution is governed by the slowest, on average, of the two processes in this respect [2]. Thus $r_a(P || A Q)$ for $a \in A$ is the minimum $\min\{r_a(P), r_a(Q)\}$. Now, if $P$ schedules an execution of $a$ with rate $r_1$ and $Q$ schedules a transition of $a$ with rate $r_2$, in the minimal apparent rate scheme the combined execution yields the action $a$ with rate $r_1 \cdot r_2 \cdot \text{arf}(P, Q)$. Here, the ‘syntactic’ scaling factor $\text{arf}(P, Q)$, the apparent rate factor, is defined by

$$\text{arf}(P, Q) = \frac{\min\{r_a(P), r_a(Q)\}}{r_a(P) \cdot r_a(Q)}$$

assuming $r_a(P), r_a(Q) > 0$, otherwise $\text{arf}(P, Q) = 0$. Thus, for $P || A Q$ the minimum $\min\{r_a(P), r_a(Q)\}$ of the apparent rates is adjusted by the relative probabilities $r_1 / r_a(P)$ and $r_2 / r_a(Q)$ for executing $a$ by $P$ and $Q$, respectively. See [17] Definition 3.3.1] (or the appendix) for an explicit definition of the apparent rate $r_a$ of a PEPA-process.

The FuTS we consider for the semantics of PEPA in Figure 2 involves a set of labels $\Delta$ defined by $\Delta = \{ \delta_a | a \in \mathcal{A} \}$. The symbol $\delta_a$ denotes the execution of the action $a$, with a duration that is still to be established. The underlying semiring for the simple FuTS for PEPA is the semiring $\mathbb{R}_{\geq 0}$ of non-negative reals.

**Definition 6.** The FuTS $S_{pepa} = (\mathcal{P}_{\text{PEPA}}, \rightarrow_p)$ over $\Delta$ and $\mathbb{R}_{\geq 0}$ has its transition relation given by the rules of Figure 2.

We discuss the rules of Figure 2. The FuTS semantics provides $\text{nil} \overset{\delta_a}{\rightarrow_p} [\lambda]_{\mathbb{R}_{\geq 0}}$, for every action $a$, with $[\lambda]_{\mathbb{R}_{\geq 0}}$ the 0-function $\lambda P 0$ of $\mathbb{R}_{\geq 0}$. However, the latter expresses $\theta_{\text{pepa}}(\text{nil})(\delta_a)(P') = 0$ for every $a \in \mathcal{A}$.

\footnote{One cannot take the slowest process per sample, because such an operation cannot be expressed as an exponential distribution in general.}
and \( P' \in \mathcal{P}_{\text{PEPA}} \), or, in standard terminology, \textbf{nil} has no transition. For the rated action prefix \((a,\lambda)\) we distinguish two cases: (i) execution of the prefix in rule (RAPF1); (ii) no execution of the prefix in rule (RAPF2). In the case of rule (RAPF1) the label \( \delta_a \) signifies that the transition involves the execution of the action \( a \). The continuation \([P \rightarrow \lambda]\) is the function that assigns the rate \( \lambda \) to the process \( P \). All other processes are assigned 0, i.e. the zero-element of the semiring \( \mathbb{R}_{\leq 0} \). In the second case, rule (RAPF2), for labels \( \delta_b \) with \( b \neq a \), we do have a transition, but it is a degenerate one. The two rules for the prefix, in particular having the ‘null-continuation’ rule (RAPF2), support the unified treatment of the choice operator in rule (CHO) and the parallel operator in rules (PAR1) and (PAR2).

Note the semantic sum of functions \( P + Q \) replacing the syntactic sum in \( P + Q \). The treatment of constants is as usual. Regarding the parallel operator \(|_A|\), with respect to some subset of actions \( A \subseteq \mathcal{A} \), the so-called cooperation set, there are again two rules. Now the distinction is between interleaving and synchronization. In the case of a label \( \delta_a \) involving an action \( a \) not in the subset \( A \), either the \( P \)-operand or the \( Q \)-operand of \( P \|_A Q \) makes progress. For example, the effect of the pattern \( P \|_A Q \) is that the value \( P(P') \cdot 1 \) is assigned to a process \( P' \|_A Q \), the value \( P(P') \cdot 0 = 0 \) to a process \( P' \|_A Q' \) for some \( Q' \neq Q \), and the value 0 for a process not of the form \( P' \|_A Q' \). Here, as in all other rules, the right-hand sides of the transitions only involve functions in \( \mathcal{T}_S(\mathcal{P}_{\text{PEPA}}, \mathbb{R}_{\geq 0}) \) and operators on them.

For the synchronization case of the parallel construct, assuming \( P \overset{\delta_a}{\rightarrow} P \) and \( Q \overset{\delta_b}{\rightarrow} Q \), the ‘semantic’ scaling factor \( \text{arf}(P, Q) \) is applied to \( P \|_A Q \) (with \(|_A| \) on \( \mathcal{T}_S(\mathcal{P}_{\text{PEPA}}, \mathbb{R}_{\geq 0}) \) induced by \(|_A| \) on \( \mathcal{P}_{\text{PEPA}} \)). This scaling factor, defined for functions in \( \mathcal{T}_S(\mathcal{P}_{\text{PEPA}}, \mathbb{R}_{\geq 0}) \), is given by

\[
\text{arf}(P, Q) = \min\{\oplus P, \oplus Q\} / \oplus P \cdot \oplus Q
\]

provided \( \oplus P, \oplus Q > 0 \), and \( \text{arf}(P, Q) = 0 \) otherwise. This results for \( \text{arf}(P, Q) \cdot (P \|_A Q) \), for a process \( R = R_1 \|_A R_2 \), in the value \( \text{arf}(P, Q) \cdot (P \|_A Q)(R_1 \|_A R_2) = \text{arf}(P, Q) \cdot P(R_1) \cdot Q(R_2) \).

The following lemma establishes the relationship between the ‘syntactic’ and ‘semantic’ apparent rate factors defined on processes and on continuation functions, respectively.

**Lemma 6.** Let \( P \in \mathcal{P}_{\text{PEPA}} \) and \( a \in \mathcal{A} \). Suppose \( P \overset{\delta_a}{\rightarrow} P \). Then \( r_d(P) = \oplus P \).

The proof of the lemma is straightforward. It is also easy to prove, by guarded induction, that the \( \mathcal{FuTS} \) \( S_{\text{pepa}} \) given by Definition 5 is total and deterministic. So, it is justified to write \( S_{\text{pepa}} = (\mathcal{P}_{\text{PEPA}}, \theta_{\text{pepa}}) \). We use \( \sim_{\text{pepa}} \) to denote the bisimilarity induced by \( S_{\text{pepa}} \).

**Lemma 7.** The \( \mathcal{FuTS} \) \( S_{\text{pepa}} \) is total and deterministic.

**Example** To illustrate the ease to deal with multiplicities in the \( \mathcal{FuTS} \) semantics, consider the PEPA processes \( P_1 = (a,\lambda).P \) and \( P_2 = (a,\lambda).P + (a,\lambda).P \) for some \( P \in \mathcal{P}_{\text{PEPA}} \). We have \( P_1 \overset{\delta_a}{\rightarrow} \{P \rightarrow \lambda\} \) by rule (RAPF1), but \( P_2 \overset{\delta_a}{\rightarrow} \{P \rightarrow 2\lambda\} \) by rule (RAPF1) and rule (CHO). The latter makes us to compute \([P \rightarrow \lambda] + [P \rightarrow \lambda]\), which equals \([P \rightarrow 2\lambda]\). Thus, in particular we have \( P_1 \sim_{\text{pepa}} P_2 \). Intuitively it is clear that, in general we cannot have \( P + P \sim P \) for any reasonable quantitative process equivalence \( \sim \) in the Markovian setting. Having twice as many \( a \)-labelled transitions, the average number for \((a,\lambda).P + (a,\lambda).P\) of executing the action \( a \) per time unit is double the average of executing \( a \) for \((a,\lambda).P\).

The standard operational semantics of PEPA \([7,18]\) is given in Figure 3. The transition relation \( \rightarrow \subseteq \mathcal{P}_{\text{PEPA}} \times (\mathcal{A} \times \mathbb{R}_{\geq 0}) \times \mathcal{P}_{\text{PEPA}} \) is the least relation satisfying the rules. For a proper treatment of the rates, the transition relation is considered as a multi-transition system, where also the number of possible
Bisimulation of FuTS

The so-called total conditional transition rate $q[P,C,a]$ of a PEPA-process [17, 18] for a subset of processes $C \subseteq \mathcal{P}_{PEPA}$ and $a \in \mathcal{A}$ is given by $q[P,C,a] = \sum_{Q \in C} \sum_{\lambda} \| P \xrightarrow{a,\lambda} Q \|$. Here, $\| P \xrightarrow{a,\lambda} Q \|$ is the multiset of transitions $P \xrightarrow{a,\lambda} Q$ and $\| P \xrightarrow{a,\lambda} Q \|$ is the multiset of all $\lambda$’s involved. The multiplicity of $P \xrightarrow{a,\lambda} Q$ is the number of different ways the transition can be derived using the rules of Figure 3. We are now ready to define PEPA’s notion of strong equivalence [17, 18].

**Definition 7.** An equivalence relation $R \subseteq \mathcal{P}_{PEPA} \times \mathcal{P}_{PEPA}$ is called a strong equivalence if $q[P_1,[Q]_R,a] = q[P_2,[Q]_R,a]$ for all $P_1, P_2 \in \mathcal{P}_{PEPA}$ such that $R(P_1,P_2)$, all $Q \in \mathcal{P}_{PEPA}$ and all $a \in \mathcal{A}$. Two processes $P_1, P_2 \in \mathcal{P}_{PEPA}$ are strongly equivalent if $R(P_1,P_2)$ for a strong equivalence $R$, notation $P_1 \equiv_{pepa} P_2$. •

The next lemma couples, for a PEPA-process $P$, an action $a$ and a function $\mathcal{P} \in \mathcal{T}S(\mathcal{P}_{PEPA}, \mathbb{R}_{\geq 0})$, the evaluation $\mathcal{P}(P')$ with respect to the FuTS-semantics to the cumulative rate for $P$ of reaching $P'$ by a transition involving the label $a$ in the standard operational semantics.

**Lemma 8.** Let $P \in \mathcal{P}_{PEPA}$ and $a \in \mathcal{A}$. Suppose $P \xrightarrow{\delta_a} \mathcal{P}$. Then it holds that $\mathcal{P}(P') = \sum \| P \xrightarrow{a,\lambda} P' \|$ for all $P' \in \mathcal{P}_{PEPA}$. □

With the lemma in place we can prove the following correspondence result for $S_{pepa}$-bisimilarity with respect to the FuTS for PEPA of Definition 6 and strong equivalence as given by Definition 7.

**Theorem 9.** For any two PEPA-processes $P_1, P_2 \in \mathcal{P}_{PEPA}$ it holds that $P_1 \equiv_{pepa} P_2$ iff $P_1 \equiv_{FuTS} P_2$.

**Proof.** Let $R$ be an equivalence relation on $\mathcal{P}_{PEPA}$. Choose $P, Q \in \mathcal{P}_{PEPA}$ and $a \in \mathcal{A}$. Suppose $P \xrightarrow{\delta_a} \mathcal{P}$. Thus $\theta_{pepa}(P)(\delta_a) = \mathcal{P}$. We have

$$q[P,[Q]_R,a] = \sum_{Q' \in [Q]_R} \sum_{\lambda} \| P \xrightarrow{a,\lambda} Q' \| \quad \text{(by definition $q[P,[Q]_R,a]$)}$$

$$= \sum_{Q' \in [Q]_R} \mathcal{P}(Q') \quad \text{(by Lemma 6)}$$

$$= \sum_{Q' \in [Q]_R} \theta_{pepa}(P)(a)(Q') \quad \text{(by definition $\theta_{pepa}$)}$$

Therefore, for PEPA-processes $P_1$ and $P_2$ it holds that $q[P_1,[Q]_R,a] = q[P_2,[Q]_R,a]$ for all $Q \in \mathcal{P}_{PEPA}$, $a \in \mathcal{A}$ if $\sum_{Q' \in [Q]_R} \theta_{pepa}(P_1)(a)(Q') = \sum_{Q' \in [Q]_R} \theta_{pepa}(P_2)(a)(Q')$ for all $Q \in \mathcal{P}_{PEPA}$, $a \in \mathcal{A}$. Thus, the equivalence relation $R$ is a strong equivalence iff $R$ is an $S_{pepa}$-bisimulation, from which the theorem follows. □
In view of our general correspondence result Theorem \[\ref{thm:inl:correspondence}\] the above theorem shows that PEPA’s strong equivalence \(\approx_{\text{pepa}}\) is a behavioral equivalence, viz. behavioral equivalence \(\approx_{\text{pepa}}\) with respect to the functor of \(\mathcal{S}_{\text{pepa}}\), and that its standard, FuTS and coalgebraic semantics coincide.

6 FuTS Semantics of IML

In this section we provide a FuTS semantics for a relevant part of the language of IMC \[\ref{imc}\]. IMC, Interactive Markov Chains, are automata that combine two types of transitions: interactive transitions that involve the execution of actions and Markovian transitions that represent the progress of time governed by exponential distribution. As a consequence, IMC embody both non-deterministic and stochastic behaviour. System analysis using IMC proves to be a powerful approach because of the orthogonality of qualitative and quantitative dynamics, their logical underpinning and tool support. A number of equivalences, both strong and weak, are available for IMC \[\ref{imc:equivalences}\]. In our treatment here, dealing with a fragment we call IML, we do not deal with internal \(\tau\)-steps and focus on strong bisimulation.

**Definition 8.** The set \(\mathcal{P}_{\text{IML}}\) of IML processes is given by the BNF \(P := \text{nil} | a.P | \lambda.P | P + P | \bigwedge A P | X\) where \(a\) ranges over the set of actions \(\mathcal{A}\), \(\lambda\) over \(\mathbb{R}_{>0}\), \(A\) over the set of finite subsets of \(\mathcal{A}\) and \(X\) over the set of constants \(X\).

In IML there are separate prefix constructions for actions \(a.P\) and for time-delays \(\lambda.P\). No restriction is imposed on the alternative and parallel composition of processes. For example, we have the process \(P = a.\lambda.\text{nil} + \mu.b.\text{nil}\) in IML. It should be noted that for IMC actions are considered to take no time.

**Definition 9.** The formal semantics of \(\mathcal{P}_{\text{IML}}\) is given by the FuTS \(S_{\text{iml}} = (\mathcal{P}_{\text{IML}}, \rightarrow_1, \rightarrow_2)\) over the label sets \(\mathcal{A}\) and \(\Delta = \{\delta\}\) and the semirings \(\mathbb{B}\) and \(\mathbb{R}_{\geq 0}\) with transition relations \(\rightarrow_1 \subseteq \mathcal{P}_{\text{IML}} \times \mathcal{A} \times \mathcal{F}(\mathcal{P}_{\text{IML}}, \mathbb{B})\) and \(\rightarrow_2 \subseteq \mathcal{P}_{\text{IML}} \times \Delta \times \mathcal{F}(\mathcal{P}_{\text{IML}}, \mathbb{R}_{\geq 0})\) defined as the least relations satisfying the rules of Figure 4.

To accommodate for action-based and delay-related transitions, the FuTS \(S_{\text{iml}}\) is non-simple, having the two transition-to-function relations \(\rightarrow_1\) and \(\rightarrow_2\). Actions \(a \in \mathcal{A}\) decorate \(\rightarrow_1\), the special symbol \(\delta\) decorates \(\rightarrow_2\). Note rule (APF3) and rule (RPF1) that involve the null-functions of \(\mathbb{R}_{\geq 0}\) and of \(\mathbb{B}\), respectively, to express that a process \(a.P\) does not trigger a delay and a process \(\lambda.P\) does not execute an action. For the
parallel construct $\parallel A$, interleaving applies both for non-synchronized actions $a \notin A$ as well as for delays (but not mixed). Therefore, rule (PAR1) pertains to both $\rightarrow_1$ and $\rightarrow_2$, with $a$ ranging over $A \cup \Delta$. The same holds for non-deterministic choice, rule (CHO), and constants, rule (CON). Finally, IML does not provide synchronization of delays in the parallel construct. Rule (PAR2) only concerns the transition relation $\rightarrow_2$. In rule (PAR1), for clarity, we decorated the characteristic functions, writing $X^i_p$, for $i = 1, 2$, for $X_p = [ P \rightarrow \text{true} ]$ in $\mathcal{FS}(P_{\text{IML}}, \mathbb{R})$ and $X_p = [ P \rightarrow 1 ]$ in $\mathcal{FS}(P_{\text{IML}}, \mathbb{R}_{\geq 0})$.

**Example** Assume $X := a.\lambda.b.X$ and $Y := a.\mu.b.Y$. Put $A = \{ a, b \}$. Then we have

\[
\begin{align*}
X \parallel A Y & \rightarrow_1 [ \lambda.b.X || A \mu.b.Y \rightarrow \text{true} ] & \lambda.b.X || A \mu.b.Y & \rightarrow_2 [ b.X || A \mu.b.Y \rightarrow \lambda, \lambda.b.X || A b.Y \rightarrow \mu ] \\
 b.X || A b.Y & \rightarrow_1 [ X \parallel A Y \rightarrow \text{true} ] & b.X || A b.Y & \rightarrow_2 [ b.X || A b.Y \rightarrow \lambda, \lambda.b.X || A b.Y \rightarrow \mu ] \\
 & & \lambda.b.X || A b.Y & \rightarrow_2 [ b.X || A b.Y \rightarrow \lambda ]
\end{align*}
\]

It is not difficult to verify that $S_{\text{IML}}$ is a total and deterministic FuTS. Below we use $S_{\text{IML}} = (P_{\text{IML}}, \theta_1, \theta_2)$ and write $\sim_{\text{IML}}$ for the associated bisimilarity.

**Lemma 10.** The FuTS $S_{\text{IML}}$ is total and deterministic. \hfill \Box

The standard SOS semantics of IML [13] is given in Figure 5 involving the transition relations

\[
\rightarrow \subseteq P_{\text{IML}} \times \mathcal{A} \times P_{\text{IML}} \quad \text{and} \quad \rightarrow_\rightarrow \subseteq P_{\text{IML}} \times \mathbb{R}_{\geq 0} \times P_{\text{IML}}
\]

Below we will use the functions $T$ and $R$ based on $\rightarrow$ and $\rightarrow_\rightarrow$, cf. [16]. We have $T \colon P_{\text{IML}} \times \mathcal{A} \times 2^{P_{\text{IML}}} \rightarrow \mathbb{R}$ given by $T(P, a, C) = \text{true}$ if the set $\{ P' \in C \mid P \\rightarrow a P' \}$ is non-empty, for all $P \in P_{\text{IML}}$, $a \in \mathcal{A}$ and any subset $C \subseteq P_{\text{IML}}$. For $R \colon P_{\text{IML}} \times P_{\text{IML}} \rightarrow \mathbb{R}_{\geq 0}$ we put $R(P, P') = \sum \| \lambda \mid P \rightarrow \lambda P' \|$. Here, as common for probabilistic and stochastic process algebras, the comprehension is over the multiset of transitions leading from $P$ to $P'$ with label $\lambda$. We extend $R$ to $P_{\text{IML}} \times 2^{P_{\text{IML}}}$ by $R(P, C) = \sum_{P' \in C} \sum \| \lambda \mid P \rightarrow \lambda P' \|$. For IML we have the following notion of strong bisimulation [13, 16] that we will compare with the notion of bisimulation associated with the FuTS $S_{\text{IML}}$.  

\[
\text{(APF)} \quad \frac{a.P \rightarrow P}{a.P \rightarrow P} \\
\text{(CHO1)} \quad \frac{P \rightarrow a R}{P + Q \rightarrow a R} \\
\text{(CHO2)} \quad \frac{Q \rightarrow a R}{P + Q \rightarrow a R} \\
\text{(CON1)} \quad \frac{P \rightarrow a Q}{X \rightarrow a Q}
\]

\[
\text{(PAR1a)} \quad \frac{P \parallel A Q \rightarrow a R}{P \parallel A Q \rightarrow a P'} \\
\text{(PAR1b)} \quad \frac{Q \rightarrow a R}{P \parallel A Q \rightarrow P \parallel A Q'} \\
\text{(PAR2)} \quad \frac{P \rightarrow a P'}{P \parallel A Q \rightarrow a Q'} \\
\text{(RPF)} \quad \frac{a.P \rightarrow P}{\lambda.P \rightarrow P} \\
\text{(CHO3)} \quad \frac{P \rightarrow a R}{P + Q \rightarrow a R} \\
\text{(CHO4)} \quad \frac{Q \rightarrow a R}{P + Q \rightarrow a R} \\
\text{(CON2)} \quad \frac{P \parallel A Q \rightarrow a P'}{P \parallel A Q \rightarrow P \parallel A Q'}
\]

Figure 5: Standard SOS rules for IML.
**Definition 10.** An equivalence relation $R \subseteq \mathcal{P}_{\text{IML}} \times \mathcal{P}_{\text{IML}}$ is called a strong bisimulation for IML if, for all $P_1, P_2 \in \mathcal{P}_{\text{IML}}$ it holds that

- for all $a \in \mathcal{A}$ and $Q \in \mathcal{P}_{\text{IML}}$: $T(P_1, a, [Q]_R) \iff T(P_2, a, [Q]_R)$
- for all $Q \in \mathcal{P}_{\text{IML}}$: $R(P_1, [Q]_R) = R(P_2, [Q]_R)$.

for all $P_1, P_2 \in \mathcal{P}_{\text{IML}}$ such that $R(P_1, P_2)$. Two processes $P_1, P_2 \in \mathcal{P}_{\text{IML}}$ are called strongly bisimilar if $R(P_1, P_2)$ for a strong bisimulation $R$ for IML, notation $P_1 =_{\text{IML}} P_2$.

To establish the correspondence of FuTS bisimilarity $\sim_{\text{iml}}$ for $S_{\text{iml}}$ of Definition 9 and strong bisimilarity $=_{\text{iml}}$ for IML, we need to connect the state-to-function relation $\mapsto_1$ and the transition relation $\rightarrow$ as well as the state-to-function relation $\mapsto_2$ and the transition relation $\rightarrow$.  

**Lemma 11.**

(a) Let $P \in \mathcal{P}_{\text{IML}}$ and $a \in \mathcal{A}$. If $P \xrightarrow{a_1} \mathcal{P}$ then $P \xrightarrow{a} P' \iff \mathcal{P}(P') = \text{true}$.

(b) Let $P \in \mathcal{P}_{\text{IML}}$. If $P \xrightarrow{\delta_2} \mathcal{P}$ then $\sum \| \lambda | P \xrightarrow{\lambda} P' \|= \mathcal{P}(P')$. □

We are now in a position to relate FuTS bisimulation and standard strong bisimulation for IML.

**Theorem 12.** For any two processes $P_1, P_2 \in \mathcal{P}_{\text{IML}}$ it holds that $P_1 \sim_{\text{iml}} P_2$ iff $P_1 =_{\text{IML}} P_2$.

**Proof.** Let $R$ be an equivalence relation on $\mathcal{P}_{\text{IML}}$. Pick $P \in \mathcal{P}_{\text{IML}}$, $a \in \mathcal{A}$ and choose any $Q \in \mathcal{P}_{\text{IML}}$. Suppose $P \xrightarrow{a} \mathcal{P}$. Thus $\theta_1(P)(a) = \mathcal{P}$. Then we have

$$T(P, a, [Q]_R) \iff \exists Q' \in [Q]_R: P \xrightarrow{a} Q' \quad \text{(by definition of $T$)}$$
$$\iff \exists Q' \in [Q]_R: \mathcal{P}(Q') = \text{true} \quad \text{(by Lemma 11a)}$$
$$\iff \sum_{Q' \in [Q]_R} \theta_1(P)(a)(Q) = \text{true} \quad \text{(by definition of $\theta_1$)}$$

Note, summation in $\mathbb{B}$ is disjunction. Likewise, on the quantitative side, we have

$$R(P, [Q]_R) = \sum_{Q' \in [Q]_R} \sum \| \lambda | P \xrightarrow{\lambda} Q' \| \quad \text{(by definition of $R$)}$$
$$= \sum_{Q' \in [Q]_R} \mathcal{P}(Q') \quad \text{(by Lemma 11b)}$$
$$= \sum_{Q' \in [Q]_R} \theta_2(P)(\delta)(Q) \quad \text{(by definition of $\theta_2$)}$$

Combining the equations, we conclude that a strong bisimulation for IML is also a bisimulation for the FuTS $S_{\text{iml}}$, and vice versa. From this the theorem follows. □

Again, as a corollary of the theorem above, we have for IML that its notion of strong bisimulation is coalgebraically underpinned, as it coincides, calling to Theorem 5 once more, with behavioral equivalence of the functor $\mathcal{V}_{\text{iml}}$ induced by the FuTS $S_{\text{iml}}$. As a consequence, the standard, FuTS and coalgebraic semantics for IML are all equal.

**7 Concluding remarks**

Total and deterministic labeled state-to-function transition systems, FuTS, are a convenient instrument to express the operational semantics of both qualitative and quantitative process languages. In this paper
we have introduced the notion of bisimulation that arises from a \textit{FuTS}, possibly involving multiple transition relations. A correspondence result, Theorem 5, relates the bisimulation of a \textit{FuTS} \( S \) to behavioral equivalence of the functor \( \mathcal{V}_S \) that arises from the \textit{FuTS} \( S \) too. For two prototypical stochastic process languages based on \textit{PEPA} and on \textit{IMC} we have shown that the notion of stochastic bisimulation associated with these calculi, coincides with the notion of bisimulation of the corresponding \textit{FuTS}. Using these \textit{FuTS} as a stepping stone, the correspondence result bridges between the concrete notion of bisimulation for \textit{PEPA} and \textit{IMC}, and the coalgebraic notion of behavioral equivalence. Hence, from this perspective, the concrete notions are seen as the natural strong equivalence to consider.

It is shown in [5], in the context of weighted automata, that in general the type of functors \( \mathcal{F}_S(\cdot, \mathcal{R}) \) may not preserve weak pullbacks and, therefore, the notions of coalgebraic bisimulation and of behavioral equivalence may not coincide. Essential for the construction in their setting is the fact that the sum of non-zero weights may add to weight 0. The same phenomenon prevents a general proof, along the lines of [28], for coalgebraic bisimulation and \textit{FuTS} bisimulation to coincide. In the construction of a mediating morphism, going from \textit{FuTS} bisimulation to coalgebraic bisimulation a denominator may be zero, hence a division undefined, in case the sum over an equivalence class cancels out. In the concrete case for [19], although no detailed proof is provided there, this will not happen with \( \mathbb{R}_{\geq 0} \) as underlying semiring. We expect that for semirings enjoying the property that for a sum \( x = \sum x_i \) it holds that \( x = 0 \) iff \( x_i = 0 \) for all \( i \in \{1, \ldots, n\} \), we will be able to prove that pullbacks are weakly preserved, and hence that coalgebraic bisimulation and behavioral equivalence are the same.

Obviously, Milner-type strong bisimulation [21, 22] and bisimulation for \textit{FuTS} over \( \mathbb{B} \) coincide. Also, strong bisimulation of [17] involving, apart from the usual transfer conditions, the comparison of state information, viz. the apparent rates, can be treated with \textit{FuTS}. Again the two notions of equivalence coincide. We expect to be able to deal with discrete time and so-called Markov automata as well. For dense time and general measures one may speculate that the use of functions of compact support with respect to a suitable topology may be fruitful. Future research needs to reveal under what algebraic conditions of the semirings, or similar structures, or the coalgebraic conditions on the format of the functors involved standard bisimulation, \textit{FuTS}-bisimulation, coalgebraic bisimulation and behavioral equivalence will amount to similar identifications.

Acknowledgments The authors are grateful to Rocco De Nicola, Michele Loreti and Jan Rutten for fruitful discussions useful suggestions. DL and MM acknowledge support by EU Project n. 257414 Autonomic Service-Components Ensembles (ASCENS) and by CNR/RSTL Project XXL. This research has been conducted while EV was spending a sabbatical leave at the CNR/ISTI. EV gratefully acknowledges the hospitality and support during his stay in Pisa.

References


doi:10.1007/978-3-642-02930-1_36


Klaim-based mobile systems. In H. Haddad et al., editor: Proc. SAC 2005, ACM, pp. 428–435,  

15375-4


M.M. Bonsangue, S. Hallerstede & M. Leuschel, editors: Proc. FMCO 2009, LNCS 6286, pp. 311–337,  
doi:10.1007/978-3-642-17071-3_16.

Computer Science 12, Cambridge University Press.

doi:10.1109/LICS.2005.35.


LNCS 104, pp. 167–183.


A Additional proofs

Additional proof for Section 2

Lemma 1. Let $X$ be a set, $R$ a semiring and $| \cdot |$ an injective binary operation on $X$. For $\varphi, \psi \in FS(X, R)$ it holds that $\oplus(\varphi + \psi) = \oplus \varphi + \oplus \psi$ and $\oplus(\varphi \mid \psi) = (\oplus \varphi) \ast (\oplus \psi)$.

Proof. We verify holds that $\oplus$ for $\nu$ for $\eta$ the lifting $FS$.

The fact $\oplus(\varphi \mid \psi) = (\oplus \varphi) \ast (\oplus \psi)$ follows direct from the definitions and commutativity of $\oplus$. □

Additional proof for Section 4

Lemma 2. Let $L$ be a set of labels and $R$ a semiring. Then functor $V^L_R$ on $Set$ is bounded.

Proof. Consider the elements $\nu \in FS(\mathbb{N}, R)^L$ as parametrized ‘valuation’ functions, and the elements $\sigma \in X^\mathbb{N}$ as ‘selection’ functions. The functor $FS(\mathbb{N}, R)^L \times (\_)^\mathbb{N} : Set \to Set$ is the product functor of the lifting $id_{FS(\mathbb{N}, R)^L}$ of the identity functor $id_{FS(\mathbb{N}, R)}$ to $L$ and of the functor $(\_)^\mathbb{N}$. Define the mapping $\eta : id_{FS(\mathbb{N}, R)^L} \times (\_)^\mathbb{N} \to V^L_R$ by putting

$$\eta^L_X(\nu, \sigma)(\ell)(x) = \sum_{n \in \sigma^{-1}(x)} \nu(\ell)(n)$$

for $\nu \in FS(\mathbb{N}, R)^L$, $\sigma \in X^\mathbb{N}$ and $\ell \in L$. For $\ell \in L$ and $x \in X$, the right-hand sum defining $\eta^L_X(\nu, \sigma)(\ell)(x)$ exists, since $\nu(\ell) : \mathbb{N} \to R$ is of finite support. Note that $\eta^L_X(\nu, \sigma)(\ell)$ is of finite support too: If $\eta^L_X(\nu, \sigma)(\ell)(x) \neq 0$, by definition $\sum \nu(\ell)(n) n \in \sigma^{-1}(x) \neq 0$. Then $\nu(\ell)(n) \neq 0$ for some $n \in \sigma^{-1}(x)$. Thus, $\sigma(n) = x$ for some $n \in spt(\nu(\ell))$. So, $spt(\eta^L_X(\nu, \sigma)) \subseteq \{\sigma(n) \mid n \in spt(\nu(\ell))\}$ and $spt(\eta^L_X(\nu, \sigma))$ is finite.

Next we verify that $\eta : id_{FS(\mathbb{N}, R)^L} \times (\_)^\mathbb{N} \to V^L_R$ is a natural transformation, i.e. we check that for $f : X \to Y$ it holds that $V^L_Y \circ \eta^L_X = \eta^Y \circ (id_{FS(\mathbb{N}, R)^L} \times f^\mathbb{N})$.

For $\nu \in FS(\mathbb{N}, R)^L$ and $\sigma \in X^\mathbb{N}$ we have, for $\ell \in L$ and $y \in Y$,

$$(FS(f, R)^L \circ \eta^L_X)(\nu, \sigma)(\ell)(y) = \sum_{x \in f^{-1}(y)} \eta^L_X(\nu, \sigma)(\ell)(x) = \sum_{x \in f^{-1}(y)} \sum_{n \in \sigma^{-1}(x)} \nu(\ell)(n) = \eta^Y(\nu, f \circ \sigma)(\ell)(y) = \eta^Y((id_{FS(\mathbb{N}, R)^L} \times f^\mathbb{N})(\nu, \sigma))(\ell)(y) = (\eta^Y \circ (id_{FS(\mathbb{N}, R)^L} \times f^\mathbb{N}))(\nu, \sigma)(\ell)(y)$$
Thus, $\mathcal{FS}(f,R)^L \circ \eta_X = \eta_Y \circ (id_{\mathcal{FS}(X,R)^L} \times f^L)$ and $\eta : id_{\mathcal{FS}(X,R)^L} \times (\cdot)^L \rightarrow \mathcal{V}_R^L$ is a natural transformation.

Finally, we check that $\eta_X : id_{\mathcal{FS}(X,R)^L} \times X^L \rightarrow \mathcal{V}_R^L(X)$ is surjective. Choose a set $X$ and a mapping $\varphi : L \rightarrow \mathcal{FS}(X,R)$. Say, $spt(\varphi(\ell)) = \{x^i_0, \ldots, x^i_{n(\ell)}\}$. Without loss of generality we assume $X \neq spt(\varphi(\ell))$ and pick $x^i_0 \in X \setminus spt(\varphi(\ell))$. Define $\nu \in \mathcal{FS}(\mathbb{N}, R)^L$ by $\nu(\ell)(n) = \varphi(\ell)(x^i_n)$ for $n = 1 \ldots n(\ell)$ and $\nu(\ell)(n) = 0$ otherwise. Define $\sigma : \mathbb{N} \rightarrow X$ by $\sigma(n) = x^i_n$ for $1 \leq n \leq n(\ell)$ and $\sigma(n) = x^i_0$ otherwise. Then we have

$$
\eta_X(\nu, \sigma)(\ell)(x^i_n) = \sum_{m \in \sigma^{-1}(x^i_n)} \nu(\ell)(m) = \nu(\ell)(i) = \varphi(\ell)(x^i_n)
$$

for $i = 1 \ldots n(\ell)$ and $x \notin spt(\nu(\ell))$. Thus $\eta_X(\nu, \sigma)(\ell)(x) = \varphi(\ell)(x)$ for all $\ell \in L$ and $x \in X$, $\eta_X(\nu, \sigma) = \varphi$ and $\eta_X : id_{\mathcal{FS}(X,R)^L} \times X^L \rightarrow \mathcal{V}_R^L(X)$ is surjective.

### Additional proofs for Section 5

**Definition.**[[17] Definition 3.3.1] We put

$$
\begin{align*}
\rho_a(\text{nil}) &= 0, & \rho_a(P + Q) &= \rho_a(P) + \rho_a(Q), \\
\rho_a((a, \lambda).P) &= \lambda, & \rho_a(P ||_A Q) &= \rho_a(P) + \rho_a(Q), & \text{if } a \notin A, \\
\rho_a((b, \lambda).P) &= 0, & \rho_a(P ||_A Q) &= \min\{\rho_a(P), \rho_a(Q)\}, & \text{if } a \in A, \\
\rho_a(X) &= \rho_a(P), & \text{if } X := P
\end{align*}
$$

**Lemma**[[4] Let $P \in \mathcal{P}_{PEPA}$ and $a \in A$. Suppose $\delta_a \rightarrow_P P$. Then $\oplus P = \rho_a(P)$.

**Proof.** Guarded recursion. We treat the two cases for the parallel construct.

**Case $P = P_1 ||_A P_2$, $a \notin A$.** Suppose $P_1 \rightarrow_P P_1$, $P_2 \rightarrow_P P_2$. Then $P = (P_1 ||_A P_2) + (X_{P_1} ||_A P_2)$. Therefore we have

$$
\begin{align*}
\oplus P &= \oplus (P_1 ||_A X_{P_2}) + (X_{P_1} ||_A P_2) \\
&= \oplus (P_1 ||_A X_{P_2}) + \oplus (X_{P_1} ||_A P_2) & \text{(by Lemma[[1])} \\
&= (\oplus P_1 + \oplus X_{P_2}) + (\oplus X_{P_1} + \oplus P_2) & \text{(by Lemma[[1])} \\
&= \oplus P_1 + \oplus P_2 & \text{(since $\oplus X_{P_1}, \oplus X_{P_2} = 1$) \\
&= \rho_a(P_1) + \rho_a(P_2) & \text{(by the induction hypothesis)} \\
&= \rho_a(P_1 ||_A P_2) & \text{(by definition $\rho_a$)}
\end{align*}
$$

**Case $P = P_1 ||_A P_2$, $a \in A$.** Suppose $P_1 \rightarrow_P P_1$, $P_2 \rightarrow_P P_2$. Then $P = arf(P_1, P_2) \cdot (P_1 ||_A P_2)$. If $\oplus P_1, \oplus P_2 > 0$ we have

$$
\begin{align*}
\oplus P &= \oplus (arf(P_1, P_2) \cdot (P_1 ||_A P_2)) \\
&= arf(P_1, P_2) \cdot \oplus P_1 + \oplus P_2 & \text{(by Lemma[[1])} \\
&= \frac{\min\{\oplus P_1, \oplus P_2\}}{\oplus P_1 + \oplus P_2} \cdot \oplus P_1 + \oplus P_2 & \text{(by definition of arf)} \\
&= \min\{\oplus P_1, \oplus P_2\} & \text{(by the induction hypothesis)} \\
&= \min\{\rho_a(P_1), \rho_a(P_2)\} & \text{(by the definition of arf)} \\
&= \rho_a(P_1 ||_A P_2) & \text{(by definition $\rho_a$)}
\end{align*}
$$

**Remark.** This construction is a natural refinement of the sinked transitions of PEPA.
If \( \oplus \mathcal{P}_1, \oplus \mathcal{P}_2 = 0 \), then \( \text{arf}(\mathcal{P}_1, \mathcal{P}_2) = 0 \), by definition, and \( r_a(P_1), r_a(P_2) = 0 \), by induction hypothesis. Therefore we have \( \oplus \mathcal{P} = \text{arf}(\mathcal{P}_1, \mathcal{P}_2) \cdot \oplus \mathcal{P}_1, \oplus \mathcal{P}_2 = 0 \) as well as \( \text{arf}(\mathcal{P}_1, P_2) = \min \{ r_a(P_1), r_a(P_2) \} = 0 \). So, also now, \( \oplus \mathcal{P} = r_a(P_1 \parallel A P_2) \). The other cases are straightforward, in the case of \( P_1 + P_2 \) also relying on Lemma [1].

**Corollary.** If \( P \rightsquigarrow \mathcal{P} \) and \( Q \rightsquigarrow Q \), then \( \text{arf}(\mathcal{P}, \mathcal{Q}) = \text{arf}(P, Q) \).

**Proof.** Direct from the definitions. □

**Lemma 8.** Let \( P \in \mathcal{P}_{\text{PEPA}} \) and \( a \in \mathcal{A} \). Suppose \( P \rightsquigarrow \mathcal{P} \). Then it holds that \( \mathcal{P}(P') = \sum \parallel \mathcal{A} \parallel_a a \rightarrow P' \parallel \) for all \( P' \in \mathcal{P}_{\text{PEPA}} \).

**Proof.** Guarded induction on \( P \). We only treat the cases for the parallel composition. Note, the operation \( \parallel_a : \mathcal{P}_{\text{PEPA}} \times \mathcal{P}_{\text{PEPA}} \rightarrow \mathcal{P}_{\text{PEPA}} \) with \( \parallel_a(P_1, P_2) = P_1 \parallel_a P_2 \) is injective. Recall, for \( \mathcal{P}_1, \mathcal{P}_2 \in \mathcal{F} S(\mathcal{P}_{\text{PEPA}}, \mathbb{R} \geq 0) \), we have \( (\mathcal{P}_1 \parallel_a \mathcal{P}_2)(P_1 \parallel_a P_2) = \mathcal{P}_1(P_1) \cdot \mathcal{P}_2(P_2) \).

Suppose \( a \notin \mathcal{A} \). Assume \( P \rightsquigarrow \mathcal{P}_1, P_2 \rightsquigarrow \mathcal{P}_2, P_1 \parallel_a P_2 \rightsquigarrow \mathcal{P} \). We distinguish three cases. Case (I), \( P' = P_1' \parallel_a P_2, P_1' \neq P_1 \). Then we have

\[
\sum \parallel \mathcal{A} \parallel_a P_2 a \rightarrow P' \parallel \\
= \sum \parallel \mathcal{A} \parallel_a P_1 a \rightarrow P_1' \parallel \quad \text{(by rule (PAR1a))} \\
= \mathcal{P}_1(P_1') \quad \text{(by the induction hypothesis)} \\
= \mathcal{P}_1(P_1') \cdot X_{P_2}(P_2) \quad \text{(as } X_{P_2}(P_2) = 1) \\
= (\mathcal{P}_1 \parallel_a X_{P_2})(P_1' \parallel_a P_2) + (X_{P_1} \parallel_a \mathcal{P}_2)(P_1' \parallel_a P_2) \\
\text{(definition } \parallel_a \text{ on } \mathcal{F} S(\mathcal{P}_{\text{PEPA}}, \mathbb{R} \geq 0), X_{P_1}(P_1') = 0) \\
= \mathcal{P}(P') \\
\text{(by rule (PAR1))}
\]

Case (II), \( P' = P_1 \parallel_a P_2, P_2' \neq P_2 \): similar. Case (III), \( P' = P_1' \parallel_a P_2 \). Then we have

\[
\sum \parallel \mathcal{A} \parallel_a P_2 a \rightarrow P' \parallel \\
= \left( \sum \parallel \mathcal{A} \parallel_a P_1 a \rightarrow P_1 \parallel \right) + \left( \sum \parallel \mathcal{A} \parallel_a P_2 a \rightarrow P_2 \parallel \right) \quad \text{(by rules (PAR1a) and (PAR1b))} \\
= \mathcal{P}_1(P_1) + \mathcal{P}_2(P_2) \quad \text{(by the induction hypothesis)} \\
= (\mathcal{P}_1 \parallel_a X_{P_2})(P_1 \parallel_a P_2) + (X_{P_1} \parallel_a \mathcal{P}_2)(P_1 \parallel_a P_2) \\
\text{(definition } \parallel_a \text{ on } \mathcal{F} S(\mathcal{P}_{\text{PEPA}}, \mathbb{R} \geq 0), X_{P_1}(P_1), X_{P_2}(P_2) = 1) \\
= \mathcal{P}(P') \\
\text{(again by rule (PAR1))}
\]

Suppose \( a \in \mathcal{A} \). Assume \( P_1 \rightsquigarrow \mathcal{P}_1, P_2 \rightsquigarrow \mathcal{P}_2, P_1 \parallel_a P_2 \rightsquigarrow \mathcal{P} \). Without loss of generality, \( P' = P_1' \parallel_a P_2' \) for suitable \( P_1', P_2' \in \mathcal{P}_{\text{PEPA}} \).
\[
\sum \| \lambda \cdot P_1 \rightarrow_A P_2 \overset{a}{\rightarrow} P' \|
\]

\[
= \sum \| \text{arf}(P_1, P_2) \cdot \lambda \cdot \lambda_2 \rightarrow_P P_1 \rightarrow_P P_2 \overset{a}{\rightarrow} P_2 \rightarrow_P P' \|
\text{ (by rule (PAR2))}
\]

\[
= \text{arf}(P_1, P_2) \cdot \left( \sum \| \lambda \rightarrow_P P_1 \rightarrow_P P_2 \| \cdot \left( \sum \| \lambda_2 \rightarrow_P P_2 \rightarrow_P P' \| \right) \right)
\text{ (by distributivity)}
\]

\[
= \text{arf}(P_1, P_2) \cdot \mathcal{P}(P_1) \cdot \mathcal{P}(P_2)
\text{ (by the induction hypothesis)}
\]

\[
= \text{arf}(P_1, P_2) \cdot \mathcal{P}(P_1 \rightarrow_A P_2)(P' \rightarrow_A P_2)
\text{ (definition } \|_A \text{ on } \mathcal{F}(\mathcal{P}_{\text{PEA}}, \mathbb{B}))
\]

\[
= \mathcal{P}(P')
\text{ (by rule (PAR2))}
\]

The other cases are simpler and omitted here.

□

**Additional proofs for Section 6**

**Lemma 11**

(a) Let \(P \in \mathcal{P}_{\text{IML}}\) and \(a \in \mathcal{A}\). If \(P \overset{a}{\rightarrow} \mathcal{P}\) then \(P \overset{a}{\rightarrow} P' \iff \mathcal{P}(P') = \text{true}\).

(b) Let \(P \in \mathcal{P}_{\text{IML}}\). If \(P \overset{\delta}{\rightarrow} \mathcal{P}\) then \(\sum \| \lambda \rightarrow_P \| = \mathcal{P}(P')\).

**Proof.** (a) Guarded induction. Let \(a \in \mathcal{A}\). We treat the typical cases \(\lambda \cdot P\) and \(P_1 \rightarrow_A P_2\) for \(a \notin A\).

Case \(\lambda \cdot P\). Suppose \(\lambda \cdot P \overset{a}{\rightarrow} \mathcal{P}\). Then we have \(\mathcal{P} = \|_B\). Thus, both \(\lambda \cdot P \overset{a}{\rightarrow} P'\) for no \(P' \in \mathcal{P}_{\text{IML}}\), as no transition is provided in \(\overset{a}{\rightarrow}\), and \(\mathcal{P}(P') = \text{false}\) by definition of \(\|_B\), for all \(P' \in \mathcal{P}_{\text{IML}}\).

Case \(P_1 \rightarrow_A P_2\), \(a \notin A\). Suppose \(P_1 \overset{a}{\rightarrow} \mathcal{P}_1\). Then it holds that \(\mathcal{P} = (\mathcal{P}_1 \rightarrow_A X_{P_1}) + (X_{P_1} \rightarrow_A \mathcal{P}_2)\). Recall, for \(Q \in \mathcal{P}_{\text{IML}}\) and \(X_Q \in \mathcal{F}(\mathcal{P}_{\text{IML}}, \mathbb{B})\), \(X_Q(Q') = \text{true}\) iff \(Q' = Q\), for \(Q' \in \mathcal{P}_{\text{IML}}\). We have

\[
P_1 \rightarrow_A P_2 \overset{a}{\rightarrow} P' \iff (P_1 \overset{a}{\rightarrow} P'_1 \land P' = P'_1 \rightarrow_A P_2) \lor (P_2 \overset{a}{\rightarrow} P'_2 \land P' = P_1 \rightarrow_A P'_2)
\text{ (by analysis of } \overset{a}{\rightarrow}\)
\]

\[
\iff (\mathcal{P}_1(P'_1) = \text{true} \land P' = P'_1 \rightarrow_A P_2) \lor (\mathcal{P}_2(P'_2) = \text{true} \land P' = P_1 \rightarrow_A P'_2)
\text{ (by the induction hypothesis)}
\]

\[
\iff (\mathcal{P}_1(P'_1) \cdot X_{P_1}(P'_2) = \text{true} \land P' = P'_1 \rightarrow_A P_2) \lor (X_{P_1}(P'_1) \cdot \mathcal{P}_2(P'_2) = \text{true} \land P' = P_1 \rightarrow_A P'_2)
\text{ (by definition of } X_{P_1} \text{ and } X_{P_2})
\]

\[
\iff ((\mathcal{P}_1 \cdot A X_{P_1})(P'_1 \rightarrow_A P_2) = \text{true} \land P' = P'_1 \rightarrow_A P_2) \lor ((X_{P_1} \cdot A \mathcal{P}_2)(P'_1 \rightarrow_A P_2) = \text{true} \land P' = P_1 \rightarrow_A P'_2)
\text{ (by definition of } \|_A\)
\]

\[
\iff (\mathcal{P}_1 \cdot A X_{P_2})(P') = \text{true} \lor (X_{P_1} \cdot A \mathcal{P}_2)(P') = \text{true}
\text{ (by definition of } \|_A, X_{P_1} \text{ and } X_{P_2})
\]

\[
= (\mathcal{P}_1 \cdot A X_{P_2} + (X_{P_1} \cdot A \mathcal{P}_2))(P') = \text{true}
\text{ (by definition of } + \text{ on } \mathcal{F}(\mathcal{P}_{\text{IML}}, \mathbb{B}))
\]

\[
\mathcal{P}(P') = \text{true}
\]
The other cases are standard or similar and easier.

(b) Guarded induction. We treat the cases for $\mu.P$ and $P_1 \parallel_A P_2$. Case $\mu.P$. Assume $P \xrightarrow{\delta} \mathcal{P}$. Suppose $P = \mu.P'$. Then it holds that $P$ admits a single $\rightarrow$-transition, viz. $P \xrightarrow{\mu} P'$. Thus we have $\sum \| \lambda \mid P \xrightarrow{\lambda} P' \| = \mu = \| P' \mapsto \mu \| (P') = \mathcal{P}(P')$. Suppose $P = \mu.P''$ for some $P'' \neq P$. Then we have $\sum \| \lambda \mid P \xrightarrow{\lambda} P' \| = 0 = \| P'' \mapsto \mu \| (P') = \mathcal{P}(P')$.

Case $P_1 \parallel_A P_2$. Assume $P_1 \xrightarrow{\delta} \mathcal{P}_1$, $P_2 \xrightarrow{\delta} \mathcal{P}_2$ and $P_1 \parallel_A P_2 \xrightarrow{\delta} \mathcal{P}$. It holds that $\mathcal{P} = (\mathcal{P}_1 \parallel_A X_{P_2}) + (X_{P_1} \parallel_A \mathcal{P}_2)$. We calculate

$$
\begin{align*}
\sum \| \lambda \mid P_1 \parallel_A P_2 \xrightarrow{\lambda} P' \| &= \sum \| \lambda \mid P_1 \xrightarrow{\lambda} P'_1, P' = P'_1 \parallel_A P_2 \| + \sum \| \lambda \mid P_2 \xrightarrow{\lambda} P'_2, P' = P_1 \parallel_A P'_2 \| \\
&(\text{by analysis of } \rightarrow) \\
&= (\text{if } P' = P'_1 \parallel_A P_2 \text{ then } \sum \| \lambda \mid P_1 \xrightarrow{\lambda} P'_1 \| \text{ else 0 end}) + \\
&(\text{if } P' = P_1 \parallel_A P'_2 \text{ then } \sum \| \lambda \mid P_2 \xrightarrow{\lambda} P'_2 \| \text{ else 0 end}) \\
&= (\text{if } P' = P'_1 \parallel_A P_2 \text{ then } \mathcal{P}_1(P'_1) \text{ else 0 end}) + \\
&(\text{if } P' = P_1 \parallel_A P'_2 \text{ then } \mathcal{P}_2(P'_2) \text{ else 0 end}) \\
&(\text{by induction hypothesis for } P_1 \text{ and } P_2) \\
&= (\mathcal{P}_1 \parallel_A X_{P_2})(P') + (X_{P_1} \parallel_A \mathcal{P}_2)(P') \\
&(\text{by definition of } \parallel_A, X_{P_1}, X_{P_2} \text{ and } + \text{ on } \mathcal{FS}(\mathcal{P}_{\text{IML}}, \mathbb{R}_{\geq 0})) \\
&= \mathcal{P}(P')
\end{align*}
$$

The remaining cases are left to the reader. \qed
Decorated proofs for computational effects: States

Jean-Guillaume Dumas∗ Dominique Duval† Laurent Fousse Jean-Claude Reynaud
LJK, Université de Grenoble, France.


Abstract. The syntax of an imperative language does not mention explicitly the state, while its denotational semantics has to mention it. In this paper we show that the equational proofs about an imperative language may hide the state, in the same way as the syntax does.

Introduction

The evolution of the state of the memory in an imperative program is a computational effect: the state is never mentioned as an argument or a result of a command, whereas in general it is used and modified during the execution of commands. Thus, the syntax of an imperative language does not mention explicitly the state, while its denotational semantics has to mention it. This means that the state is encapsulated: its interface, which is made of the functions for looking up and updating the values of the locations, is separated from its implementation; the state cannot be accessed in any other way than through its interface. In this paper we show that equational proofs in an imperative language may also encapsulate the state: proofs can be performed without any knowledge of the implementation of the state. We will see that a naive approach (called “apparent”) cannot deal with the updating of states, while this becomes possible with a slightly more sophisticated approach (called “decorated”). This is expressed in an algebraic framework relying on category theory. To our knowledge, the first categorical treatment of computational effects, using monads, is due to Moggi [Moggi 1991]. The examples proposed by Moggi include the side-effects monad \( T(A) = (A \times S)^S \) where \( S \) is the set of states. Later on, Plotkin and Power used Lawvere theories for dealing with the operations and equations related to computational effects. The Lawvere theory for the side-effects monad involves seven equations [Plotkin & Power 2002]. In Section 1 we describe the intended denotational semantics of states. Then in Section 2 we introduce three variants of the equational logic for formalizing the computational effects due to the states: the apparent, decorated an explicit logics. This approach is illustrated in Section 3 by proving some of the equations from [Plotkin & Power 2002], using rules which do not mention any type of states.

1 Motivations

This section is made of three independent parts. Section 1.1 is devoted to the semantics of states, an example is presented in Section 1.2 and our logical framework is described in Section 1.3.

∗This work is partly funded by the project HPAC of the French Agence Nationale de la Recherche (ANR 11 BS02 013).
†This work is partly funded by the project CLIMT of the French Agence Nationale de la Recherche (ANR 11 BS02 016).
1.1 Semantics of states

This section deals with the denotational semantics of states, by providing a set-valued interpretation of the \textit{lookup} and \textit{update} operations. Let $St$ denote the set of states. Let $Loc$ denote the set of locations (also called variables or identifiers). For each location $i$, let $Val_i$ denote the set of possible values for $i$.

For each location $i$ there is a \textit{lookup} function for reading the value of location $i$ in the given state, without modifying this state: this corresponds to a function $\text{lookup}_{i,1} : St \to Val_i$ or equivalently to a function $\text{lookup}_i : St \to Val_i \times St$ such that $\text{lookup}_i(s) = \langle \text{lookup}_{i,1}(s), s \rangle$ for each state $s$. In addition, for each location $i$ there is an \textit{update} function $\text{update}_i : Val_i \times St \to St$ for setting the value of location $i$ to the given value, without modifying the values of the other locations in the given state. This is summarized as follows, for each $i \in Loc$: a set $Val_i$, two functions $\text{lookup}_{i,1} : St \to Val_i$ and $\text{update}_i : Val_i \times St \to St$, and equations (1):

\begin{equation}
\forall a \in Val_i, \forall s \in St, \text{lookup}_{i,1}(\text{update}_i(a, s)) = a,
\end{equation}

\begin{equation}
\forall a \in Val_i, \forall s \in St, \text{lookup}_{j,1}(\text{update}_i(a, s)) = \text{lookup}_{j,1}(s) \text{ for every } j \in Loc, j \neq i.
\end{equation}

The state can be observed thanks to the lookup functions. We may consider the tuple $\langle \text{lookup}_{i,1} \rangle_{i \in Loc} : St \to \prod_{i \in Loc} Val_i$. If this function is an isomorphism, then Equations (1) provide a definition of the update functions. In [Plotkin & Power 2002] an equational presentation of states is given, with seven equations: in Remark [1.1] these equations are expressed according to [Mellies 2010] and they are translated in our framework. We use the notations $l_i = \text{lookup}_i : St \to Val_i \times St$, $l_{i,1} = \text{lookup}_{i,1} : St \to Val_i$ and $u_i = \text{update}_i : Val_i \times St \to St$, and in addition $id_i : Val_i \to Val_i$ and $q_i : Val_i \times St \to St$ respectively denote the identity of $Val_i$ and the projection, while $\text{perm}_{i,j} : Val_i \times Val_i \times St \to Val_i \times Val_i \times St$ permutes its first and second arguments.

\textbf{Remark 1.1.} The equations in [Plotkin & Power 2002] can be expressed as the following Equations (2):

\begin{equation}
\forall i \in Loc, \forall s \in St, u_i(l_i(s)) = s \in St
\end{equation}

\begin{equation}
\forall i \in Loc, \forall s \in St, l_i(q_i(l_i(s))) = l_i(s) \in Val_i \times St
\end{equation}

\begin{equation}
\forall i \in Loc, \forall s \in St, \forall a, a' \in Val_i, u_i(a', u_i(a, s)) = u_i(a', s) \in St
\end{equation}

\begin{equation}
\forall i \in Loc, \forall s \in St, l_{i,1}(u_i(a, s)) = a \in Val_i
\end{equation}

\begin{equation}
\forall i \neq j \in Loc, \forall s \in St, (id_i \times l_j)(l_i(s)) = \text{perm}_{i,j}((id_j \times l_i)(l_j(s))) \in Val_i \times Val_j \times St
\end{equation}

\begin{equation}
\forall i \neq j \in Loc, \forall s \in St, \forall a \in Val_i, \forall b \in Val_j, u_j(b, u_i(a, s)) = u_i(a, u_j(b, s)) \in St
\end{equation}

\begin{equation}
\forall i \neq j \in Loc, \forall s \in St, \forall a \in Val_i, l_j(u_i(a, s)) = (id_j \times u_i)(\text{perm}_{j,i}(a, l_i(s))) \in Val_j \times St
\end{equation}

\textbf{Proposition 1.2.} Let us assume that $\langle l_{i,1} \rangle_{i \in Loc} : St \to \prod_{i \in Loc} Val_i$ is invertible. Then Equations (1) are equivalent to Equations (2).
Proof. It may be observed that (2.4) is exactly (1.1). In addition, (2.7) is equivalent to (1.2): indeed, (2.7) is equivalent to the conjunction of its projection on \( \text{Val}_j \) and its projection on \( \text{St} \); the first one is \( l_{j,1}(u_i(a,s)) = l_{j,1}(s) \), which is (1.2), and the second one is \( u_i(a,s) = u_i(a,s) \). Equations (2.2) and (2.5) follow from \( q_i(l_i(s)) = s \). For the remaining equations (2.1), (2.3) and (2.6), which return states, it is easy to check that for each location \( k \), by applying \( l_k \) to both members and using equation (1.1) or (1.2) according to \( k \), we get the same value in \( \text{Val}_k \) for both hand-sides. Then equations (2.1), (2.3) and (2.6) follow from the fact that \( \langle l_{i,1} \rangle_{i \in \text{Loc}} : \text{St} \to \prod_{i \in \text{Loc}} \text{Val}_i \) is invertible.

Proposition 1.2 will be revisited in Section 3, where it will be proved that equations (1) imply equations (2) without ever mentioning explicitly the state in the proof.

1.2 Computational effects: an example

In an informal way, we consider that a computational effect occurs when there is an apparent mismatch, i.e., some lack of soundness, between the syntax and the denotational semantics of a language. For instance in an object-oriented language, the state of an object does not appear explicitly as an argument nor as a result of any of its methods. In this section, as a toy example, we build a class `BankAccount` for managing (very simple!) bank accounts. We use the types `int` and `void`, and we assume that `int` is interpreted by the set of integers \( \mathbb{Z} \) and `void` by a singleton \( \{\star\} \). In the class `BankAccount`, there is a method `balance()` which returns the current balance of the account and a method `deposit(x)` for the deposit of \( x \) Euros on the account. The `deposit` method is a modifier, which means that it can use and modify the state of the current account. The `balance` method is an inspector, or an accessor, which means that it can use the state of the current account but it is not allowed to modify this state. In the object-oriented language C++, a method is called a member function; by default a member function is a modifier, when it is an accessor it is called a constant member function and the keyword `const` is used.

So, the C++ syntax for declaring the member functions of the class `BankAccount` looks like:

```cpp
int balance() const;
void deposit(int);
```

- Forgetting the keyword `const`, this piece of C++ syntax can be translated as a signature `Bank_{app}`, which we call the apparent signature (we use the word “apparent” in the sense of “seeming” i.e., “appearing as such but not necessarily so”).

\[
\text{Bank}_{\text{app}} : \begin{cases}
\text{balance} : \emptyset \to \mathbb{Z} \\
\text{deposit} : \mathbb{Z} \times \emptyset \to \emptyset
\end{cases}
\]

In a model (or algebra) of the signature `Bank_{app}`, the operations would be interpreted as functions:

\[
\left\{ \begin{array}{l}
[\text{balance}] : \emptyset \to \mathbb{Z} \\
[\text{deposit}] : \mathbb{Z} \to \emptyset
\end{array} \right.
\]

which clearly is not the intended interpretation.

- In order to get the right semantics, we may use another signature `Bank_{expl}`, which we call the explicit signature, with a new symbol `state` for the “type of states”:

\[
\text{Bank}_{\text{expl}} : \begin{cases}
\text{balance} : \text{state} \to \mathbb{Z} \\
\text{deposit} : \mathbb{Z} \times \text{state} \to \text{state}
\end{cases}
\]
The intended interpretation is a model of the explicit signature $Bank_{expl}$, with $St$ denoting the set of states of a bank account:

\[
\begin{align*}
[[\text{balance}]] &: St \to \mathbb{Z} \\
[[\text{deposit}]] &: \mathbb{Z} \times St \to St
\end{align*}
\]

So far, in this example, we have considered two different signatures. On the one hand, the apparent signature $Bank_{app}$ is simple and quite close to the C++ code, but the intended semantics is not a model of $Bank_{app}$. On the other hand, the semantics is a model of the explicit signature $Bank_{expl}$, but $Bank_{expl}$ is far from the C++ syntax: actually, the very nature of the object-oriented language is lost by introducing a “type of states”. Let us now define a *decorated signature* $Bank_{deco}$, which is still closer to the C++ code than the apparent signature and which has a model corresponding to the intended semantics. The decorated signature is not exactly a signature in the classical sense, because there is a classification of its operations. This classification is provided by superscripts called *decorations*: the decorations (1) and (2) correspond respectively to the object-oriented notions of accessor and modifier.

\[
Bank_{deco} : \begin{cases}
  \text{balance}(1) &: \text{void} \to \text{int} \\
  \text{deposit}(2) &: \text{int} \to \text{void}
\end{cases}
\]

The decorated signature is similar to the C++ code, with the decoration (1) corresponding to the keyword `const`. The apparent specification $Bank_{app}$ may be recovered from $Bank_{deco}$ by dropping the decorations. In addition, we claim that the intended semantics can be seen as a *decorated model* of this decorated signature: this will become clear in Section 2.3. In order to add to the signature constants of type `int` like 0, 1, 2, ... and the usual operations on integers, a third decoration is used: the decoration (0) for *pure* functions, which means, for functions which neither inspect nor modify the state of the bank account. So, we add to the apparent and explicit signatures the constants 0, 1, ... : `void` → `int` and the operations $+, -,* : \text{int} \times \text{int} \to \text{int}$, and we add to the decorated signature the pure constants $0^{(0)}, 1^{(0)}, ... : \text{void} \to \text{int}$ and the pure operations $+^{(0)}, -^{(0)}, *^{(0)} : \text{int} \times \text{int} \to \text{int}$. For instance the C++ expressions deposit(7); balance() and 7 + balance() can be seen as the decorated terms:

\[
\text{balance}(1) \circ \text{deposit}(2) \circ 7^{(0)} \quad \text{and} \quad +^{(0)} \circ \langle 7^{(0)}, \text{balance}(1) \rangle
\]

which may be illustrated as:

\[
\begin{align*}
\text{void} & \xrightarrow{7^{(0)}} \text{int} \\
& \xrightarrow{\text{deposit}(2)} \text{void} \\
& \xrightarrow{\text{balance}(1)} \text{int}
\end{align*}
\]

and

\[
\begin{align*}
\text{void} & \xrightarrow{\langle 7^{(0)}, \text{balance}(1) \rangle} \text{int} \times \text{int} \\
& \xrightarrow{+^{(0)}} \text{int}
\end{align*}
\]

These two decorated terms have different effects: the first one does modify the state while the second one is an accessor; however, both return the same integer. Let us introduce the symbol $\sim$ for the relation “same result, maybe distinct effects”. Then:

\[
\text{balance}(1) \circ \text{deposit}(2) \circ 7^{(0)} \sim +^{(0)} \circ \langle 7^{(0)}, \text{balance}(1) \rangle
\]

### 1.3 Diagrammatic logics

In this paper, in order to deal with a relevant notion of morphisms between logics, we define a *logic* as a *diagrammatic logic*, in the sense of [Domínguez & Duval 2010]. For the purpose of this paper let us simply say that a logic $\mathcal{L}$ determines a category of theories $\mathcal{T}$ which is cocomplete, and that a morphism
of logics is a left adjoint functor, so that it preserves the colimits. The objects of $\mathbf{T}$ are called the a theories of the logic $\mathcal{L}$. Quite often, $\mathbf{T}$ is a category of structured categories. The inference rules of the logic $\mathcal{L}$ describe the structure of its theories. When a theory $\Phi$ is generated by some presentation or specification $\Sigma$, a model of $\Sigma$ with values in a theory $\Theta$ is a morphism $M: \Phi \to \Theta$ in $\mathbf{T}$.

The monadic equational logic. For instance, and for future use in the paper, here is the way we describe the monadic equational logic $\mathcal{L}_{\text{meqn}}$. In order to focus on the syntactic aspect of the theories, we use a congruence symbol “$\equiv$” rather than the equality symbol “$=$”. Roughly speaking, a monadic equational theory is a sort of category where the axioms hold only up to congruence (in fact, it is a 2-category). Precisely, a monadic equational theory is a directed graph (its vertices are called objects or types and its edges are called morphisms or terms) with an identity term $\text{id}_X : X \to X$ for each type $X$ and a composed term $g \circ f : X \to Z$ for each pair of consecutive terms $(f : X \to Y, g : Y \to Z)$; in addition it is endowed with equations $f \equiv g : X \to Y$ which form a congruence, which means, an equivalence relation on parallel terms compatible with the composition; this compatibility can be split in two parts: substitution and replacement. In addition, the associativity and identity axioms hold up to congruence. These properties of the monadic equational theories can be described by a set of inference rules, as in Figure 1.

![Figure 1: Rules of the monadic equational logic](image-url)

Adding products to the monadic equational logic. In contrast with equational theories, the existence of products is not required in a monadic equational theory. However some specific products may exist. A product in a monadic equational theory $\mathbf{T}$ is “up to congruence”, in the following sense. Let $(Y_i)_{i \in I}$ be a family of objects in $\mathbf{T}$, indexed by some set $I$. A product with base $(Y_i)_{i \in I}$ is a cone $(q_i : Y \to Y_i)_{i \in I}$ such that for every cone $(f_i : X \to Y_i)_{i \in I}$ on the same base there is a term $f = (f_i)_{i \in I} : X \to Y$ such that $q_i \circ f \equiv f_i$ for each $i$, and in addition this term is unique up to congruence, in the sense that if $g : X \to Y$ is such that $q_i \circ g \equiv f_i$ for each $i$ then $g \equiv f$. When $I$ is empty, we get a terminal object 1, such that for every $X$ there is an arrow $(\_)_X : X \to 1$ which is unique up to congruence. The corresponding inference rules are given in Figure 2. The quantification “$\forall i$”, or “$\forall i \in I$”, is a kind of “syntactic sugar”: when occuring in the premisses of a rule, it stands for a conjuction of premisses.
When \((q_i : Y \rightarrow Y_i)_{i \in I}\) is a product:

\[
\begin{align*}
(f_i : X \rightarrow Y_i) &\quad \text{(tuple)} \\
\langle f_i \rangle : X \rightarrow Y &\quad \text{(tuple-proj)} \\
q_i \circ \langle f_i \rangle &\equiv f_i &\quad \text{(tuple-unique)} \\
g : X \rightarrow Y &\quad \forall i \ q_i \circ g \equiv f_i &\quad \text{(tuple-unique)}
\end{align*}
\]

When \(\bot\) is a terminal type (“empty product”):

\[
\begin{align*}
X &\quad \text{(final)} \\
\langle \rangle : X \rightarrow \bot &\quad \text{(final-unique)} \\
g : X \rightarrow \bot &\equiv \langle \rangle X
\end{align*}
\]

Figure 2: Rules for products

## 2 Three logics for states

In this section we introduce three logics for dealing with states as computational effects. This generalizes the example of the bank account in Section 1.2. We present first the explicit logic (close to the semantics), then the apparent logic (close to the syntax), and finally the decorated logic and the morphisms from the decorated logic to the apparent and the explicit ones. In the syntax of an imperative language there is no type of states (the state is “hidden”) while the interpretation of this language involves a set of states \(S\). More precisely, if the types \(X\) and \(Y\) are interpreted as the sets \([X]\) and \([Y]\), then each term \(f : X \rightarrow Y\) is interpreted as a function \([[f]] : [[X]] \times St \rightarrow [[Y]] \times St\). In Moggi’s paper introducing monads for effects [Moggi 1991] such a term \(f : X \rightarrow Y\) is called a computation, and whenever the function \([[f]]\) is \([[f]]_0 \times id_{St}\) for some \([[f]]_0 : [[X]] \rightarrow [[Y]]\) then \(f\) is called a value. We keep this distinction, using modifier and pure term instead of computation and value, respectively. In addition, an accessor (or inspector) is a term \(f : X \rightarrow Y\) that is interpreted by a function \([[f]] = \langle [[f]]_1, q_X \rangle\), for some \([[f]]_1 : [[X]] \times St \rightarrow [[Y]]\), where \(q_X : [[X]] \times St \rightarrow St\) is the projection. It follows that every pure term is an accessor and every accessor is a modifier. We will respectively use the decorations (0), (1) and (2), written as superscripts, for pure terms, accessor and modifiers. Moreover, we distinguish two kinds of equations: when \(f, g : X \rightarrow Y\) are parallel terms, then a strong equation \(f \equiv g\) is interpreted as the equality \([[f]] = [[g]] : [[X]] \times St \rightarrow [[Y]] \times St\), while a weak equation \(f \sim g\) is interpreted as the equality \(p_Y \circ [[f]] = p_Y \circ [[g]] : [[X]] \times St \rightarrow [[Y]]\), where \(p_Y : [[Y]] \times St \rightarrow [[Y]]\) is the projection. Clearly, strong and weak equations coincide on accessor and on pure terms, while they differ on modifiers. As in Section 1.1, we consider some given set of locations \(Loc\) and for each location \(i\) a set \(Val_i\) of possible values for \(i\). The set of states is defined as \(St = \prod_{i \in Loc} Val_i\), and the projections are denoted by \(\text{lookup}_{i,1} : St \rightarrow Val_i\). For each location \(i\), let \(\text{update}_i : Val_i \times St \rightarrow St\) be defined by Equations (1) as in Section 1.1. In order to focus on the fundamental properties of states as effects, the three logics for states are based on the “poor” monadic equational logic (as described in Section 1.3).

### 2.1 The explicit logic for states

The explicit logic for states \(\mathcal{L}_{\text{expl}}\) is a kind of “pointed” monadic equational logic: a theory \(\Theta_{\text{expl}}\) for \(\mathcal{L}_{\text{expl}}\) is a monadic equational theory with a distinguished object \(S\), called the type of states, and with a product-with-\(S\) functor \(X \times S\). As in Section 1.2 the explicit logic provides the relevant semantics, but it is far from the syntax. The explicit theory for states \(State_{\text{expl}}\) is generated by a type \(V_i\) and an operation \(l_{i,1} : S \rightarrow V_i\) for each location \(i\), which form a product \((l_{i,1} : S \rightarrow V_i)_{i \in Loc}\). Thus, for each location \(i\) there
is an operation \( u_i : V_i \times S \rightarrow S \), unique up to congruence, which satisfies the equations below (where \( p_i : V_i \times S \rightarrow V_i \) and \( q_i : V_i \times S \rightarrow S \) are the projections):

\[
\begin{aligned}
\text{State}_{\text{expl}} : & \quad \text{operations} & l_{i,1} : S \rightarrow V_i , & u_i : V_i \times S \rightarrow S \\
& \quad \text{product} & (l_{i,1} : S \rightarrow V_i)_{i \in \text{Loc}} \\
& \quad \text{equations} & l_{i,1} \circ u_i \equiv p_i : V_i \times S \rightarrow V_i , & l_{j,1} \circ u_i \equiv l_{j,1} \circ q_i : V_i \times S \rightarrow V_j \quad \text{for each } j \neq i
\end{aligned}
\]

Let us define the explicit theory \( \text{Set}_{\text{expl}} \) as the category of sets with the equality as congruence and with the set of states \( St = \prod_{i \in \text{Loc}} \text{Val}_i \) as its distinguished set. The semantics of states, as described in Section 1.1, is the model \( M_{\text{expl}} : \text{State}_{\text{expl}} \rightarrow \text{Set}_{\text{expl}} \) which maps the type \( V_i \) to the set \( \text{Val}_i \) for each \( i \in \text{Loc} \), the type \( S \) to the set \( St \), and the operations \( l_{i,1} \) and \( u_i \) to the functions \( \text{lookup} \) and \( \text{update}_i \), respectively.

### 2.2 The apparent logic for states

The apparent logic for states \( \mathcal{L}_{\text{app}} \) is the monadic equational logic (Section 1.3). As in Section 1.2, the apparent logic is close to the syntax but it does not provide the relevant semantics. The apparent theory for states \( \text{State}_{\text{app}} \) can be obtained from the explicit theory \( \text{State}_{\text{expl}} \) by identifying the type of states \( S \) with the unit type \( \bot \). So, there is in \( \text{State}_{\text{app}} \) a terminal type \( \bot \) and for each location \( i \) a type \( V_i \) for the possible values of \( i \) and an operation \( l_i : \bot \rightarrow V_i \) for observing the value of \( i \). A set-valued model for this part of \( \text{State}_{\text{app}} \), with the constraint that for each \( i \) the interpretation of \( V_i \) is the given set \( \text{Val}_i \), is made of an element \( u_i \in \text{Val}_i \) for each \( i \) (it is the image of the interpretation of \( l_i \)). Thus, such a model corresponds to a state, made of a value for each location; this is known as the states-as-models or states-as-algebras point of view [Gaudel et al. 1996]. In addition, it is assumed that in \( \text{State}_{\text{app}} \) the operations \( l_i \)’s form a product \( (l_i : \bot \rightarrow V_i)_{i \in \text{Loc}} \). This assumption implies that each \( l_i \) is an isomorphism, so that each \( V_i \) must be interpreted as a singleton: this does not fit with the semantics of states. However, we will see in Section 2.3 that this assumption becomes meaningful when decorations are added, in a similar way as in the bank example in Section 1.2. Formally, the assumption that \( (l_i : \bot \rightarrow V_i)_{i \in \text{Loc}} \) is a product provides for each location \( i \) an operation \( u_i : V_i \rightarrow \bot \), unique up to congruence, which satisfies the equations below (where \( \text{id}_i : V_i \rightarrow V_i \) is the identity and \( \langle \_ \rangle_i = \langle \_ \rangle_{V_i} : V_i \rightarrow \bot \)):

\[
\begin{aligned}
\text{State}_{\text{app}} : & \quad \text{operations} & l_i : \bot \rightarrow V_i , & u_i : V_i \rightarrow \bot \\
& \quad \text{product} & (l_i : \bot \rightarrow V_i)_{i \in \text{Loc}} \quad \text{with terminal type } \bot \\
& \quad \text{equations} & l_i \circ u_i \equiv \text{id}_i : V_i \rightarrow V_i , & l_j \circ u_i \equiv l_j \circ \langle \_ \rangle_i : V_i \rightarrow V_j \quad \text{for each } j \neq i
\end{aligned}
\]

At first view, these equations mean that after \( u_i(a) \) is executed, the value of \( i \) is put to \( a \) and the value of \( j \) (for \( j \neq i \)) is unchanged. However, as noted above, this intuition is not supported by the semantics in the apparent logic. However, the apparent logic can be used for checking the validity of a decorated proof, as explained in Section 2.4.

### 2.3 The decorated logic for states

Now, as in Section 1.2, we introduce a third logic for states, which is close to the syntax and which provides the relevant semantics. It is defined by adding “decorations” to the apparent logic. A theory \( \Theta_{\text{deco}} \) for the decorated logic for states \( \mathcal{L}_{\text{deco}} \) is made of:

- A monadic equational theory \( \Theta^{(2)} \). The terms in \( \Theta^{(2)} \) may be called the modifiers and the equations \( f \equiv g \) may be called the strong equations.
• Two additional monadic equational theories $\Theta^{(0)}$ and $\Theta^{(1)}$, with the same types as $\Theta^{(2)}$, and such that $\Theta^{(0)} \subseteq \Theta^{(1)} \subseteq \Theta^{(2)}$ and the congruence on $\Theta^{(0)}$ and on $\Theta^{(1)}$ is the restriction of the congruence on $\Theta^{(2)}$. The terms in $\Theta^{(1)}$ may be called the *accessors*, and if they are in $\Theta^{(0)}$ they may be called the *pure terms*.

• A second equivalence relation $\sim$ between parallel terms in $\Theta^{(2)}$, which is only “weakly” compatible with the composition; the relation $\sim$ satisfies the substitution property but only a weak version of the replacement property, called the *pure replacement*: if $f_1 \sim f_2 : X \to Y$ and $g : Y \to Z$ then in general $g \circ f_1 \not\sim g \circ f_2$, except when $g$ is pure. The relations $f \sim g$ are called the *weak equations*. It is assumed that every strong equation is a weak equation and that every weak equation between accessors is a strong equation, so that the relations $\equiv$ and $\sim$ coincide on $\Theta^{(0)}$ and on $\Theta^{(1)}$.

We use the following notations, called *decorations*: a pure term $f$ is denoted $f^{(0)}$, an accessor $f$ is denoted $f^{(1)}$, and a modifier $f$ is denoted $f^{(2)}$; this last decoration is unnecessary since every term is a modifier, however it may be used for emphasizing. Figure 3 provides the decorated *rules*, which describe the properties of the decorated theories. For readability, the decoration properties may be grouped with other properties: for instance, “$f^{(1)} \sim g^{(1)}$” means “$f^{(1)}$ and $(f^{(1)}$ and $f \sim g$”.

![Diagram](image.png)

**Figure 3**: Rules of the decorated logic for states

Some specific kinds of products may be used in a decorated theory, for instance:

• A distinguished type $\bot$ with the following *decorated terminality* property: for each type $X$ there is a pure term $\langle \rangle_X : X \to \bot$ such that every modifier $g : X \to \bot$ satisfies $g \sim \langle \rangle_X$. It follows from the properties of weak equations that $\bot$ is a terminal type in $\Theta^{(0)}$ and in $\Theta^{(1)}$.

• An *observational product* with base $(Y_i)_{i \in I}$ is a cone of accessors $(q_i : Y \to Y_i)_{i \in I}$ such that for every cone of accessors $(f_i : X \to Y_i)_{i \in I}$ on the same base there is a modifier $f = (f_i)_{i \in I} : X \to Y$ such that $q_i \circ f \sim f_i$ for each $i$, and in addition this modifier is unique up to strong equations, in the sense that if $g : X \to Y$ is a modifier such that $q_i \circ g \sim f_i$ for each $i$ then $g \equiv f$. An observational product allows to prove strong equations from weak ones: by looking at the results of some observations, thanks to the properties of the observational product, we get information on the state.
When \( \mathbb{1} \) is a decorated terminal type:

\[
\begin{align*}
\text{(0-final)} & \quad \frac{X}{\langle X \rangle_\mathbb{1}: X \to \mathbb{1}} \\
\text{(~-final-unique)} & \quad \frac{g: X \to \mathbb{1}}{g \sim \langle X \rangle}
\end{align*}
\]

When \((q_i^{(1)}: Y \to Y_i)_i\) is an observational product:

\[
\begin{align*}
\text{(obs-tuple)} & \quad \frac{(f_i^{(1)}: X \to Y_i)_i}{\langle f_i \rangle_i^{(2)}: X \to Y} \\
\text{(obs-tuple-proj-i)} & \quad \frac{q_i \circ \langle f_i \rangle_i \sim f_i}{\langle f_i \rangle_i^{(2)}: X \to Y} \quad \text{and} \quad \frac{g: X \to Y \forall i q_i \circ g \sim f_i^{(1)}_i}{g \equiv \langle f_i \rangle_i}
\end{align*}
\]

Figure 4: Rules for some decorated products for states

The decorated theory of states \( \text{State}_{\text{deco}} \) is generated by a type \( V_i \) and an accessor \( l_i^{(1)}: \mathbb{1} \to V_i \) for each \( i \in \text{Loc} \), which form an observational product \((l_i^{(1)}: \mathbb{1} \to V_i)_i \in \text{Loc}\). The modifiers \( u_i \)'s are defined (up to strong equations), using the property of the observational product, by the weak equations below:

\[
\begin{align*}
\text{State}_{\text{deco}}: & \quad \begin{cases} 
\text{operations} & \quad l_i^{(1)}: \mathbb{1} \to V_i, u_i^{(2)}: V_i \to \mathbb{1} \\
\text{observational product} & \quad (l_i^{(1)}: \mathbb{1} \to V_i)_{i \in \text{Loc}} \text{ with decorated terminal type } \mathbb{1} \\
\text{equations} & \quad l_i \circ u_i \sim id_i: V_i \to V_i, l_j \circ u_i \sim l_j \circ \langle \rangle_i: V_i \to V_j \text{ for each } j \neq i
\end{cases}
\end{align*}
\]

The decorated theory of sets \( \text{Set}_{\text{deco}} \) is built from the category of sets, as follows. There is in \( \text{Set}_{\text{deco}} \) a type for each set, a modifier \( f^{(2)}: X \to Y \) for each function \( f: X \times \text{St} \to Y \times \text{St} \), an accessor \( f^{(1)}: X \to Y \) for each function \( f: X \times \text{St} \to Y \), and a pure term \( f^{(0)}: X \to Y \) for each function \( f: X \to Y \), with the straightforward conversions. Let \( f^{(2)}: g^{(2)}: X \to Y \) corresponding to \( f, g: X \times \text{St} \to Y \times \text{St} \). A strong equation \( f \equiv g \) is an equality \( f = g: X \times \text{St} \to Y \times \text{St} \), while a weak equation \( f \sim g \) is an equality \( p \circ f = p \circ g: X \times \text{St} \to Y \), where \( p: Y \times \text{St} \to Y \) is the projection. For each location \( i \) the projection \( \text{lookup}_i: \text{St} \to \text{Val}_i \) corresponds to an accessor \( \text{lookup}_i^{(1)}: \mathbb{1} \to \text{Val}_i \) in \( \text{Set}_{\text{deco}} \), so that the family \((\text{lookup}_i^{(1)}): i \in \text{Loc}\) forms an observational product in \( \text{Set}_{\text{deco}} \). We get a model \( M_{\text{deco}} \) of \( \text{State}_{\text{deco}} \) with values in \( \text{Set}_{\text{deco}} \) by mapping the type \( V_i \) to the set \( \text{Val}_i \) and the accessor \( l_i^{(1)} \) to the accessor \( \text{lookup}_i^{(1)} \), for each \( i \in \text{Loc} \). Then for each \( i \) the modifier \( u_i^{(2)} \) is mapped to the modifier \( \text{update}_i^{(2)} \).

### 2.4 From decorated to apparent

Every decorated theory \( \Theta_{\text{deco}} \) gives rise to an apparent theory \( \Theta_{\text{app}} \) by dropping the decorations, which means that the apparent theory \( \Theta_{\text{app}} \) is made of a type \( X \) for each type \( X \) in \( \Theta_{\text{deco}} \), a term \( f: X \to Y \) for each modifier \( f: X \to Y \) in \( \Theta_{\text{deco}} \) (which includes the modifiers and the pure terms), and an equation \( f \equiv g \) for each weak equation \( f \sim g \) in \( \Theta_{\text{deco}} \) (which includes the strong equations). Thus, the distinction between modifiers, accessors and pure terms disappears, as well as the distinction between weak and strong equations. Equivalently, the apparent theory \( \Theta_{\text{app}} \) can be defined as the apparent theory \( \Theta^{(2)} \) together with an equation \( f \equiv g \) for each weak equation \( f \sim g \) in \( \Theta_{\text{deco}} \) which is not associated to a strong equation in \( \Theta_{\text{deco}} \) (otherwise, it is yet in \( \Theta^{(2)} \)). Thus, a decorated terminal type in \( \Theta_{\text{deco}} \) becomes a terminal type in \( \Theta_{\text{app}} \) and an observational product \((q_i^{(1)}: Y \to Y_i)_i\) in \( \Theta_{\text{deco}} \) becomes a product \((q_i: Y \to \)
Every decorated theory \( \Theta_{\text{deco}} \) gives rise to an explicit theory \( \Theta_{\text{expl}} \) by expanding the decorations, which means that the explicit theory \( \Theta_{\text{expl}} \) is made of:

- A type \( X \) for each type \( X \) in \( \Theta_{\text{deco}} \); projections are denoted by \( p_X : X \times S \to X \) and \( q_X : X \times S \to S \).
- A term \( f : X \times S \to Y \times S \) for each modifier \( f : X \to Y \) in \( \Theta_{\text{deco}} \), such that:
  - if \( f \) is an accessor then there is a term \( f_1 : X \times S \to Y \) in \( \Theta_{\text{expl}} \) such that \( f = \langle f_1, q_X \rangle \),
  - if moreover \( f \) is a pure term then there is a term \( f_0 : X \to Y \) in \( \Theta_{\text{expl}} \) such that \( f_1 = f_0 \circ p_X : X \times S \to Y \), hence \( f = \langle f_0 \circ p_X, q_X \rangle = f_0 \times \text{id}_S \) in \( \Theta_{\text{expl}} \).
- An equation \( f \equiv g : X \times S \to Y \times S \) for each strong equation \( f \equiv g : X \to Y \) in \( \Theta_{\text{deco}} \).
- An equation \( p_Y \circ f \equiv p_Y \circ g : X \times S \to Y \) for each weak equation \( f \sim g : X \to Y \) in \( \Theta_{\text{deco}} \).
- A product \( (q_i, 1 : Y \times S \to Y_i) \) for each observational product \( (q_i^1 : Y \to Y_i)_i \) in \( \Theta_{\text{deco}} \).

This construction of \( \Theta_{\text{expl}} \) from \( \Theta_{\text{deco}} \) is a morphism from \( \mathcal{L}_{\text{deco}} \) to \( \mathcal{L}_{\text{expl}} \), denoted \( F_{\text{expl}} \) and called the expansion. The expansion morphism makes explicit the meaning of the decorations, by introducing a "type of states" \( S \). Thus, each modifier \( f^{(2)} \) gives rise to a term \( f \) which may use and modify the state, while whenever \( f^{(1)} \) is an accessor then \( f \) may use the state but is not allowed to modify it, and when moreover \( f^{(0)} \) is a pure term then \( f \) may neither use nor modify the state. When \( f^{(2)} \equiv g^{(2)} \) then \( f \) and \( g \) must return the same result and the same state; when \( f^{(2)} \sim g^{(2)} \) then \( f \) and \( g \) must return the same result but maybe not the same state. We have seen that the semantics of states cannot be described in the apparent logic, but can be described both in the decorated logic and in the explicit logic. It should be reminded that every morphism of logics is a left adjoint functor. This is the case for the expansion morphism \( F_{\text{expl}} : \mathcal{L}_{\text{deco}} \to \mathcal{L}_{\text{expl}} \); it is a left adjoint functor \( F_{\text{expl}} : T_{\text{deco}} \to T_{\text{expl}} \), its right adjoint is denoted \( G_{\text{expl}} \). In fact, it is easy to check that \( \text{Set}_{\text{deco}} = G_{\text{expl}}(\text{Set}_{\text{expl}}) \), and since \( \text{State}_{\text{expl}} = F_{\text{expl}}(\text{State}_{\text{deco}}) \) it follows that the decorated model \( M_{\text{deco}} : \text{State}_{\text{deco}} \to \text{Set}_{\text{deco}} \) and the explicit model \( M_{\text{expl}} : \text{State}_{\text{expl}} \to \text{Set}_{\text{expl}} \) are related by the adjunction \( F_{\text{expl}} \dashv G_{\text{expl}} \). This means that the models \( M_{\text{deco}} \) and \( M_{\text{expl}} \) are two different ways to formalize the semantics of states from Section 1.1. In order to conclude Section 2, the morphisms of logic \( F_{\text{app}} \) and \( F_{\text{expl}} \) are summarized in Figure 5.

3 Decorated proofs

The inference rules of the decorated logic \( \mathcal{L}_{\text{deco}} \) are now used for proving some of the Equations (2) (in Remark 1.1). All proofs in this section are performed in the decorated logic; for readability the identity and associativity rules \( (\text{id-src}) \), \( (\text{id-tgt}) \) and \( (\text{assoc}) \) are omitted. Some derived rules are proved in Section 3.1; then Equation (2.1) is proved in Section 3.2. In order to deal with the equations with two values as argument or as result, we use the semi-pure products introduced in [Dumas et al., 2011]; the rules for semi-pure products are reminded in Section 3.3; then all seven Equations (2) are expressed in the decorated logic and Equation (2.6) is proved in Section 3.4. Proving the other equations would be similar. We use as axioms the fact that \( l_i \) is an accessor and the weak equations in \( \text{State}_{\text{deco}} \) (Section 2.3).
3.1 Some derived rules

Let us now derive some rules from the rules of the decorated logic (Figures 3 and 4).

\[
\begin{array}{c|c|c|c|c|c}
\text{Figure 5: A span of logics for states} \\
\end{array}
\]

\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\Theta_{\text{app}} & \Theta_{\text{deco}} & \Theta_{\text{expl}} \\
\hline
f : X \to Y & \text{modifier} & \text{accessor} & \text{pure term} & f : X \to Y & f^{(1)} : X \to Y & f^{(0)} : X \to Y & f : X \times S \to Y \times S \\
f : X \to Y & \text{strong equation} & \text{weak equation} & \text{weak equation} & f : X \to Y & f : X \to Y & f : X \times S \to Y \times S & f : X \times S \to Y \times S \\
f \equiv g : X \to Y & f \equiv g : X \to Y & f \equiv g : X \times S \to Y \times S & p_Y \circ f \equiv p_Y \circ g : X \times S \to Y \times S \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|c|c|c|c|c|}
\hline
\text{Figure 6: Some derived rules in the decorated logic for states} \\
\end{array}
\]

**Proof.** The derived rules in the left part of Figure 6 can be proved as follows. The proof of the rules in the right part are left to the reader.

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c|c}
\hline
(E_1^{(1)}) & f^{(1)} : X \to 1 & f \equiv \langle \rangle_X \\
(E_2^{(1)}) & f^{(1)} : X \to 1 & g^{(1)} : X \to 1 & f \equiv g \\
(E_3^{(1)}) & f^{(1)} : X \to Y & g^{(1)} : Y \to 1 & h^{(1)} : X \to 1 & g \circ f \equiv h \\
(E_4^{(1)}) & f^{(1)} : 1 \to X & \langle \rangle_X \circ f \equiv id_1 \\
\hline
(E_1^{(0)}) & f^{(0)} : X \to 1 & f \equiv \langle \rangle_X \\
(E_2^{(0)}) & f^{(0)} : X \to 1 & g^{(0)} : X \to 1 & f \equiv g \\
(E_3^{(0)}) & f^{(0)} : X \to Y & g^{(0)} : Y \to 1 & h^{(0)} : X \to 1 & g \circ f \equiv h \\
(E_4^{(0)}) & f^{(0)} : 1 \to X & \langle \rangle_X \circ f \equiv id_1 \\
\hline
\end{array}
\]

\[
\begin{array}{c}
(0\text{-final}) & X \\
(0\text{-to-1}) & \langle \rangle_X^{(0)} \quad \langle \rangle_X^{(1)} \\
(\sim\text{-final-unique}) & f : X \to 1 \\
\hline
(1\sim\text{-to-1}) & f^{(1)} \\
(\equiv\text{-sym}) & \langle \rangle_1^{(1)} \equiv g \\
\hline
(E_1^{(1)}) & f^{(1)} : 1 \to X \\
(\equiv\text{-trans}) & f \equiv \langle \rangle_1^{(1)} \\
\hline
(E_2^{(1)}) & \langle \rangle_1^{(1)} \equiv g \\
(E_3^{(1)}) & f^{(1)} : 1 \to X \\
(0\text{-final}) & X \\
(0\text{-to-1}) & \langle \rangle_X^{(0)} : X \to 1 \\
(0\text{-to-1}) & \langle \rangle_X^{(1)} : X \to 1 \\
(0\text{-id}) & \langle \rangle_1^{(0)} : 1 \to 1 \\
(0\text{-id}) & \langle \rangle_1^{(1)} : 1 \to 1 \\
\hline
(E_4^{(1)}) & \langle \rangle_X \circ f \equiv id_1 \\
\end{array}
\]
3.2 Annihilation lookup-update

It is easy to check that the decorated equation $u_i^{(2)} \circ l_i^{(1)} \equiv id_1^{(0)}$ gets expanded as $u_i \circ l_i \equiv id_1$, which clearly gets interpreted as Equation (2.1) in Remark 1.1. Let us prove this decorated equation, using the axioms (for each location $i$), from $\text{State}_{\text{deco}}$ in Section 2.3.

\[(A_0) \ l_i^{(1)}, \quad (A_1) \ l_i \circ u_i \sim id_i, \quad (A_2) \ l_j \circ u_i \sim \langle \rangle_i \text{ for each } j \neq i.
\]

**Proposition 3.1.** For each location $i$, reading the value of a location $i$ and then updating the location $i$ with the obtained value is just like doing nothing.

$u_i^{(2)} \circ l_i^{(1)} \equiv id_1^{(0)} : \mathbb{1} \to \mathbb{1}$.

**Proof.** Let $i$ be a location. Using the unicity property of the observational product, we have to prove that $l_k \circ u_i \circ l_i \sim l_k : \mathbb{1} \to V_k$ for each location $k$.

- When $k = i$, the substitution rule for $\sim$ yields:

  \[\frac{ (\sim\text{-subs}) \quad (A_1) \ l_i \circ u_i \sim id_i }{ l_i \circ u_i \circ l_i \sim l_i }\]

- When $k \neq i$, using the substitution rule for $\sim$ and the replacement rule for $\equiv$ we get:

  \[\frac{ (\sim\text{-trans}) \quad (\sim\text{-subs}) \quad (A_2) \ l_k \circ u_i \sim l_k \circ \langle \rangle_i }{ l_k \circ u_i \circ l_i \sim l_k \circ \langle \rangle_i \circ l_i }\]

  \[\frac{ (\equiv\text{-repl}) \quad (A_0) \ l_i^{(1)} }{ l_k \circ \langle \rangle_i \circ l_i \equiv id_1 }\]

  \[\frac{ (\equiv\text{-to-}\sim) }{ l_k \circ l_i \equiv id_1 }\]

  \[\frac{ (\equiv\text{-repl}) \quad (A_0) \ l_i^{(1)} }{ l_k \circ l_i \equiv id_1 }\]

  \[\frac{ (\equiv\text{-to-}\sim) }{ l_k \circ l_i \equiv id_1 }\]

**Remark 3.2.** At the top of the right branch in the proof above, the decoration (1) for $l_i$ could not be replaced by (2). Indeed, from $l_i^{(2)}$ we can derive the weak equation $\langle \rangle_i \circ l_i \sim id_1$, but this is not sufficient for deriving $l_k \circ \langle \rangle_i \circ l_i \sim l_k$ by replacement since $l_k$ is not pure.

3.3 Semi-pure products

Let $\Theta_{\text{deco}}$ be a theory with respect to the decorated logic for states and let $\Theta^{(0)}$ be its pure part, so that $\Theta^{(0)}$ is a monadic equational theory. The **product** of two types $X_1$ and $X_2$ in $\Theta_{\text{deco}}$ is defined as their product in $\Theta^{(0)}$ (it is a product up to strong equations, as in Section 1.1. The projections from $X_1 \times X_2$ to $X_1$ and $X_2$ are respectively denoted by $\pi_1^{(0)}$ and $\pi_2^{(0)}$ when the types $X_1$ and $X_2$ are clear from the context. The **product** of two pure morphisms $f_1^{(0)} : X_1 \to Y_1$ and $f_2^{(0)} : X_2 \to Y_2$ is a pure morphism $f_1 \times f_2^{(0)} : X_1 \times X_2 \to Y_1 \times Y_2$ subject to the rules in Figure 7 which are the usual rules for products up to strong equations. Moreover when $X_1$ or $X_2$ is $\mathbb{1}$ it can be proved in the usual way that the projections $\pi_1^{(0)} : X_1 \times \mathbb{1} \to X_1$ and $\pi_2^{(0)} : \mathbb{1} \times X_2 \to X_2$ are isomorphisms. The permutation $\text{perm}_{X_1,X_2}^{(0)} : X_1 \times X_2 \to X_2 \times X_1$ is defined as usual by $\pi_1 \circ \text{perm}_{X_1,X_2} \equiv \pi_2$ and $\pi_2 \circ \text{perm}_{X_1,X_2} \equiv \pi_1$.

The rules in Figure 7 which are symmetric in $f_1$ and $f_2$, cannot be applied to modifiers: indeed, the effect of building a pair of modifiers depends on the evaluation strategy. However, following
\[ \begin{align*}
(0\text{-prod}) & \quad f_1^{(0)} : X_1 \to Y_1 \quad f_2^{(0)} : X_2 \to Y_2 \\
(f_1 \times f_2)^{(0)} : X_1 \times X_2 \to Y_1 \times Y_2 \\
(0\text{-proj-1}) & \quad f_1^{(0)} : X_1 \to Y_1 \\
\quad f_2^{(0)} : X_2 \to Y_2 \\
\pi_1 \circ (f_1 \times f_2) & \equiv f_1 \circ \pi_1 \\
(0\text{-proj-2}) & \quad f_1^{(0)} : X_1 \to Y_1 \\
\quad f_2^{(0)} : X_2 \to Y_2 \\
\pi_2 \circ (f_1 \times f_2) & \equiv f_2 \circ \pi_2 \\
(0\text{-prod-unique}) & \quad g^{(0)} : X_1 \times X_2 \to Y_1 \times Y_2 \\
\quad \pi_1 \circ g \equiv f_1 \circ \pi_1 \\
\quad \pi_2 \circ g \equiv f_2 \circ \pi_2 \\
\end{align*} \]

Figure 7: Rules for products of pure morphisms

[Dumas et al. 2011], we define the left semi-pure product of an identity \( id_X \) and a modifier \( f : X_2 \to Y_2 \), as a modifier \( id_X \times f : X \times X_2 \to X \times Y_2 \) subject to the rules in Figure 8, which form a decorated version of the rules for products. Symmetrically, the right semi-pure product of a modifier \( f : X_1 \to Y_1 \) and an identity \( id_X \) is a modifier \( f \times id_X : X_1 \times X \to Y_1 \times X \) subject to the rules symmetric to those in Figure 8.

\[ \begin{align*}
(\text{left-prod}) & \quad f^{(2)} : X_2 \to Y_2 \\
(id_X \times f)^{(2)} : X \times X_2 \to X \times Y_2 \\
(\text{left-proj-1}) & \quad f^{(2)} : X_2 \to Y_2 \\
\quad \pi_1 \circ (id_X \times f) \sim \pi_1 \\
(\text{left-proj-2}) & \quad f^{(2)} : X_2 \to Y_2 \\
\quad \pi_2 \circ (id_X \times f) \equiv f \circ \pi_2 \\
(\text{left-prod-unique}) & \quad g^{(2)} : X \times X_2 \to Y \times Y_2 \\
\quad \pi_1 \circ g \sim \pi_1 \\
\quad \pi_2 \circ g \equiv f \circ \pi_2 \\
\end{align*} \]

Figure 8: Rules for left semi-pure products

Let us add the rules for semi-pure products to the decorated logic for states. In the decorated theory of states \( \text{State}_{\text{deco}} \), let us assume that there are products \( V_i \times V_j \) and \( \mathbb{1} \times V_j \) for all locations \( i \) and \( j \). Then it is easy to check that the expansion of the decorated Equations (2)\(_d\) below gets interpreted as Equations (2) in Remark 1.1. We use the simplified notations \( id_i = id_{V_i} \) and \( \langle \rangle_i = \langle \rangle_{V_i} \) and \( \text{perm}_{i,j} = \text{perm}_{V_i,V_j} \). Equation (2.1)\(_d\) has been proved in Section 3.2 and Equation (2.6)\(_d\) will be proved in Section 3.4. The other equations can be proved in a similar way.

(2.1)\(_d\) Annihilation lookup-update. \( \forall i \in \text{Loc} \), \( u_i \circ l_i \equiv id_i : 1 \to 1 \)

(2.2)\(_d\) Interaction lookup-update. \( \forall i \in \text{Loc} \), \( l_i \circ \langle \rangle_i \circ l_i \equiv l_i : 1 \to V_i \)

(2.3)\(_d\) Interaction update-update. \( \forall i \in \text{Loc} \), \( u_i \circ \pi_2 \circ (u_i \times id_i) \equiv u_i \circ \pi_2 : V_i \times V_i \to \mathbb{1} \)

(2.4)\(_d\) Interaction update-update. \( \forall i \in \text{Loc} \), \( l_i \circ u_i \sim id_i : V_i \to V_i \)

(2.5)\(_d\) Commutation lookup-update. \( \forall i \neq j \in \text{Loc} \), \( l_j \circ \langle \rangle_j \circ l_j \equiv \text{perm}_{j,i} \circ l_i \circ \langle \rangle_j \circ l_j : 1 \to V_i \times V_j \)

(2.6)\(_d\) Commutation update-update. \( \forall i \neq j \in \text{Loc} \), \( u_i \circ \pi_2 \circ (u_i \times id_j) \equiv u_i \circ \pi_2 \circ (id_i \times u_j) : V_i \times V_j \to \mathbb{1} \)

(2.7)\(_d\) Commutation update-update. \( \forall i \neq j \in \text{Loc} \), \( l_j \circ u_i \equiv \pi_2 \circ (id_i \times l_j) \circ (u_i \times id_j) \circ \pi_1^{-1} : V_i \to V_j \)
3.4 Commutation update-update

Proposition 3.3. For each locations $i \neq j$, the order of storing in the locations $i$ and $j$ does not matter:

$$u_j^{(2)} \circ \pi_2^{(0)} \circ (u_i \times id_j)^{(2)} \equiv u_i^{(2)} \circ \pi_1^{(0)} \circ (id_i \times u_j)^{(2)} : V_i \times V_j \rightarrow V .$$

Proof. In order to avoid ambiguity, in this proof the projections from $V_i \times 1$ are denoted $\pi_{1,i}$ and $\pi_{2,i}$ and the projections from $1 \times V_j$ are denoted $\pi_{1,j}$ and $\pi_{2,j}$, while the projections from $V_i \times V_j$ are denoted $\pi_{1,i,j}$ and $\pi_{2,i,j}$. It follows from Section 3.3 that $\pi_{1,i}$ and $\pi_{2,i}$ are isomorphisms, while the derived rule $(E_1^{(0)})$ implies that $\pi_{2,i} \equiv \langle \rangle_i$ and $\pi_{1,i} \equiv \langle \rangle_i$. Using the unicity property of the observational product, we have to prove that $l_k \circ u_j \circ \pi_{2,j} \circ (u_i \times id_j) \sim l_k \circ u_i \circ \pi_{1,i} \circ (id_i \times u_j)$ for each location $k$.

- When $k \neq i, j$, let us prove independently four weak equations (W1) to (W4):

\[
\begin{align*}
\text{(\sim -subs)} \quad & (A_2) \quad l_k \circ u_j \sim l_k \circ \langle \rangle_j \quad (W1) \\
\text{\quad (right-prod)} \quad & l_k \circ u_j \circ \pi_{2,j} \circ (u_i \times id_j) \sim l_k \circ \langle \rangle_j \circ \pi_{2,j} \circ (u_i \times id_j) \\
\text{\quad (\equiv -subs)} \quad & \langle \rangle_j \circ \pi_{2,j} \equiv \pi_{1,j} \quad (W2) \\
\text{\quad (\equiv -trans)} \quad & \langle \rangle_j \circ \pi_{2,j} \circ (u_i \times id_j) \equiv \pi_{1,j} \circ (u_i \times id_j) \equiv u_i \circ \pi_{1,i,j} \\
\text{\quad (\equiv -repl)} \quad & l_k \circ \langle \rangle_j \circ \pi_{2,j} \circ (u_i \times id_j) \equiv l_k \circ u_i \circ \pi_{1,i,j} \\
\text{\quad (\equiv -to -~)} \quad & l_k \circ \langle \rangle_j \circ \pi_{2,j} \circ (u_i \times id_j) \sim l_k \circ u_i \circ \pi_{1,i,j} \quad (W2)
\end{align*}
\]

Equations (W1) to (W4) together with the transitivity rule for $\sim$ give rise to the weak equation $l_k \circ u_j \circ \pi_{2,j} \circ (u_i \times id_j) \sim l_k \circ \langle \rangle_{V_i \times V_j}$. A symmetric proof shows that $l_k \circ u_i \circ \pi_{1,i} \circ (id_i \times u_j) \sim l_k \circ \langle \rangle_{V_i \times V_j}$. With the symmetry and transitivity rules for $\sim$, this concludes the proof when $k \neq i, j$.

- When $k = i$, it is easy to prove that $l_i \circ u_i \circ \pi_{1,i} \circ (id_i \times u_j) \sim \pi_{1,i,j}$, as follows.

\[
\begin{align*}
\text{(\sim -subs)} \quad & (A_1) \quad l_i \circ u_i \sim id_i \\
\text{\quad (left-proj-1)} \quad & l_i \circ u_i \circ \pi_{1,i} \circ (id_i \times u_j) \sim \pi_{1,i} \circ (id_i \times u_j) \\
\text{\quad (\equiv -trans)} \quad & \pi_{1,i} \circ (id_i \times u_j) \sim \pi_{1,i,j}
\end{align*}
\]

Now let us prove that $l_i \circ u_j \circ \pi_{2,j} \circ (u_i \times id_j) \sim \pi_{1,i,j}$, as follows.

\[
\begin{align*}
\text{(\sim -subs)} \quad & (A_2) \quad l_i \circ u_j \sim l_i \circ \langle \rangle_j \\
\text{\quad (\equiv -repl)} \quad & l_i \circ u_j \circ \pi_{2,j} \equiv l_i \circ \langle \rangle_{2 \times V_j} \\
\text{\quad (\equiv -to -~)} \quad & l_i \circ \langle \rangle_j \circ \pi_{2,j} \sim l_i \circ \langle \rangle_{2 \times V_j} \\
\text{\quad (\sim -subs)} \quad & l_i \circ u_j \circ \pi_{2,j} \circ (u_i \times id_j) \sim l_i \circ \langle \rangle_{1 \times V_j} \circ (u_i \times id_j) \quad (W'_1)
\end{align*}
\]
\[
\begin{align*}
\left(\equiv\text{-subs}\right) & \quad \langle E_2^{(0)} \rangle_{1 \times V_j} \equiv \pi_{1,j} \\
\left(\equiv\text{-trans}\right) & \quad \langle E_2^{(0)} \rangle_{1 \times V_j} \circ (u_i \times id_j) \equiv \pi_{1,j} \circ (u_i \times id_j) \\
& \quad \left(\equiv\text{-repl}\right) \quad l_i \circ \langle E_2^{(0)} \rangle_{1 \times V_j} \circ (u_i \times id_j) \equiv l_i \circ u_i \circ \pi_{1,i,j} \\
& \quad \left(\equiv\text{-to-}\sim\right) \quad l_i \circ u_i \circ \pi_{1,i,j} \sim \pi_{1,i,j} (W_2') \\
& \quad \left(\sim\text{-subs}\right) \quad l_i \circ u_i \circ \pi_{1,i,j} \sim \pi_{1,i,j} (W_3') \\
& \quad (\sim\text{-proj-1}) \quad \pi_{1,j} \circ (u_i \times id_j) \equiv u_i \circ \pi_{1,i,j}
\end{align*}
\]

Equations \((W_1')\) to \((W_3')\) and the transitivity rule for \(\sim\) give rise to \(l_i \circ u_j \circ \pi_{2,j} \circ (u_i \times id_j) \sim \pi_{1,j,j}\).

With the symmetry and transitivity rules for \(\sim\), this concludes the proof when \(k = i\).

\[\blacksquare\]

**Conclusion**

In this paper, decorated proofs are used for proving properties of states. To our knowledge, such proofs are new. They can be expanded in order to get the usual proofs, however decorated proofs are more concise and closer to the syntax; in the expanded proof the notion of effect is lost. This approach can be applied to other computational effects, like exceptions [Dumas et al. 2012a] [Dumas et al. 2012b].

**References**


Characterizing Van Kampen Squares via Descent Data

Harald Köring
University of Applied Sciences
FHDW Hannover, Germany
harald.koenig@fhdw.de

Uwe Wolter
Department of Informatics
University of Bergen, Norway
wolter@ii.uib.no

Michael Löwe
University of Applied Sciences
FHDW Hannover, Germany
michael.loewe@fhdw.de

Categories in which cocones satisfy certain exactness conditions w.r.t. pullbacks are subject to current research activities in theoretical computer science. Usually, exactness is expressed in terms of properties of the pullback functor associated with the cocone. Even in the case of non-exactness, researchers in model semantics and rewriting theory inquire an elementary characterization of the image of this functor. In this paper we will investigate this question in the special case where the cocone is a cospan, i.e. part of a Van Kampen square. The use of Descent Data as the dominant categorical tool yields two main results: A simple condition which characterizes the reachable part of the above mentioned functor in terms of liftings of involved equivalence relations and (as a consequence) a necessary and sufficient condition for a pushout to be a Van Kampen square formulated in a purely algebraic manner.

1 Introduction

There is a considerable amount of theoretical work in software engineering and category theory that has frequently encountered the question whether the interplay of pushouts and pullbacks satisfies certain exactness conditions. There is ongoing research in classifying and characterizing categories in which colimits and pullbacks are reasonably related. A prominent example are adhesive categories \[13\], in which pushouts along monomorphisms are Van Kampen squares. However, this property can be formulated for any commutative square in the bottom of Figure 1 as follows: The functor \(PB\), which takes \(\sigma \in C \downarrow S\) and maps it to a rear pullback span by pulling back along \(\overline{a} \circ r = r \circ a\), has to be an equivalence of categories.

\[
\begin{array}{cccc}
I & \rightarrow & H \\
\downarrow & & \downarrow \\
J & \rightarrow & K \\
\downarrow \alpha & & \downarrow \sigma \\
A & \rightarrow & S \\
\downarrow \tau & & \downarrow \pi
\end{array}
\]

Figure 1: Van Kampen square

If the bottom square is a pushout, the following property is equivalent to this definition: In every commutative cube as in Figure 1 with two pullbacks as rear faces, the following equivalence holds: The top face is a pushout if and only if the front and right faces are pullbacks \[18\].

The category \(SET\) of sets and mappings between them as well as the category \(GRAPH\) of directed graphs \[1\] and graph morphisms are adhesive. In many applications (e.g. \[4,18\]) it is sufficient to infer exactness

\[1\] i.e. directed graphs \((V,E;s,t:E \rightarrow V)\]
for pushouts along monomorphisms only. However, it is well-known that already in \textit{SET} there are many more Van Kampen squares than the ones where one participating morphism is monic. Additionally, several research topics have evolved, where the implications of the Van Kampen property were needed in the case of non-monic \( a \) and \( r \).

An important example are \textit{diagrammatic specifications}\footnote{E.g. UML class diagrams or ER diagrams} in model driven engineering\footnote{We use this abbreviation, because the term "interpretation" is often substituted by the term "algebra."}: In Figure 2 there are specifications \( A, L, R, \) and \( S \) each of which contain (data) types and directed relations between them, i.e. they are small graphs. Since the specifications require \textit{compositionality} [3], it is important to investigate \textit{amalgamation}, a simple and natural construction which provides the basis for compositionality. It is a method to uniquely and correctly compose interpretations of parts of an already composed specification. Formulated in indexed semantics this takes the form as shown in the left diagram of Figure 2. There is also a global view on the amalgamation procedure in the indexed setting: Let us denote the category of interpretations of a specification \( X \) by \( \text{Alg}(X) \) and let \( V \) denote the usual forgetful functor along a specification morphism (e.g. the functor \( V_r : \text{Alg}(R) \to \text{Alg}(L) \) is defined by \( V_r(\beta) = \beta \circ r \)), cf. Figure 3. The Amalgamation Lemma [5] states that (2) is a pullback in the category \textit{CAT} of categories if (1) is a pushout of specifications.

![Figure 2: Indexed vs Fibred Amalgamation of \( \tau \) and \( \beta \) with common part \( \gamma \)](image2)

There is also a global view on the amalgamation procedure in the indexed setting: Let us denote the category of interpretations of a specification \( X \) by \( \text{Alg}(X) \) and let \( V \) denote the usual forgetful functor along a specification morphism (e.g. the functor \( V_r : \text{Alg}(R) \to \text{Alg}(L) \) is defined by \( V_r(\beta) = \beta \circ r \)), cf. Figure 3. The Amalgamation Lemma [5] states that (2) is a pullback in the category \textit{CAT} of categories if (1) is a pushout of specifications.

![Figure 3: Amalgamation Lemma (Indexed setting)](image3)
nodes $t_1, t_2$ in $L$). On the other hand, one can determine all objects that are $t$-typed by considering $\gamma(t)$.

But reliable semantics for model-driven structures has to omit the "philosophy" of semantic universes, because in software environments each object possesses exactly one type and it should not be possible to determine the set of $t$-typed objects. This mismatch requires the shift from indexed to fibred semantics [3]. In the fibred setting, interpretations are called instances and are formalized by objects of the slice categories $\mathcal{C}_{\downarrow}A$, $\mathcal{C}_{\downarrow}L$, $\mathcal{C}_{\downarrow}R$, and $\mathcal{C}_{\downarrow}S$. Forgetful functors are now "pulling back"-functors (e.g. the functor $r^*$ which constructs the pullback of $(r, \beta)$, see the right part of Figure 2).

This raises the question whether the amalgamation procedure smoothly carries over to the fibred setting. I.e. given two instances $\tau \in \mathcal{C}_{\downarrow}A$ and $\beta \in \mathcal{C}_{\downarrow}R$ with common part $\gamma$, i.e. $r^*\beta = \gamma = a^*\tau$, one wants to prove that the syntactical composition (pushout of $a$ and $r$) is reflected on the instance level by a unique construction. The counterpart for correctness is the requirement to obtain an $S$-instance of $\mathcal{C}_{\downarrow}S$, such that its pullbacks along $a$ and $r$ yield $\beta$ and $\tau$, resp. cf. Figure 2.

2 The Reachability Problem

In contrast to indexed amalgamation, there are intrinsic difficulties for the fibred setting, because the given rear pullback span must not be in the image of $PB$. In other words, a reasonable construction on the instance-level fails if and only if the pullback span is not reachable by $PB$. This is demonstrated in Example 1 In Figure 4 objects are denoted $i:t$, instances map objects to their types. $a$ and $r$ map according to the letters. $i:t, j:s \in I$ are connected via dashed lines if $r'(i:t) = r'(j:s)$. Dotted lines depict the kernel of $a'$. It can easily be computed that the two rear squares establish a pullback span in $SET$.

However, the span is not reachable: On the one hand, pullback complements for the right and the front face with sets over $S$ containing two elements will always yield a non-commutative top face. On the other hand, the pushout on the top face creates a $\mathcal{C}_{\downarrow}S$-object (the mediator out of the pushout), whose domain is a singleton set. But pulling back this instance along $\tau, \bar{a}$ resp. does not yield $\tau$ and $\beta$, resp.

![Figure 4: Unreachable pullback span](image)

These effects can not occur in the indexed setting because multiple typing was allowed. To get rid of multiple typing, the transition from indexed to fibred semantics entails the production of copies. E.g. in the indexed setting, it would be sufficient to let $\gamma$ map each element of $L$ to the set $\{1, 2\}$, whereas the fibred view requires to produce 4 copies of this 2-element set (yielding the set $I$ in Figure 4). It is

---

4 Consider conformance relations in standards of software engineering (e.g. UML object diagrams or MOF) [17].
well-known that indexed categories are related to fibrations via the Grothendieck construction \cite{1,19}. However, since the image of this construction is the category of split fibrations, all produced copies behave in a uniform way as in the next example.

**Example 2** In this example fibres are lifted in a uniform way. The pullback span is now reachable. It is isomorphic to \(\text{PB}(\sigma)\) where \(\sigma : \{1:xyzw, 2:xyzw\} \to S\).

![Figure 5: Reachable pullback span](image)

But if pullback spans are not results of the Grothendieck construction, we suffer from the enlarged degree of freedom for defining the relationship between fibres, i.e. the equivalence relations of \(a'\) and \(r'\) may chaotically be intertwined as in Example 1.

Of course, fibred amalgamation is successful, if the bottom square in the cube of Figure 2 would be a Van Kampen square. In this case, one simply has to construct the pushout on top of the cube and can automatically deduce that this produces two pullbacks in front as desired. Then the question arises, how to detect whether a square is a Van Kampen square from properties of \(a\) and \(r\) only.

**Example 3** In the pushout in Figure 6 neither \(a\) nor \(r\) is monic. Hence we cannot infer the Van Kampen property from the fact that SET is an adhesive category.

![Figure 6: Van Kampen square?](image)

**Question 1** Can we find a feasible condition which characterizes reachability in terms of the rear pullback span only (even in the case that the bottom square is not a Van Kampen square)?

**Question 2** Can we find a necessary and sufficient condition for a pushout to be a Van Kampen square in terms of the span \((a,r)\) only?
It became evident that a comprehensive investigation has to be performed in a more abstract categorical environment, see also [15]. A good generalization are topoi [8], i.e. categories which have finite limits, are cartesian closed, and where the subobject functor is representable\footnote{In the sequel, we assume the reader to possess basic understanding of the notion of topos.}. \textit{SET} and \textit{GRAPH} are topoi. Topoi are adhesive [14], i.e. pushouts along monomorphisms are Van Kampen squares. Thus the above questions are relevant only for the case where both $a$ and $r$ in Figure 2 have non-trivial kernel relations.

It has turned out that \textit{Descent Theory} [9] is a good tool for quantifying the interrelation of kernel pairs on a common domain. In Section 3 we describe descent data and point out its two main facets: On the one hand it describes algebraic structures, on the other hand it codes lifted equivalence relations in pullback squares.

In Section 4 we introduce precise notions of reachability of pullback spans and of coherence of a pair of algebraic structures. Algebraic structures are coherent if they are reducts of a uniquely determined larger algebraic structure. Thus coherence is a local property in the sense that it can be checked by investigating the basic material only, whereas reachability is a global property which is hardly checkable. In the main contribution of this paper (Proposition 17) we prove reachability to be equivalent to coherence, if the specification square is a pushout. This provides an answer to Question 1.

This answer is formulated in a practical way in Theorem 19. It also yields an answer to Question 2 in Theorem 20 which is a surprising analogon to the amalgamation lemma in Figure 3. Unfortunately, Theorem 20 is still unpractical in that we still have to investigate all pullback spans in order to decide the Van Kampen property. But we can show that a practical answer to Question 2 can be achieved if matters are restricted to sets and graphs (Section 5, Theorem 24).

As related work we want to mention that [14] prove topoi to be adhesive with similar methods (i.e. they use descent theory and some similar auxiliary results). Furthermore, [10] show that being a Van Kampen square in a category $\mathcal{C}$ is equivalent to saying that its embedding into a certain span category over $\mathcal{C}$ is a pushout. In contrast to this generalization to higher level structures, we aim at an elementary characterization which can be checked within $\mathcal{C}$.

\section{Descent Theory}

In this section, we work in a general topos $\mathcal{C}$. We will use the following notations: $\text{Ob}_{\mathcal{C}}, \text{Mor}_{\mathcal{C}}$ denote objects and arrows of any category $\mathcal{C}$, resp. $\xrightarrow{\alpha}$ denotes epimorphisms. $x \in \mathcal{C}$ means $x \in \text{Ob}_{\mathcal{C}}$. The application of a functor $\mathcal{F}$ to an object or an arrow $x$ will be denoted without parenthesis: $\mathcal{F}x$. For an arrow $p$ of $\mathcal{C}$ we sometimes want pullbacks along $p$ to be uniquely determined. Thus we work with \textit{chosen} pullbacks. The notation for the pullback functor $p^*$ is

$$
\begin{array}{c}
E \times_B A \\
\downarrow \pi_1^p \\
\downarrow \pi_2^p \\
E \xrightarrow{\pi_2^p} B \\
\pi_1^p := p^* \alpha \\
\alpha
\end{array}
$$

where $(\pi_1^p, \pi_2^p)$ is the chosen pullback of $(\alpha, p)$ (emphasized by decorating projections with $\alpha$).

In an adjoint situation $\bot \dashv \eta$, $\eta$ is the unit, $\varepsilon$ the co-unit. If $p : E \to B$ is any arrow in a category with pullbacks and $p_* : \mathcal{C} \downarrow E \to \mathcal{C} \downarrow B$ is the post-composing-functor, we have $p_* \dashv p^*$. The monad arising from this adjunction is $(\mathcal{F}^p, \eta^p, \mu^p)$, i.e., $\mathcal{F}^p := p^* \circ p_* : \mathcal{C} \downarrow E \to \mathcal{C} \downarrow E$, $\eta^p := \eta$, and $\mu^p := p^* \varepsilon_{p_*}$.
We intend to describe the categories $\text{des}(p)$ of Descent Data, where $p : E \to B$ is an arrow in $\mathcal{C}$. Grothendieck invented this theory in order to reason about structures in $\mathcal{C} \downarrow B$ (which may be difficult) by reasoning about monadic algebraic structures over $\mathcal{C} \downarrow E$, thus in a sense ”descending” along $p$ [9].

We analyse the relationship between these algebraic structures and the category $\text{pb}(p)$ of all pullbacks along $p$ (to be defined precisely later on) such that it will facilitate our characterization of reachability in terms of descent data.

**Definition 4 (Descent Data)** Let $C \xrightarrow{\gamma} E \xrightarrow{p} B$ be given and $(\mathcal{T}^p, \eta^p, \mu^p)$ be the monad on $\mathcal{C} \downarrow E$ arising from the adjunction $p_* \dashv p^*$. Descent data for $\gamma$ relative to $p$ is an arrow

$$\xi : \pi_1^{p \gamma} = \mathcal{T}^p \gamma \to \gamma$$

of $\mathcal{C} \downarrow E$ with

$$\xi \circ \eta^p = \text{id}_E$$

and

$$\xi \circ \mathcal{T}^p \xi = \xi \circ \mu^p.$$  \hfill (1)

The situation is as in Figure 7.

![Figure 7: Monadic Descent Data](image)

Besides the $\mathcal{C} \downarrow B$-arrow $\pi_2^{p \gamma}$, the right-hand side shows objects and the arrow $\xi$ after applying the left-adjoint $p_*$ only ($p_*$ is the identity on arrows of $\mathcal{C} \downarrow E$). Note that $\pi_2^{p \gamma}$ establishes the co-unit of the adjunction $p_* \dashv p^*$. Thus

$$\pi_2^{p \gamma} \circ \eta^p = \text{id}$$

and

$$\mu^p = p^* \pi_2^{p \gamma}.$$ \hfill (2)

Note that, for some $\gamma$ and $p$, an arrow $\xi$ as in Definition 4 must not exist and must not be unique. For future reference, we note that the $E \times_B C$-endomorphism $\overline{\xi} := \langle \gamma \circ \pi_2^{p \gamma}, \xi \rangle$ can reconstruct $\xi$ via

$$\xi = \pi_2^{p \gamma} \circ \overline{\xi}.$$ \hfill (3)

[11] gives a detailed investigation on that topic. It is also shown that

$$\overline{\xi} \circ \overline{\xi} = \text{id}_{E \times_B C}.$$ \hfill (4)

**Definition 5 (Category of Descent Data)** The category $\text{des}(p)$ has objects $(\gamma, \xi)$ with the properties of Definition 4 and arrows $h : (\gamma, \xi) \to (\gamma', \xi')$ the morphisms $h : \gamma \to \gamma'$ of $\mathcal{C} \downarrow E$ with $\xi' \circ \mathcal{T}^p h = h \circ \xi$.

**Definition 6 (Category of Pullbacks)** For any $E \xrightarrow{p} B \in \text{Mor}_\mathcal{C}$ let $\text{pb}(p)$ denote the category with objects commutative diagrams of arbitrary pullbacks along $p$ together with morphism pairs $(m_1, m_2) \in \text{Mor}_{\mathcal{C} \times \mathcal{C}} \times \text{Mor}_{\mathcal{C} \times \mathcal{C}}$ such that the rear square in Figure 8 commutes. Note that by the decomposition property of pullbacks the rear square is a pullback, too.

\[ \text{... not only chosen pullbacks ...} \]
The monoidal conditions \( \{1\} \) (neutrality and associativity) imply that \( \text{des}(p) \) is the Eilenberg-Moore Category associated with the monad \( \mathcal{T}^p \). Thus, there is the comparison functor \( \Phi^p : \mathcal{C}_B \to \text{des}(p) \) \( \{1\} \). Obviously \( pb(p) \) is equivalent to \( \mathcal{C}_B \) via chosen pullbacks, such that we obtain a functor\(^7\)

\[
\Phi^p : pb(p) \to \text{des}(p).
\]

In order to compute this functor, let us consider an arbitrary pullback \( (q, \gamma) \) of a co-span \( (p, \alpha) \) in \( \mathcal{C} \), cf. Fig. 9. Computing \( \mathcal{T}^p \gamma \) using the chosen pullback \( (p^p \circ p \circ \gamma = \mathcal{T}^p \gamma, \pi_2 := \pi_2^{po}_2) \) of \( (p \circ \gamma, p) \) yields a unique \( \xi^p : \mathcal{T}^p \gamma \to \gamma \) such that

\[
q \circ \xi^p = q \circ \pi_2.
\]

From \( \{2\}, \{5\}, \) and the uniqueness of mediating morphisms for the original pullback one easily deduces

\[
\xi^p \circ \eta^p = \text{id}_C.
\]

Let \( \pi_2 := \pi_2^{po}_2 \), then \( \mathcal{T}^p \xi^p : (\mathcal{T}^p)^2 \gamma \to \mathcal{T}^p \gamma \) is unique with \( \xi^p \circ \pi_2 = \pi_2 \circ \mathcal{T}^p \xi^p \), such that a similar argumentation together with the second equation in \( \{2\} \) and \( \{5\} \) yields

\[
\xi^p \circ \mathcal{T}^p \xi^p = \xi^p \circ \mu^p_2.
\]

Hence \( \xi^p \) fulfills \( \{1\} \). Thus the original pullback is mapped to \( (\gamma, \xi^p) \), an object of \( \text{des}(p) \). An investigation of the general construction of \( \Phi^p \) \( \{1\} \) shows that our mapping reflects this construction where

\[
\Phi^p(m_1, m_2) = (m_1, \mathcal{T}^p m_1)
\]

on arrows. In the sequel, \( (\gamma, \xi^p) \) (or just \( \xi^p \) if \( \gamma \) is fixed) will be called canonical descent data for the pullback of \( \alpha \) along \( p \).

From Fig. 7 we obtain \( p \circ \gamma \circ \pi_2^{po} = p \circ \gamma \circ \xi \) for each \( (\gamma, \xi) \in \text{Ob}_{\text{des}(p)} \). Hence there is a functor \( \Psi^p : \text{des}(p) \to \mathcal{C}_B \) which maps \( (\gamma, \xi) \) to the unique arrow \( \alpha \), which mediates \( p \circ \gamma \) and a chosen coequalizer \( c \) of \( \pi_2^{po} \) and \( \xi \) (cf. also Fig. 10). \( \{1\} \) shows that

i) \( \Psi^p \) is left-adjoint to the comparison functor \( \Phi^p : \mathcal{C}_B \to \text{des}(p) \) with monic co-unit and

ii) if \( p \) is an epimorphism, \( \Phi^p \) becomes an equivalence of categories with pseudo-invers \( \Psi^p \).

We use these facts to state the main result of this section. For this, we need some auxiliary considerations. The following statement is Lemma 20 in \( \{14\} \):

**Lemma 7** Let \( \mathcal{C} \) be a topos and a commutative diagram be given with an epimorphism as indicated. If \( (1) + (2) \) and \( (1) \) are pullbacks, then \( (2) \) is a pullback, too.

\(^7\) To simplify matters, we still use the name \( \Phi^p \) for this functor.
Definition 8 (Equivalence Relation) An equivalence relation on $A \in \text{Ob}_C$ is a pair of arrows $a, b : U \to A$, such that $U \xrightarrow{(a,b)} A \times A$ is a monomorphism, and which is
1. reflexive: $\exists r : A \to U : a \circ r = b \circ r = \text{id}$,
2. symmetric: $\exists s : U \to U : a \circ s = b, b \circ s = a$, and
3. transitive: If $(p : P \to U, q : P \to U)$ is the pullback of $(a,b)$ (especially $b \circ p = a \circ q$), there is $t : P \to U$, such that $a \circ t = a \circ p$ and $b \circ t = b \circ q$.

Lemma 9 $E \times_B C \xrightarrow{(\xi, \pi_2^{\gamma})} C \times C$ establishes an equivalence relation.

Proof: Because $\xi : \pi_1^{\gamma} = T^p \gamma \to \gamma$, it is not difficult to see, that $(\xi, \pi_2^{\gamma})$ is monic. For reflexivity, let $r := \eta^\gamma_p$ and use (1) and (2). Symmetry follows with $s := \xi, (3)$, and (4). Transitivity can be established via $t := \mu_p^\gamma$ (using the commuting top square in Fig. 9 and (1)).

Note that this implies that $(\xi, \pi_2^{\gamma})$ is the kernel pair of its coequalizer, because in topoi, equivalence relations are effective (see [12], A 2.4.1.). Consider now the above introduced coequalizer construction for $\Psi^p$.

Lemma 10 The right square in Figure 10 is a pullback. Hence, $\Psi^p : \text{des}(p) \to \text{pb}(p)$

Proof: By Lemma 9 and because equivalence relations are the kernel pair of their coequalizer, the left square in Figure 10 is a pullback. Because $\gamma \circ \xi = T^p \gamma$ and $\alpha \circ c = p \circ \gamma$ by definition of $\alpha$, the outer rectangle in Figure 10 is the pullback of $p \circ \gamma$ and $p$ as indicated in Figure 7. Since $c$ is epic, the result follows from Lemma 7.

---

8 Again not changing the name for this new functor.
Proposition 11 (Correspondence of Pullbacks and Descent Data)
a) For each choice of coequalizer in the construction of $\Psi^p$ the unit of the adjunction $\Psi^p \dashv \Phi^p : pb(p) \to des(p)$ is the identity.

b) If $p$ is an epimorphism, $\Phi^p : pb(p) \to des(p)$ becomes an equivalence of categories. Moreover, the coequalizer in the construction of $\Psi^p$ can be chosen such that the co-unit is identical.

Proof: We use the facts i. and ii. on page 67. For a) we use Lemma 10: If $(\gamma, \xi) \in \text{Ob} des(p)$, $\Phi^p \Psi^p (\gamma, \xi)$ is unique with (5) (with $q$ replaced by $c$) by the above considerations on $\Phi^p$. But the coequalizer construction also yields $c \circ \xi = c \circ \pi^0_{\gamma}$, such that $(\gamma, \xi) = \Phi^p \Psi^p (\gamma, \xi)$, hence the unit is the identity. To prove b) consider an arbitrary pullback square $sqr$ as in Figure 9. Pullbacks in topoi preserve epimorphisms (8, 5.3) thus, using the isomorphic co-unit, it is easy to show, that the diagram $q \circ \xi \alpha = q \circ \pi^0_{\gamma}$ in the upper right corner of Figure 9 establishes a coequalizer situation. Hence for this choice of coequalizer, $\Psi^p \Phi^p sqr = sqr$, yielding an identical co-unit. $\square$

For future reference, we want to illustrate these facts in the category $SET$. In the following proposition, the first part reformulates neutrality and associativity, whereas the nature of descent data as equivalence relation (on $C$) becomes evident from the second part. For a detailed explanation of this proposition, the reader is referred to the Appendix.

Proposition 12 (Descent Data in SET) Let $\mathcal{C} = SET$.
1. There is a bijective correspondence between objects $(\gamma, \xi)$ of $des(p)$ and families $(\xi_{e,e'})_{(e,e') \in \ker(p)} : \gamma^{-1} e \to \gamma^{-1} e'$ of bijections which satisfy

\[
\xi_{e,e} = id_{\gamma^{-1} e} \quad \text{and} \quad \xi_{e,e''} = \xi_{e,e'} \circ \xi_{e,e''},
\]

for all $(e, e'), (e, e'') \in \ker(p)$.

2. Let $c$ be the coequalizer of $\xi$ and $\pi^0_{\gamma}$. Then

\[
\ker(c) = \{ (x, \xi_{\gamma(x), \gamma(y)}(x)) \mid x, y \in C, (\gamma(x), \gamma(y)) \in \ker(p) \}.
\]

4 Coherence and Van Kampen Squares

In this section we study the interplay of reachability of pullback spans and coherent coexistence of descent data in a general topos $\mathcal{C}$. After having defined these two concepts precisely, we state a local criterion for reachability and a global characterization of Van Kampen squares in terms of coherent algebraic structures. Let a commuting square as in the bottom of Figure 11 be given.

![Diagram](image-url)
Reachability: Because $s = \tau \circ a$, we can decompose any diagonal pullback in $pb(s)$ into a left and a right part by calculating the right part via the chosen $\tau^* \sigma$. This calculation of the left part of the pullback of $pb(s)$ extends to a functor $\Delta^\tau : pb(s) \to pb(a)$. From $s = \sigma \circ r$, we obtain $\Delta^\sigma : pb(s) \to pb(r)$ in the same way. Then we define
\[
PB := \langle \Delta^\tau, \Delta^\sigma \rangle : pb(s) \to pb(a) \times_{e_{LL}} pb(r).
\]
where $pb(a) \times_{e_{LL}} pb(r)$ is the category of all pullback spans over $(a, r)$ together with morphism triples similar to the definition in Figure 3. The name clash of this functor with the functor $PB$ in the introduction is deliberate: Both functors are equal up to an equivalence of categories, because $C \downarrow S \cong pb(s)$.

Definition 13 (Reachability) A pullback span in $pb(a) \times_{e_{LL}} pb(r)$ is said to be reachable, if it is in the image of $PB$ up to a $pb(a) \times_{e_{LL}} pb(r)$-isomorphism.

Coherence: To investigate the counterpart of reachability on the instance level, we apply the methodology of Section 3 to the situation in Figure 11 in which the two back faces are pullbacks. Let $f : L \to B$, $g : B \to S$ be any two arrows in $C$ and let $h := g \circ f$. We consider the pullbacks $f^* (f \circ \gamma)$ and $h^* (h \circ \gamma)$ as in Figure 7 (with $C := I$, $E := L$, and $p : E \to B$ replaced by $f : L \to B$, $h : L \to S$, resp.). Let $\pi^{f \gamma}_2, \pi^{h \gamma}_2$ be the "second projections" in these pullbacks, resp.

For any $\gamma \in \mathcal{C} \downarrow L$ we have $h \circ \gamma \circ \pi^{f \gamma}_2 = h \circ f \gamma$, thus there is a unique $u^\gamma_f : L \times_B I \to L \times_S I$ with
\[
\pi^{h \gamma}_2 \circ u^\gamma_f = \pi^{f \gamma}_2 \quad \text{and} \quad \pi^h \gamma \circ u^\gamma_f = \pi^f \gamma
\]
cf. Figure 12

![Figure 12: Construction of Embedding](image)

Note, that in $SET$, $\pi^f \gamma$ and $\pi^h \gamma$ are first projections, which actually makes $u^\gamma$ invariant under projections: Indeed $L \times_B I = \{(l, i) \mid f(\gamma(i)) = f(l)\} \subseteq \{(l, i) \mid h(\gamma(i)) = h(l)\} = L \times_S I$ where the embedding is $u^\gamma$. This justifies the use of the hooked arrow in Figure 12.

In this way, we obtain 5 embeddings for the original pushout situation:
\[
u^\gamma_f : L \times_A I \to L \times_S I, \quad u^\gamma_L : L \times_R I \to L \times_S I
\]
(Using $s = \sigma \circ r = \tau \circ a$ instead of $h = g \circ f$) as well as
\[
u^\gamma_f : L \times_L I \to L \times_R I, \quad u^\gamma_L : L \times_L I \to L \times_A I, \quad u^\gamma_r : L \times_L I \to L \times_S I
\]
(Using $r = r \circ id_L, a = a \circ id_L, \text{and} \ s = s \circ id_L$) with corresponding projection compatibility and uniqueness as in (7). The uniqueness property easily yields compositionality:
\[
\forall \gamma \in \mathcal{C} \downarrow L : u^\gamma_f \circ u^\gamma_r = u^\gamma_r \circ u^\gamma_f \tag{8}
\]
It can easily be shown that $u^\gamma_f$ are monomorphisms, but we can do better (see the Appendix for a proof):
Lemma 14 Let \( L \xrightarrow{f} B \xrightarrow{g} S \) be given with \( h := g \circ f \). \( u^g : \mathcal{T}f \Rightarrow \mathcal{T}h \) is a monad monomorphism.

Lemma 15 Let \( f, g, h \) be as in Lemma 14 There is a full and faithful functor \( U^g : \text{des}(h) \rightarrow \text{des}(f) \) for which
\[
U^g(\gamma, \xi) = (\gamma, \xi \circ u^g).
\]

Proof: Since \( \text{des}(p) \) is the category of Eilenberg-Moore-Algebras associated with \( \mathcal{T}p \), the result follows from Lemma 14 and the proof of a theorem of Barr and Wells ([2], Theorem 6.3 in Chapter 3)

Let us fix the rear pullback span \( PBS \) in Figure 11. Since \( \gamma \) is fixed, considered objects of \( \text{des}(r) \), \( \text{des}(a) \), and \( \text{des}(s) \) will always have codomain \( \gamma \), hence \( \xi^\beta \) and \( \xi^\tau \) are appropriate abbreviations for the two canonical descent datas (cf. Section 3) arising from the two pullbacks.

Definition 16 (Coherence) \( \xi^\tau \) and \( \xi^\beta \) are called coherent, if there is \( (\gamma, \xi) \in \text{des}(s) \), such that
\[
\langle U^\gamma, U^\beta \rangle (\gamma, \xi) = (\xi^\tau, \xi^\beta) \tag{9}
\]

We call any \( (\gamma, \xi) \in \text{des}(s) \) with this property a coherence witness (for \( \xi^\tau \) and \( \xi^\beta \)).

Thus two algebraic structures are coherent, if there is an algebraic structure over \( \gamma \) relative to \( s \) which effectively approximates them. We are ready to state the main technical result of this section:

Proposition 17 (Reachability vs. Coherence) Let in a topos \( \mathcal{C} \) a diagram be given as in Figure 7 where the bottom square is commutative and the rear faces form a pullback span. Let \( \xi^\tau \) and \( \xi^\beta \) be the above introduced canonical descent datas.

a) If the span is reachable, \( \xi^\tau \) and \( \xi^\beta \) are coherent.

b) If the bottom square is a pushout and \( \xi^\tau \) and \( \xi^\beta \) are coherent, then the span is reachable.

c) Under the prerequisites of b), the coherence witness is unique.

Proof: To simplify matters we write \( u \) instead of \( u_\gamma \). To show a), let \( \text{sqr}_{\text{diag}} \in \text{pb}(s) \) (the pullback \( (\gamma, s') \) of \( (\sigma, s) \)) with \( \text{PB}(\text{sqr}_{\text{diag}}) \) being the rear pullback span (this is Figure 11 without question marks). \( \xi^\tau \) is unique with \( a' \circ \xi^\tau = a' \circ \pi_2^{\mathcal{T}\gamma} \), such that for the first projection in (9) it suffices to show validity of this equation with \( \xi^\tau \) replaced by \( \xi \circ u^\tau \). The argumentation for the second projection is then similar. We have
\[
\tau \circ a' \circ \xi \circ u^\tau = a \circ \gamma \circ \xi \circ u^\tau \quad \text{Left rear pullback in Figure 11}
\]
\[
= a \circ \mathcal{T}^\gamma \gamma \circ u^\tau \quad \text{Since } \xi : \mathcal{T}^\gamma \gamma \rightarrow \gamma
\]
\[
= a \circ \mathcal{T}^\gamma \gamma \quad \text{By (7)}
\]
\[
= a \circ \gamma \circ \xi^\tau \quad \text{Since } \xi^\tau : \mathcal{T}^\gamma \gamma \rightarrow \gamma
\]
\[
= \tau \circ a' \circ \pi_2^{\mathcal{T}^\gamma \gamma} \quad \text{Left rear pullback and (5) for } \xi^\tau
\]

and also \( \tau' \circ a' \circ \xi \circ u^\tau = \tau' \circ a' \circ \pi_2^{\mathcal{T}^\gamma \gamma} \circ u^\tau = \tau' \circ a' \circ \pi_2^{\mathcal{T}^\gamma \gamma} = \tau' \circ a' \circ \pi_2^{\mathcal{T}^\gamma \gamma} \) (by (5) for \( \xi \) and (7)). This implies the desired result, because in the front face pullback \( \tau \) and \( \tau' \) are jointlymonic.

To show b), assume we already knew the result in the case \( a \) and \( r \) are both epimorphisms. We can then use epi-mono-factorizations \( r = r_m \circ r_c \) and \( a = a_m \circ a_e \) (which exist in topos) to decompose both back

---

\[\text{If } \text{PB}(\text{sqr}_{\text{diag}}) \text{ yields the rear pullback span not exactly but only up to isomorphism, we can exchange the instances over } A \text{ and } R \text{ by their compositions with the isomorphisms, such that there is a complete cube with 4 pullbacks having the original pullback span as rear faces. It is no problem that front and right pullbacks are no longer chosen.}\]
face pullbacks into two pullbacks resp. It can then be verified that the bottom face can be decomposed into 4 pushouts along these epi-mono-factorizations. Because $r_e$ and $a_e$ are both epic and $\xi^r$ and $\xi^a$ are also canonical descent data of $r_e$ and $a_e$ (this follows from fact i. on page 67), the inner pullback span is reachable. Since topoi are adhesive \cite{14}, this reachability can be continued along the other pairs of bottom arrows (of which either one or both are now monic) by constructing top face pullouts.

Thus it suffices to assume that $r$ and $a$ are epic. Then $r'$ in Figure 11 is the appropriate coequalizer of $\xi^a$ and $\pi_2^{\circ a}$ by Proposition 11 b). Let $s' : I \to K$ be the coequalizer of $\pi_2^{\circ a}$ and the coherence witness $\xi$ with $sqr_{diag} := \Psi^r(\gamma, \xi)$ the resulting diagonal pullback by Lemma 10. By coherence and (7)

$$s' \circ \xi^a = s' \circ \pi_2^{\circ a}$$

yielding a unique mediator $\pi' : H \to K$ for the coequalizer $r'$, i.e.

$$\pi' \circ r' = s'.$$

(10)

Let $\sigma \in C - S$ be part of $sqr_{diag}$ as indicated in Figure 11. Then by construction and (10) $\pi \circ \beta \circ r' = \pi \circ r \circ \gamma = s \circ \gamma = s \circ s' = s \circ \pi' \circ r'$, hence we obtain a commutative square as right face of the cube in Figure 11 (the coequalizer $r'$ is an epimorphism which is also a pullback by Lemma 7 i.e. $\pi' \circ \sigma \cong \beta$. Analogously one shows $r' \circ \sigma \cong \tau$.

To show c) assume that there are two coherence witnesses $(\gamma_1, \xi_1), (\gamma_2, \xi_2)$. Clearly $\gamma := \gamma_1 = \gamma_2$ by Lemma 15 such that it remains to show $\xi_1 = \xi_2$. By b), $\xi_1$ and $\xi_2$ yield two cubes each of which possess 4 pullbacks as side faces. They possess the same arrows except $\pi', r'$, and $\sigma$. But the two variants of the arrows $\pi', r'$ both form a top pushout of $a', r'$ because, in topoi, pullbacks preserve colimits. Hence there is an isomorphism $\pi$ which can be shown to mediate between the two variants of $\sigma$.

Consequently, we have two diagonal pullbacks $sqr_{diag}^{1, 2} = \Psi^r(\gamma, \xi_1)$ and $sqr_{diag}^{2} = \Psi^a(\gamma, \xi_2)$ (see part b)) for which by (9) and Proposition 11 a)

$$\Phi^{\pi}(sqr_{diag}^{1, 2} \xrightarrow{(id,id)} sqr_{diag}^{2}) = (\gamma, \xi_1) \xrightarrow{(id,\pi')} (\gamma, \xi_2)$$

which yields $\xi_1 = \xi_2$. \hfill \Box

By the remark after Lemma 9 any descent data $(\gamma, \xi) \in des(p)$ yields the kernel pair $ker(q) := (\xi, \pi_2)$ of the top arrow $q$ of $\Psi^p(\gamma, \xi)$, see Figure 9. In the category $Eq(C)$ of equivalence relations on $C \in Ob_e$ (i.e. the full subcategory of $C - (C \times C)$ of arrows with the properties of Definition 8), we call an object $e$ an upper bound of $e_1$ and $e_2$, if there are $Eq(C)$-arrows (necessarily monos) $v_1$ and $v_2$ with

$$e \circ v_1 = e_1 \text{ and } e \circ v_2 = e_2.$$  

(11)

It is well-known \cite{2} that the least upper bound (lub) of two equivalence relations $ker(a') : X \to C \times C, ker(r') : Y \to C \times C$ can be constructed by extracting the mono part $m$ of $[ker(a'), ker(r')] : X + Y \to C \times C$ followed by constructing the kernel pair of the coequalizer of $m$. Let $\pi_2$ be the second projection in the pullback associated with the monad $\mathbb{T}^\circ$.

**Lemma 18** Let the bottom square in Figure 11 be a pushout. $\xi^a$ and $\xi^b$ are coherent if and only if there is $(\gamma, \xi) \in des(s)$ with

$$lub(ker(a'), ker(r')) \cong (\xi, \pi_2)$$

Proof: If $\xi^a$ and $\xi^b$ are coherent, then the coherence witness $\xi$ from Proposition 17 b) and c) was used to complete the pullback cube. Since, in topoi, the top face becomes a pushout, and $(\xi, \pi_2)$ is the

...
kernel pair of the top diagonal, using the universal property of pushouts, it can easily be shown that 
\((\xi, \pi_2^I) \cong \text{lub}(\ker(a^I), \ker(r^I))\).

The opposite direction follows directly from the uniqueness properties (7) of the monad morphisms 
\(u^I\) and \(u^I\): Any mediating monomorphisms \(v_1, v_2\) as in (11) in the least upper bound constellation must
coincide with \(u^I, u^I\), resp.

**Theorem 19 (Answer to Question 1)** Let \(\mathcal{C}\) be a topos and a pullback span be given as in the rear of
Figure [11] with the bottom square a pushout. The span is reachable if and only if \(\text{lub}(\ker(a^I), \ker(r^I)) \cong 
(\xi, \pi_2^I)\) for some \((\gamma, \xi) \in \text{des}(s)\).

**Proof:** This follows from Proposition [17] and Lemma [18].

Thus there is an algorithm to check reachability: Given a rear pullback span PBS with top arrows \(a^I, r^I\)

1. Compute \(e := \text{lub}(\ker(a^I), \ker(r^I))\).

2. Check the monadicity requirements (1) of \(e\) relative to \(s\) by interpreting it as a pair \((\xi, \pi_2^I)\).

3. PBS is reachable if and only if \(e\) meets the requirements.

In the next section we will recall the introductory examples from Section [2] such that these theoretical
results become more evident.

We conclude this section with a global statement on Van Kampen squares in the spirit of Figure 3.
We still assume a square as the bottom in Figure [11] to be given. In the following, the category \(\text{des}(\text{id}_L)\)
is integrated. It represents the "common part" of the forgetful functors \(U^a\) and \(U^r\), namely the carrier \(\gamma\)
represented by certain isomorphisms from the "graph" \(L \times_L I\) of \(\gamma\) to \(I\).

**Theorem 20 (Fibred Version of Amalgamation Lemma)** Let \(\mathcal{C}\) be a topos. In Figure [13] the pushout
(1) is a Van Kampen square if and only if (2) is a pullback in \(\text{CAT}\).

\[
\begin{array}{ccc}
L & \xrightarrow{r} & R \\
a & \downarrow & \pi \\
A & \xrightarrow{\tau} & S \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{des}(\text{id}_L) & \xleftarrow{U^r} & \text{des}(r) \\
U^a & \downarrow & U^\pi \\
\text{des}(a) & \xleftarrow{U^\tau} & \text{des}(s) \\
\end{array}
\]

Figure 13: Amalgamation Lemma (Fibred setting)

**Proof:**

"\(\Rightarrow\):" By (8) and Lemma [15] (2) commutes. By assumption, \(PB\) is an equivalence of categories, 
i.e. each rear pullback span is reachable. By Proposition [17] a) each pair \((\xi^T, \xi^B)\) is coherent and by
Proposition [17] c) the coherence witness is unique. Standard arguments together with the fact that \(U^B\)
are full and faithful functors (Lemma [15]) yield the pullback property.

"\(\Leftarrow\):" The pullback property immediately yields coherence for each pair \(((\gamma, \xi^T), (\gamma, \xi^B)) \in \text{des}(a) \times 
\text{des}(r)\). Because (1) is a pushout, Proposition [17] b) implies reachability of each rear pullback span, thus
\(PB\) is essentially surjective, which is sufficient for (1) to have the Van Kampen property [18].

\(\square\)
5 Coherence and Van Kampen Squares in SET and GRAPH

This section illustrates the use of Theorem 19 and develops a simply checkable characterization of Van Kampen squares in SET and GRAPH based on Theorem 20. As mentioned before, in SET, the u’s are natural embeddings. Hence coherence (cf. Definition 16) yields the existence of descent data \( \xi \) relative to \( s = \overline{a} \circ r = \overline{r} \circ a \) with

\[
\forall (x, x') \in ker(r) : \xi_{x, x'} = \xi^\beta_{x, x'} \quad \text{and} \quad \forall (y, y') \in ker(a) : \xi_{y, y'} = \xi^\tau_{y, y'}
\]  

(12)

where all mappings are understood as the components of the families of bijections from Proposition 12.

We can now observe Theorem 19 at work: Recall the situation in Figure 4. By Proposition 12, \( \xi^\beta \) and \( \xi^\tau \) map along the dashed and dotted lines, resp. E.g. \( \xi^\beta_{1, y}(1 : x) = 2 : y, \xi^\tau_{1, y}(2 : x) = 1 : y \). Reachability means that the least upper bound of the kernels of \( a' \) and \( r' \) yield a monadic structure \( \xi \) relative to \( s \). By (12) and hypothetical associativity (cf. Proposition 12) of \( \xi \) the bijection \( \xi_{x, y} \) must be equal to \( \xi^\tau_{w, y} \circ \xi^\beta_{w, x} \circ \xi^\tau_{x, z} \) on the fibre over \( x \). But this must then coincide with \( \xi^\beta_{x, y} \), which is not the case in Figure 4.

Obviously, the kernels of \( r \) and \( a \) are intertwined through the cycle \( (x, z), (z, w), (w, y), (y, x) \in ker(s) \) and are thus not enough separated. The following definition makes this more precise:

**Definition 21 (Separated Kernels)** Let \( \mathfrak{C} = \text{SET} \) and \( a \) and \( r \) be given as in Figure 17. A sequence \( (x_i)_{i \in \{0, 1, \ldots, 2k + 1\}} \) of elements in \( L \) is called a domain cycle (of \( a \) and \( r \), if \( k \in \mathbb{N} \) and the following conditions hold:

1. \( \forall j \in \{0, 1, \ldots, 2k + 1\} : x_j \neq x_{j+1} \)
2. \( \forall i \in \{0, \ldots, k\} : (x_{2i+1}, x_{2i+2}) \in ker(a) \)
3. \( \forall i \in \{0, \ldots, k\} : (x_{2i+2}, x_{2i+3}) \in ker(r) \)

where the sums are understood modulo \( 2k + 2 \). We call \( 2k + 2 \) the length of the domain cycle. Moreover, a domain cycle is proper if we have for all \( i, j \in \{0, 1, \ldots, 2k + 1\} \) that \( x_i \neq x_j \) if \( i \neq j \).

The pair \( a \) and \( r \) is said to have separated kernels, if it has no domain cycle.

**Remark 1:** It is easy to see that each domain cycle \( c \) possesses a proper subcycle, i.e. a proper cycle with smaller or equal length than the length of \( c \) and whose elements are a subset of the elements of \( c \).

**Remark 2:** “Having separated kernels” is only sufficient but not necessary for “being jointly monic”. Indeed, being not jointly monic induces a domain cycle of length 2. But longer domain cycles occur for jointly monic \( a \) and \( r \) (see Figure 4).

Domain cycles are connected to coherence as follows:

**Proposition 22** Let \( \mathfrak{C} = \text{SET} \) and a commutative square be given like the bottom square in Figure 17 and let the two rear faces be pullbacks with canonical descent data \( \xi^\tau \) and \( \xi^\beta \), resp. \( \xi^\tau \) and \( \xi^\beta \) are coherent iff for all domain cycles \( (x_i)_{i \in \{0, 1, \ldots, 2k + 1\}} \) of \( a \) and \( r \) we have

\[
\xi^\beta_{x_{2k+1}, x_0} \circ \xi^\tau_{x_{2k+2}, x_{2k+1}} \circ \cdots \circ \xi^\tau_{x_3, x_2} \circ \xi^\beta_{x_1, x_2} \circ \xi^\tau_{x_0, x_1} = id_{\gamma^{-1}x_0}
\]  

(13)

The statement is illustrated in Example 2 where coherence is now achieved by harmonizing the equivalences of \( a' \) and \( r' \) in the two copies of \( L \) that make up the domain of \( \gamma \). Alternatively, we can use Theorem 19 to check reachability: The least upper bound yields a descent data for \( \gamma \) relative to \( s \), because it is evident that neutrality and associativity are not destroyed. In order not to interrupt the flow of arguments, the proof of Proposition 22 is contained in the Appendix.

The next proposition illustrates how domain cycles are connected to reachability. This time we include the proof, because it demonstrates the use of descent data.
Proposition 23 Let $\mathcal{C} = \text{SET}$ and a commutative square be given like the bottom square in Figure 11. If all pullback spans in the rear are reachable, $a$ and $r$ have separated kernels.

Proof: Assume to the contrary that $a$ and $r$ possess a domain cycle $(x_i)_{i \in \{0,1,\ldots,2k+1\}}$ for some $k \in \mathbb{N}$. By the first remark after Definition 21 we can assume that this cycle is proper. Let $\Omega = \{0,1\}$ and $\gamma := \tau_2 : \Omega \times L \to L$ be the ordinary second projection. We construct descent data $\xi^a$ for $\gamma$ relative to $a$ and $\xi^r$ for $\gamma$ relative to $r$: Because the fibre of $\gamma$ over $x = \{(0, x), (1, x)\}$ we can define $\xi^r_{x,x'} : (b, x') := (b, x')$ for all $(x, x') \in \text{ker}(r)$ and $b \in \{0,1\}$. It is obvious that this yields neutrality and associativity of Proposition 12.

Consider now the equivalence class $E_0 = \{x \in L \mid a(x) = a(x_0)\}$ of $\text{ker}(a)$, where $x_0$ is the begin of the cycle. The domain cycle has at least length 2, hence we have $x_1 \neq x_0$, $x_1 \in E_0$ in the cycle. For any $x \in E_0$ we define a bijection $\xi^a_{x_0, x} : \{0,1\} \times \{x_0\} \to \{0,1\} \times \{x\}$ by

$$\xi^a_{x_0, x}(b, x_0) := \begin{cases} (b, x) & \text{if } x \neq x_1 \\ (1-b, x) & \text{if } x = x_1 \end{cases}$$

Further we set

$$\xi^a_{x,x'} := \xi^a_{x_0,x} \circ (\xi^a_{x_0,x})^{-1} \quad \text{for all } x, x' \in E_0, x \neq x_0.$$

Neutrality and associativity are straightforwardly ensured by these definitions. For $(x, x') \in \text{ker}(a) - E_0^2$ we define $\xi^a_{x,x'}$ in the same way as $\xi^r$.

By Proposition 11 a), $\xi^\beta := \xi^r$ and $\xi^\gamma := \xi^a$ are canonical descent datas of the pullbacks $\Psi^r(\gamma, \xi^r)$ and $\Psi^a(\gamma, \xi^a)$, resp, such that for the resulting pullback span we obtain:

$$(\xi^\beta^\gamma_{x_1,x_1} \circ \xi^\gamma_{x_1,x_{2k+1}} \circ \cdots \circ \xi^\gamma_{x_{2k+1},x_1} \circ \xi^\beta^r_{x_0,x_1})(0, x_0) = (1, x_0)$$

because, in this chain, $\xi^\beta$ always preserves the first projection and $\xi^\gamma$ interchanges it only if $x = x_0$ and $x' = x_1$ since the cycle is proper. Thus, by Proposition 22 $\xi^\gamma$ and $\xi^\beta$ are not coherent, hence, by Proposition 17 the pullback span is not reachable contradicting the assumption.

The following theorem is the main result of this section (cf. [16]):

Theorem 24 Let $\mathcal{C} = \text{SET}$ or $\mathcal{C} = \text{GRAPH}$. A pushout diagram as the bottom square in Figure 11 is a Van Kampen square if and only if $a$ and $r$ have separated kernels.

Proof: ”$\Rightarrow$” follows from Theorem 20 and Propositions 17 and 23. ”$\Leftarrow$” follows from Proposition 22 and Theorem 20. It is shown in [16] that the argumentation easily carries over to graphs once the result has been proven for $\text{SET}$.

Recall Example 3 where we can now easily derive from Theorem 24 that each pullback span is reachable, i.e. each amalgamation of instances is successful.

6 Outlook

The paper presents first outcomes of a more comprehensive collaborative project based on [3, 16, 19] and addressing ”compositional fibred semantics in topoi”. There are several topics for future research: First we have to address persistency requirements and extension lemmas for fibred semantics. Moreover, we are looking for a categorical generalization of Proposition 22 which would give rise, due to Proposition 17 to a kind of ”conditional compositionality”.

An interesting open question, in this context, is to characterize domain cycles on a pure categorical level. This should yield an elementary characterization of Van Kampen squares in more general categories in the spirit of Theorem 24.
7 Appendix

Descent Data in SET: Here, we give details about the different view on descent data from Proposition 12 in SET. First, we remind that pullbacks, in general, can be described as products in slice categories. For the situation in Definition 4, this means that the diagonal $p \circ \mathcal{I} \gamma = p \circ \gamma \circ \pi_2^{p \circ \gamma} : E \times_B C \to B$ forms the product $p \times (p \circ \gamma)$ in $\mathcal{C} \downarrow B$ with projections $\mathcal{I} \gamma : p \times (p \circ \gamma) \to p$ and $\pi_2^{p \circ \gamma} : p \times (p \circ \gamma) \to p \circ \gamma$. Second, any $\xi : E \times_B C \to C$ in $\mathcal{C}$ which is an arrow $\xi : \mathcal{I} \gamma \to \gamma$ in $\mathcal{C} \downarrow B$ establishes also an arrow $\xi : p \times (p \circ \gamma) \to p \circ \gamma$ in $\mathcal{C} \downarrow B$. $\mathcal{C} \downarrow B$, however, is also a topos by the fundamental theorem of Freyd [7] and thus, especially cartesian closed. In $\mathcal{C} = SET$, finally, any $\xi \in \text{Hom}(p \times (p \circ \gamma), p \circ \gamma) \cong \text{Hom}(p, (p \circ \gamma)^{p \circ \gamma})$ can be interpreted as a map that assigns to any element $e \in E$ an endomap $\xi(e, \cdot)$ of the fibre of $p \circ \gamma$ over $p(e)$ (cf. [8], Chapter 4).

For our purposes, an appropriate representation of these maps for descent data will be in terms of the kernel of $p$: The fibre of $p \circ \gamma$ over $p(e)$ is the pre-image of the equivalence class $[e]_{\ker(p)}$ w.r.t. $\gamma$. Let $\xi_{e,e'}$ be the restriction of the map $\xi(e, \cdot)$ to $\gamma^{-1}e$ whenever $(e, e') \in \ker(p)$. If $c \in \gamma^{-1}e$ we obtain $\gamma(\xi(e', c)) = e'$ from Definition 4 hence the codomain of $\xi_{e,e'}$ is $\gamma^{-1}e$ and $\xi$ represents a family

$$\xi(e', c) = \xi_{e,e'}(c) \text{ for } \gamma(c) = e. \tag{15}$$

Let us now investigate the influence of neutrality and associativity [1] to this family. A canonical choice of pullbacks in SET yields

$$E \times_B C = \{(e, c) \in E \times C \mid p(e) = p(\gamma(c))\},$$

$$E \times_B (E \times_B C) = \{(e, (e', c)) \in E \times (E \times C) \mid p(e) = p(e') = p(\gamma(c))\},$$

and

$$\eta_f^p(c) = (\gamma(c), c), \mu_f^p(e'', (e', c)) = (e'', c), \mathcal{I} \gamma \xi(e'', (e', c)) = (e'', \xi(e', c)). \tag{16}$$

Thus for all $(e, e'), (e'', e''') \in \ker(p)$ and $c \in \gamma^{-1}e$, [1] and the first equation in (16) yield

$$\xi_{e,e}(c) = c,$$

whereas the second equation in (1) (applied to a triple $(e'', (e', c))$) and the second and third equation of (16) imply

$$\xi_{e', e''}(\xi_{e,e'}(c)) = \xi_{e,e''}(c).$$

By choosing $e'' = e$, these two equations force each $\xi_{e,e'}$ to be bijective.

By reversing the whole argumentation, we can also show that any family as in (14) which satisfies these two equations yields a descent data by defining $\xi$ as in (15) for $e := \gamma(c)$. Altogether we obtain the statement in Proposition 12 which subsumes the monoidal nature of descent data in SET. Moreover, 2 follows from effectiveness of equivalence relations and (15).

Proof of Lemma 14: For simplicity we write $u$ instead of $u^e$. There are several statements to prove:

1. Each $u_\gamma$ is a monomorphism.
2. $u : \mathcal{I} f \Rightarrow \mathcal{I} h$ is a natural transformation.
3. $u$ is compatible with units, i.e. $u \circ \eta_f = \eta_h$.
4. \( u \) is compatible with co-units, i.e. \( \mu^h \circ u^2 = u \circ \mu^f \) where \( u^2 \) is the horizontal composition of \( u \) with itself.

1. To show that \( u_\gamma \) is monic for each \( \gamma \), let \( x, y : X \to L \times_B I \) with \( u_\gamma \circ x = u_\gamma \circ y \) be given. By (7), one computes \( T^f \gamma \circ x = T^f \gamma \circ y \) and \( \pi_2^f \circ x = \pi_2^f \circ y \). Because \( T^f \gamma \) and \( \pi_2^f \) are jointly monic (being a limit cone in a pullback square), we obtain \( x = y \). In the sequel, the property of a pullback cone to be jointly monic will be used several times. We will do this without further reference.

2. Let \( \gamma, \hat{\gamma} \in C \downarrow L \) and
\[
\phi : \gamma \to \hat{\gamma}
\]
be a \( C \downarrow L \)-morphism. As before, \( \pi_2 \) and \( \pi_2' \) denote the second projections in the pullbacks involving \( \gamma \) and the monads \( T^f \) and \( T^h \), resp. \( \hat{\pi}_2 \) and \( \hat{\pi}_2' \) denote the second projections in the pullbacks involving \( \hat{\gamma} \).

Pulling back \( \phi \) (as an arrow in \( C \downarrow B \) and as an arrow in \( C \downarrow S \)) yields
\[
\phi \circ \pi_2 = \hat{\pi}_2 \circ T^f \phi
\tag{17}
\]
and
\[
\phi \circ \pi_2' = \hat{\pi}_2' \circ T^h \phi
\tag{18}
\]
Let now \( d_1 = T^h \phi \circ u_\gamma \) and \( d_2 = u_\gamma \circ T^f \phi \), which are both arrows from \( T^f \gamma \) to \( T^h \hat{\gamma} \). \( d_1 = d_2 \) (and thus the desired result) follows from
\[
T^h \hat{\gamma} \circ d_1 = T^h (\hat{\gamma} \circ \phi) \circ u_\gamma
\]
\[
= T^h \gamma \circ u_\gamma
\]
\[
= T^f \gamma
\]
\[
= T^f \hat{\gamma} \circ T^f \phi
\]
\[
= T^h \hat{\gamma} \circ u_\gamma \circ T^f \phi
\]
\[
= T^h \hat{\gamma} \circ d_2
\]
and
\[
\hat{\pi}_2' \circ d_1 = \hat{\pi}_2' \circ T^h \phi \circ u_\gamma
\]
\[
= \phi \circ \hat{\pi}_2' \circ u_\gamma
\]
\[
= \phi \circ \pi_2
\]
\[
= \pi_2 \circ T^f \phi
\]
\[
= \pi_2' \circ u_\gamma \circ T^f \phi
\]
\[
= \pi_2' \circ d_2
\]
In the sequel we denote projections with \( \pi_2, \overline{\pi}_2 \) in pullbacks along \( f \) and with \( \pi_2', \overline{\pi}_2' \) in pullbacks along \( h \).

3. Compatibility with the units follows from \( \pi_2' \circ u_\gamma \circ \eta_\gamma^f = \pi_2 \circ \eta_\gamma^f = id = \pi_2' \circ \eta_\gamma^h \) (apply (17) and (2) twice) and \( T^h \gamma \circ u_\gamma \circ \eta_\gamma^f = T^f \gamma \circ \eta_\gamma^f = \gamma = T^h \gamma \circ \eta_\gamma^h \) (again using (7) and the fact, that \( \eta^p_\gamma : T^p \gamma \to \gamma \) for \( p \in \{ f, h \} \)).

4. Let \( u^2 := u \ast u \) be the horizontal composition. By the definition of \( u^2 \) we have for each \( \gamma \in C \downarrow L \):
\[
u_{\gamma} = u \circ T^f u_\gamma = T^h u_\gamma \circ u_{T^f \gamma}.
\tag{19}
From Fig. 9, we get

\[ \pi_2 \circ \pi_2 = \pi_2 \circ \mu^p_I \]  

(20)

where \( \mu^p_I = p^* \pi_2 \). In the sequel, we use this for \( p := f \) and \( p := h \). The diagrams

\[
\begin{array}{ccc}
L \times S \times (L \times S) & \xrightarrow{\mu^h = h^* \pi_2} & L \times S \\
\downarrow \pi_2 & & \downarrow \pi_2 \\
L \times B & \xrightarrow{\pi_2} & L
\end{array}
\]

Figure 14: Compatibility with co-unit, part 1

and

\[
\begin{array}{ccc}
L \times B \times (L \times B) & \xrightarrow{\pi_2} & L \times B \\
\downarrow \pi_2 & & \downarrow \pi_2 \\
L \times S \times (L \times B) & \xrightarrow{\mu^h = h^* \pi_2} & L \times S \\
\downarrow \pi_2 & & \downarrow \pi_2 \\
L & \xrightarrow{h \circ \pi_2} & S
\end{array}
\]

Figure 15: Compatibility with co-unit, part 2

commute: In the first diagram, the triangle commutes by applying \( h^* \) to (7) interpreted as diagram in \( C \downarrow S \). The square is just the pullback which arises from pulling back \( \pi_2 : h \circ \pi_2 \rightarrow h \circ \gamma \) along \( h \). We denote with \( \tilde{\pi}_2 \) the second projection in this case.

The second diagram is just Figure 12 taken at \( \pi_2 \) instead of \( \gamma \) where the same \( \tilde{\pi}_2 \) occurs again. Thus

\[
\pi_2' \circ \mu^h_\gamma \circ u^2_\gamma = \pi_2' \circ \mu^h_\gamma \circ \pi_2 \circ \tilde{\pi}_2 \circ u^2_{\pi_2 \gamma} \quad \text{By (19)} \\
\pi_2' \circ \mu^h_\gamma \circ u^2_\gamma = \pi_2' \circ \tilde{\pi}_2 \circ u^2_{\pi_2 \gamma} \quad \text{Figure 14} \\
\pi_2' \circ \mu^f_\gamma = \pi_2 \circ \tilde{\pi}_2 \quad \text{Figure 15} \\
\pi_2' \circ \mu^f_\gamma = \pi_2' \circ \mu^f_\gamma \quad \text{By (20)} \\
\pi_2' \circ u^2_\gamma \circ \mu^f_\gamma = \pi_2' \circ u^2_\gamma \quad \text{By (7)}
\]

On the other hand, by (7) and the fact that \( \mu^f \) and \( \mu^h \) are \( \gamma \)-indexed families of arrows from \((\pi_2')^2 \gamma \) to \((\pi_2)^2 \gamma \) and \((\pi_2')^2 \gamma \) to \((\pi_2)^2 \gamma \), resp., we obtain

\[
\pi_2' \circ u^2_\gamma \circ \mu^f_\gamma = \pi_2' \circ u^2_\gamma \circ \pi_2' \circ \mu^f_\gamma = (\pi_2)^2 \gamma.
\]

Since \( u^2 \) is a \( \gamma \)-indexed family of arrows from \((\pi_2')^2 \gamma \) to \((\pi_2')^2 \gamma \), we also have

\[
\pi_2' \circ u^2_\gamma \circ \mu^f_\gamma = (\pi_2')^2 \gamma \circ u^2_\gamma = (\pi_2)^2 \gamma.
\]

Because \( \pi_2' \) and \( \pi_2' \) are jointly monic, the proof is complete. \( \square \)
Proof of Proposition 22: "\(\Rightarrow\)" follows immediately from (12) and Proposition 12 applied to the coherence witness \(\xi\).

"\(\Leftarrow\)": We call a sequence \((y_i)_{i \in \{0,1,\ldots,m\}}\) of elements in \(L\) an alternating sequence (of \(a\) and \(r\)), if \(m \in \mathbb{N}\) and the following conditions hold:

a) for all even \(i \in \{0,\ldots,m-1\}\) : \((y_i, y_{i+1}) \in \ker(p)\)

b) for all odd \(i \in \{0,\ldots,m-1\}\) : \((y_i, y_{i+1}) \in \ker(-p)\)

where \(p \in \{a, r\}\) and \(-a = r\) and \(-r = a\). \(m + 1\) is called the length of the sequence. A sequence is called proper if \(y_i \neq y_j\) for all \(i \in \{0,1,\ldots,m\}\) and \(j \in \{0,1,\ldots,m-1\}\) with \(i \neq j\).

For the rear pullback span with canonical descent data \(\xi^\beta, \xi^\gamma\), we define for any alternating sequence \(\sigma = (y_i)_{i \in \{0,1,\ldots,m\}}\) a bijection \(\xi_\sigma : \gamma^{-1} y_0 \to \gamma^{-1} y_m\) as follows: For \(m = 0\): \(\xi_\sigma := id_{\gamma^{-1} y_0}\) and for \(m \geq 1\)

\[
\xi_\sigma := \xi_{y_{m-1}, y_m} \circ \cdots \circ \xi_{y_1, y_2} \circ \xi_{y_0, y_1}
\]

where \(\sigma \neq \gamma\) and \(r = \beta\).

(21)

Obviously, for a domain cycle \(c = (x_i)_{i \in \{0,1,\ldots,2k+1\}}\)

\[
\sigma_c = (x_0, x_1,\ldots, x_{2k+1}, x_0)
\]

is an alternating sequence, thus we can reformulate condition (13) as \(\xi_{\sigma_c} = id_{\gamma^{-1} x_0}\) for all domain cycles \(c\). We claim that the following conditions are equivalent:

1. \(\xi_{\sigma_c} = id_{\gamma^{-1} x_0}\) for all domain cycles \(c = (x_i)_{i \in \{0,1,\ldots,2k+1\}}\).
2. \(\xi_\sigma = \xi_{\sigma'}\) for all alternating sequences \(\sigma = (y_i)_{i \in \{0,1,\ldots,m\}}\) and \(\sigma' = (z_i)_{i \in \{0,1,\ldots,n\}}\) with \(y_0 = z_0\) and \(y_m = z_n\) (Independence of representative on paths from \(y_0\) to \(y_m\)).

Assume for the moment that this is true, then 2 is true because 1 is the assumption of the proposition. We can then use this independence of representative to uniquely construct a coherence witness, i.e. a family \((\xi_{e,e'})_{(e,e') \in \ker(s)}\) where \(s = \bar{a} \circ r = r \circ a\) of bijections which satisfies neutrality and associativity from Proposition 12 and for which (12) is valid: Clearly, \((x, x') \in \ker(s)\) iff there exists an alternating sequence \(\sigma = (y_i)_{i \in \{0,1,\ldots,m\}}\) with \(x = y_0\) and \(x' = y_m\) such that

\[
\xi_{x,x'} := \xi_\sigma
\]

does not depend on the choice of \(\sigma\). Neutrality follows from (21) for sequences of length 0, (12) is ensured by sequences of length 1.

To show associativity we define the composition of two alternating sequences by

\[
\sigma' \circ \sigma := (y_0,\ldots,y_m = z_0,\ldots,z_n)
\]

if \(mn = 0\) or \(m,n \geq 1\) and \((y_{m-1},y_m) \in \ker(p), (z_0,z_1) \in \ker(-p)\)

\[
\sigma' \circ \sigma := (y_0,\ldots,y_{m-1},z_1,\ldots,z_n)
\]

if \(m,n \geq 1\) and \((y_{m-1},y_m) \in \ker(p), (z_0,z_1) \in \ker(p)\)

Again by the independence of representative we obtain for each pair \((x,x'), (x',x'') \in \ker(s)\) (with representing alternating sequences \(\sigma, \sigma'\)) \(\xi_{\sigma'} \circ \xi_{\sigma} = \xi_{\sigma' \circ \sigma}\), hence associativity.

It remains to prove the equivalence \(1 \iff 2\). It is easy to show that one can restrict oneself to proper alternating sequences. Then \(2 \Rightarrow 1\) because 1 is a special case of 2 with \(m = 2k+2\) for \(k \in \mathbb{N}\), \(p = a\), and \(n = 0\). Thus, it remains to show \(1 \Rightarrow 2\).

Note first that the equation \(\xi_{e,e'} = (\xi_{e',e})^{-1}\) (cf. Proposition 12) carries over to alternating sequences: If \(\sigma = (y_0,\ldots,y_m)\), then \(\sigma^{-} := (y_m,\ldots,y_0)\) is an alternating sequence with

\[
\xi_{\sigma} = (\xi_{\sigma^{-}})^{-1}.
\]

(22)

For the proof of \(1 \Rightarrow 2\), we use induction over \(n\). For \(n = 0\) we have \(y_0 = z_0 = y_m\) and \(\xi_{\sigma'} = id_{\gamma^{-1} y_0}\). For \(m = 0\) and \(m = 1\) we have \(\xi_{\sigma} = id_{\gamma^{-1} y_0} = id_{\gamma^{-1} y_0} = id_{\gamma^{-1} y_0} = \xi_{\sigma'}\). Thus there remain two major cases.
1. \( m = 2k + 2 \) for some \( k \in \mathbb{N} \): Then either \( \sigma \) represents a domain cycle (if \( p = a \)) or the reverse cycle \( \sigma^{-} \) is a domain cycle (if \( p = r \)). Both situations yield \( \xi_\sigma = id_{\gamma^{-1}y_0} = \xi_\sigma' \) (cf. (22)).

2. \( m = 2k + 3 \) for some \( k \in \mathbb{N} \): If \( p = a \) we know that \( y_{m-1} \neq y_1 \), since \( \sigma \) is proper, thus the alternating sequence \( \bar{\sigma} = (y_{m-1}, y_1, \ldots, y_{m-1}) \) represents a domain cycle which is connected to \( \sigma \) via \( \xi_\sigma = \xi_{y_{m-1},y_m} \circ \xi_{\bar{\sigma}} \circ \xi_{y_0,y_{m-1}}^{-1} \) (using associativity for \( (y_0 = y_m, y_{m-1}), (y_{m-1}, y_1) \in ker(a) \)). By assumption

\[
\xi_\sigma = \xi_{y_{m-1},y_m} \circ \xi_{\bar{\sigma}} \circ \xi_{y_0,y_{m-1}}^{-1} = \xi_{y_{m-1},y_m} = id_{\gamma^{-1}y_0} = \xi_\sigma'
\]

because \( y_0 = y_m \). If \( p = r \) the same argument can be carried out with \( \bar{\sigma} = (y_1, \ldots, y_{m-1}, y_1) \).

Now we show the induction step to \( n \geq 1 \) under the hypothesis that the assertion is true for all pairs \((m, n')\) with \( n' < n \). Again there are several cases with possible subcases:

1. \( z_1 = y_0 \): This means \( z_1 = y_0 = z_0 \) and thus \( \xi_\sigma' = \xi_{\sigma_1'} \) for the sequence \( \sigma_1' = (z_1, \ldots, z_n) \). \( \xi_\sigma = \xi_{\sigma_1'} \), however, holds by induction hypothesis.

2. \( z_1 = y_k \) for some \( 1 \leq k \leq m \): By induction hypothesis we have \( \xi_{\sigma_1} = \xi_{\sigma_1'} \) and \( \xi_{\sigma_2} = \xi_{\sigma_2'} \) for the subsequences \( \sigma_1 = (y_0, \ldots, y_k), \sigma_2 = (y_k, \ldots, y_m), \sigma_1' = (z_0, z_1), \sigma_2' = (z_1, \ldots, z_n) \) thus we also obtain \( \xi_{\sigma_1} = \xi_{\sigma_2} \circ \xi_{\sigma_1} = \xi_{\sigma_2'} \circ \xi_{\sigma_1} = \xi_{\sigma_1'} \).

3. \( z_1 \neq y_k \) for all \( 0 \leq k \leq m \):

   (a) \((y_0, y_1) \in ker(p), (z_0, z_1) \in ker(p)\): Then \( \sigma_1 = (z_1, y_1, \ldots, y_m) \) is a proper alternating sequence. By induction hypothesis we have \( \xi_{\sigma_1} = \xi_{\sigma_1'} \) for the alternating sequence \( \sigma_1' = (z_1, \ldots, z_n) \), thus \( \xi_{\sigma_1} = \xi_{\sigma_1'} \circ \xi_{\sigma_1} = \xi_{\sigma_1'} \circ \xi_{\sigma_1} = \xi_{\sigma_1} \).

   (b) \((y_0, y_1) \in ker(p), (z_0, z_1) \in ker(-p)\): Then \( \sigma_1 = (z_1, z_0, y_1, \ldots, y_m) \) is a proper alternating sequence. By induction hypothesis we have \( \xi_{\sigma_1} = \xi_{\sigma_1'} \) for the alternating sequence \( \sigma_1' = (z_1, \ldots, z_n) \), thus we obtain, finally, \( \xi_{\sigma} = \xi_{\sigma_1} \circ \xi_{\sigma_1}^{-1} = \xi_{\sigma_1} \circ \xi_{\sigma_1}^{-1} = \xi_{\sigma} \).

\[\square\]

References


Satisfaction, Restriction and Amalgamation of Constraints in the Framework of $\mathcal{M}$-Adhesive Categories

Hanna Schölzel$^1$, Hartmut Ehrig$^1$, Maria Maximova$^1$, Karsten Gabriel$^1$, and Frank Hermann$^{1,2}$

1) Institut für Softwaretechnik und Theoretische Informatik, Technische Universität Berlin, Germany
2) University of Luxembourg, Interdisciplinary Centre for Security, Reliability and Trust

[hannas, ehrig, mascham, kgabriel, frank]@cs.tu-berlin.de

Application conditions for rules and constraints for graphs are well-known in the theory of graph transformation and have been extended already to $\mathcal{M}$-adhesive transformation systems. According to the literature we distinguish between two kinds of satisfaction for constraints, called general and initial satisfaction of constraints, where initial satisfaction is defined for constraints over an initial object of the base category. Unfortunately, the standard definition of general satisfaction is not compatible with negation in contrast to initial satisfaction.

Based on the well-known restriction of objects along type morphisms, we study in this paper restriction and amalgamation of application conditions and constraints together with their solutions. In our main result, we show compatibility of initial satisfaction for positive constraints with restriction and amalgamation, while general satisfaction fails in general.

Our main result is based on the compatibility of composition via pushouts with restriction, which is ensured by the horizontal van Kampen property in addition to the vertical one that is generally satisfied in $\mathcal{M}$-adhesive categories.

1 Introduction

The framework of $\mathcal{M}$-adhesive categories has been introduced recently [8,4] as a generalization of different kinds of high level replacement systems based on the double pushout (DPO) approach [6]. Prominent examples that fit into the framework of $\mathcal{M}$-adhesive categories are (typed attributed) graphs [6,19] and (high-level) Petri nets [2,10]. In the context of domain specific languages and model transformations based on graph transformation, graph conditions (constraints) are already used extensively for the specification of model constraints and the specification of application conditions of transformation rules. Graph conditions can be nested, may contain Boolean expressions [13,14] and are expressively equivalent to first-order formulas on graphs [4] as shown in [14,20]. We generally use the term “nested condition” whenever we refer to the most general case.

Restriction is a general concept for the definition of views of domain languages and is used for reducing the complexity of a model and for increasing the focus to relevant model element types. A major research challenge in this field is to provide general results that allow for reasoning on properties of the full model (system) by analyzing restricted properties on the views (restrictions) of the model only. Technically, a restriction of a model is given as a pullback along type morphisms. While this construction can be extended directly to restrictions of nested conditions, the satisfaction of the restricted nested conditions is not generally guaranteed for the restricted models, but—as we show in this paper—can be ensured under some sufficient conditions.

According to the literature [14,6], we distinguish between two kinds of satisfaction for nested conditions, called general and initial satisfaction, where initial satisfaction is defined for nested conditions over
an initial object of the base category. Intuitively, general satisfaction requires that a property holds for all occurrences of a premise pattern, while initial satisfaction requires this property for at least one occurrence. Unfortunately, the standard definition of general satisfaction is not compatible with the Boolean operators for negation and disjunction, but initial satisfaction is compatible with all Boolean operators (see App. A in [21]). In order to show, in addition, compatibility of initial satisfaction with restriction, we introduce the concept of amalgamation for typed objects, where objects can be amalgamated along their overlapping according to the given type restrictions.

As the main technical result, we show that solutions for nested conditions can be composed and decomposed along an amalgamation of them (Thm. 4.10), if the nested conditions are positive, i.e., they contain neither a negation nor a “for all” expression (universal quantification). Based on this property, we show in our main result (Thm. 5.1), that initial satisfaction of positive nested conditions is compatible with amalgamation based on restrictions that agree on their overlappings. Note in particular that this result does not hold for general satisfaction which we illustrate by a concrete counterexample.

The structure of the paper is as follows. Section 2 reviews the general framework of \( \mathcal{M} \)-adhesive categories and main concepts for nested conditions and their satisfaction. Thereafter, Sec. 3 presents the restriction of objects and nested conditions along type object morphisms. Section 4 contains the constructions and results concerning the amalgamation of objects and nested conditions and in Sec. 5, we present our main result showing the compatibility of initial satisfaction with amalgamation and restriction. Related work is discussed in Sec. 6. Section 7 concludes the paper and discusses aspects of future work. Appendix A contains the proofs that are not contained in the main part. Additionally, App. A in [21] provides formal details concerning the transformation between both satisfaction relations and, moreover, their compatibility resp. incompatibility with Boolean operators.

2 General Framework and Concepts

In this section we recall some basic well-known concepts and notions and introduce some new notions that we are using in our approach. Our considerations are based on the framework of \( \mathcal{M} \)-adhesive categories. An \( \mathcal{M} \)-adhesive category \([8]\) consists of a category \( \mathcal{C} \) together with a class \( \mathcal{M} \) of monomorphisms as defined in Def. 2.1 below. The concept of \( \mathcal{M} \)-adhesive categories generalizes that of adhesive [17], adhesive HLR [9], and weak adhesive HLR categories [6].

**Definition 2.1 (\( \mathcal{M} \)-Adhesive Category).** An \( \mathcal{M} \)-adhesive category \((\mathcal{C}, \mathcal{M})\) is a category \( \mathcal{C} \) together with a class \( \mathcal{M} \) of monomorphisms satisfying:

- the class \( \mathcal{M} \) is closed under isomorphisms, composition and decomposition,
- \( \mathcal{C} \) has pushouts and pullbacks along \( \mathcal{M} \)-morphisms,
- \( \mathcal{M} \)-morphisms are closed under pushouts and pullbacks, and
- it holds the vertical van Kampen (short VK) property. This means that pushouts along \( \mathcal{M} \)-morphisms are \( \mathcal{M} \)-VK squares, i.e., pushout (1) with \( m \in \mathcal{M} \) is an \( \mathcal{M} \)-VK square, if for all commutative cubes (2) with (1) in the bottom, all vertical morphisms \( a, b, c, d \in \mathcal{M} \) and pullbacks in the back faces we have that the top face is a pushout if and only if the front faces are pullbacks.
Remark 2.2. In Sec. 3, Sec. 4 and Sec. 5 we will also need the horizontal VK property, where the VK property is only required for commutative cubes with all horizontal morphisms in $M$ (see [8]), to show the compatibility of object composition and the corresponding restrictions. Note moreover, that an $M$-adhesive category which also satisfies the horizontal VK property is a weak adhesive HLR category [6].

A set of transformation rules over an $M$-adhesive category according to the DPO approach constitutes an $M$-adhesive transformation system [8]. For various examples (graphs, Petri nets, etc.) see [6].

In Sec. 3, Sec. 4 and Sec. 5 we are considering $M$-adhesive categories with effective pushouts. According to [18], the formal definition is as follows.

Definition 2.3 (Effective Pushout). Given $M$-morphisms $a : B \rightarrow X$, $b : C \rightarrow X$ in an $M$-adhesive category $(C,M)$ and let $(A,p_1,p_2)$ be obtained by the pullback of $a$ and $b$. Then pushout (1) of $p_1$ and $p_2$ is called effective, if the unique morphism $u : D \rightarrow X$ induced by pushout (1) is an $M$-morphism.

Nested conditions in this paper are defined as application conditions for rules in [13]. Depending on the context in which a nested condition occurs, we use the terms application condition [13] and constraint [6], respectively. Furthermore, we define positive nested conditions to be used in Sec. 3, Sec. 4, and Sec. 5 for our main results.

Definition 2.4 (Nested Condition). A nested condition $acP$ over an object $P$ is inductively defined as follows:

- true is a nested condition over $P$.
- For every morphism $a : P \rightarrow C$ and nested condition $acC$ over $C$, $\exists (a,acC)$ is a nested condition over $P$.
- A nested condition can also be a Boolean formula over nested conditions. This means that also $\neg acP$, $\bigwedge_{i \in I} acP_i$, and $\bigvee_{i \in I} acP_i$ are nested conditions over $P$ for nested conditions $acP_i, acP_i; (i \in I)$ over $P$ for some index set $I$.

Furthermore, we distinguish the following concepts:

- A nested condition is called application condition in the context of rules and match morphisms.
- A nested condition is called constraint in the context of properties of objects.
- A positive nested condition is built up only by nested conditions of the form true, $\exists (a,ac)$, $\bigwedge_{i \in I} acP_i$ and $\bigvee_{i \in I} acP_i$, where $I \neq \emptyset$.

An example for a nested condition and its meaning is given below.
Example 2.5 (Nested Condition). Given the nested condition $ac_P$ from Fig. 2, where all morphisms are inclusions. Condition $ac_P$ means that the source of every $b$-edge has a $b$-self-loop and must be followed by some $c$-edge such that subsequently, there is a path in the reverse direction visiting the source and target of the first $b$-edge with precisely one $c$-edge and one $b$-edge in an arbitrary order. We denote this nested condition by $ac_P = \exists (a_1, true) \land \exists (a_2, \exists (a_3, true) \lor \exists (a_4, true))$.

We are now defining inductively whether a morphism satisfies a nested condition (see [6]).

Definition 2.6 (Satisfaction of Nested Condition). Given a nested condition $ac_P$ over $P$, a morphism $p : P \rightarrow G$ satisfies $ac_P$ (see Fig. 1(a)), written $p \vDash ac_P$, if:

- $ac_P = true$, or
- $ac_P = \exists (a, ac_C)$ with $a : P \rightarrow C$ and there exists a morphism $q : C \rightarrow G \in \mathcal{M}$ such that $q \circ a = p$ and $q \vDash ac_C$, or
- $ac_P = \neg ac'_P$ and $p \not\vDash ac'_P$, or
- $ac_P = \bigwedge_{i \in I} ac_{P_i}$ and for all $i \in I$ holds $p \vDash ac_{P_i}$, or
- $ac_P = \bigvee_{i \in I} ac_{P_i}$ and for some $i \in I$ holds $p \vDash ac_{P_i}$.

In the following we distinguish two kinds of satisfaction relations for constraints: General [6] and initial satisfaction [14]. Initial satisfaction is defined for constraints over an initial object of the base category while general satisfaction is considered for constraints over arbitrary objects. Intuitively, while general satisfaction requires that a constraint $ac_P$ is satisfied by every $\mathcal{M}$-morphism $p : P \rightarrow G$, initial satisfaction requires just the existence of an $\mathcal{M}$-morphism $p : P \rightarrow G$ which satisfies $ac_P$.

Definition 2.7 (General Satisfaction of Constraints). Given a constraint $ac_P$ over $P$. An object $G$ generally satisfies $ac_P$, written $G \vDash ac_P$, if $\forall p : P \rightarrow G \in \mathcal{M}$. $p \vDash ac_P$ (see Fig. 1(a)).

Definition 2.8 (Initial Satisfaction of Constraints). Given a constraint $ac_I$ over an initial object $I$. An object $G$ initially satisfies $ac_I$, written $G \vDash ac_I$, if $i_G \vDash ac_I$ for the initial morphism $i_G : I \rightarrow G$.

Note, that for $ac_I = \exists (i_P, ac_P)$ we have

- $G \vDash ac_I \iff \exists p : P \rightarrow G \in \mathcal{M}$. $p \vDash ac_P$ (see Fig. 1(b)).

This means that the general satisfaction corresponds more to the universal satisfaction of constraints while the initial satisfaction corresponds more to the existential satisfaction.

For positive nested conditions, we define solutions for the satisfaction problem. A solution $Q$ (a tree of morphisms) determines which morphisms are used to fulfill the satisfaction condition.

Definition 2.9 (Solution for Satisfaction of Positive Nested Conditions). Given a positive nested condition $ac_P$ over $P$ and a morphism $p : P \rightarrow G$. Then $Q$ is a solution for $p \vDash ac_P$ if:

- $ac_P = true$ and $Q = \emptyset$, or

![Figure 1: Satisfaction of nested conditions](image)
• \( ac_P = \exists (a, ac_C) \) with \( a : P \to C \) and \( Q = (q, Q_C) \) with \( \mathcal{M}\)-morphism \( q : C \to G \) such that \( q \circ a = p \) and \( Q_C \) is a solution for \( q \models ac_C \) (see Fig. 1(a)), or

• \( ac_P = \wedge_{i \in \mathcal{I}} ac_{P_i} \) and \( Q = (Q_i)_{i \in \mathcal{I}} \) such that \( Q_i \) is a solution for \( p \models ac_{P_i} \) for all \( i \in \mathcal{I} \), or

• \( ac_P = \vee_{i \in \mathcal{I}} ac_{P_i} \) and \( Q = (Q_i)_{i \in \mathcal{I}} \) such that there is \( j \in \mathcal{I} \) with solution \( Q_j \) for \( p \models ac_{P_j} \) and for all \( k \in \mathcal{I} \) with \( k \neq j \) it holds that \( Q_k = \emptyset \).

The following example demonstrates the general and initial satisfaction of constraints and gives their corresponding solutions.

**Example 2.10 (Satisfaction and Solution of Constraints).**

1. **General Satisfaction**
   Consider the graph \( G_A \) from Fig. 2 below and the constraint \( ac_P \) from Ex. 2.5. There are two possible \( \mathcal{M}\)-morphisms \( p_1, p_2 : P \to G_A \), where \( p_1 \) is an inclusion and \( p_2 \) maps \( b_1 \) to \( b_2 \) with the corresponding node mapping. For both matches \( p_1 \) and \( p_2 \), there is a \( c \)-edge outgoing from the image of node 1, a \( c \)-edge outgoing from the image of node 2, as well as the corresponding images for edges \( b_2 \) and \( c_2 \) in \( C_3 \). Thus, \( G_A \) generally satisfies \( ac_P \).

   A corresponding solution for \( p_1 \models ac_P \) is given by \( Q_{gen} = (Q_i)_{i \in \{1, 2\}} \) with \( Q_1 = (q_1, \emptyset) \) and \( Q_2 = (q_2, (Q_j)_{j \in \{3, 4\}}) \), where \( Q_3 = (q_3, \emptyset) \). \( Q_4 = \emptyset \) and \( q_1 : C_i \to G_A \) for \( i = 1, 2, 3 \) are inclusions.

![Figure 2: General and initial satisfaction of constraints](image)

2. **Initial Satisfaction**
   Let \( ac_I = \exists (i_P, ac_P) \) with \( i_P \) as depicted in Fig. 2 and \( ac_P \) from Ex. 2.5. The graph \( G_A \) initially satisfies \( ac_I \) since there is \( p_1 : P \to G_A \in \mathcal{M} \) satisfying \( ac_P \) as mentioned before.

   A corresponding solution for \( i_G \models ac_I \) is given by \( Q_{init} = (p_1, Q_{gen}) \) with \( Q_{gen} \) from the example for general satisfaction.

**Remark 2.11.** A nested condition is called typed over a given type object, if all nested conditions in every of its nesting levels are also typed over the same type object. Furthermore, matches and corresponding solutions are required to be compatible with this type of object as well.

### 3 Restriction Along Type Morphisms

In this section, we present the restriction of objects, morphisms, positive nested conditions and their solutions along type morphisms which are the basis for the amalgamation of nested conditions in Sec. 4.
General Assumption. In this and the following sections, we consider an $\mathcal{M}$-adhesive category $(\mathcal{C}, \mathcal{M})$ satisfying the horizontal VK property (see [Rem. 2.2]) and has effective pushouts (see [Def. 2.3]).

**Definition 3.1 (Restriction along Type Morphism).** Given an object $G_A$ typed over $TG_A$ by $t_{G_A}: G_A \rightarrow TG_A$ and $t: TG_B \rightarrow TG_A \in \mathcal{M}$, then $TG_B$ is called restriction of $TG_A$, $G_B$ is a restriction of $G_A$, and $t_{G_B}$ is a restriction of $t_{G_A}$, if (1) is a pullback. Given $a: G_A' \rightarrow G_A$, then $b$ is a restriction of $a$ along type morphism $t$, written $b = \text{Restr}_t(a)$, if (2) is a pullback.

For positive nested conditions, we can define the restriction recursively as restriction of their components.

**Definition 3.2 (Restriction of Positive Nested Conditions).** Given a positive nested condition $ac_{P_A}$ typed over $TG_A$ and let $TG_B$ be a restriction of it with $t: TG_B \rightarrow TG_A \in \mathcal{M}$. Then we define the restriction $ac_{P_B} = \text{Restr}_t(ac_{P_A})$ over the restriction $P_B$ of $P_A$ as follows:

- The restriction of true is true,
- the restriction of $\exists (a, ac_{C_A})$ is given by restriction of $a$ and $ac_{C_A}$, i.e., $ac_{P_B} = \exists (\text{Restr}_t(a), \text{Restr}_t(ac_{C_A}))$, and
- the restriction of a Boolean formula is given by the restrictions of its components, i.e.,
  
  $\text{Restr}_t(\neg ac_{P_A}) = \neg \text{Restr}_t(ac_{P_A})$, \quad $\text{Restr}_t(\bigwedge_{i \in \mathcal{I}} ac_{P_A,i}) = \bigwedge_{i \in \mathcal{I}} \text{Restr}_t(ac_{P_A,i})$, and $\text{Restr}_t(\bigvee_{i \in \mathcal{I}} ac_{P_A,i}) = \bigvee_{i \in \mathcal{I}} \text{Restr}_t(ac_{P_A,i})$.

Now we extend the restriction construction to solutions of positive nested conditions and show in [Fact 3.4] that a restriction of a solution is also a solution for the corresponding restricted constraint.

**Definition 3.3 (Restriction of Solutions for Positive Nested Conditions).** Given a positive nested condition $ac_{P_A}$ typed over $TG_A$ together with a restriction $ac_{P_B}$ along $t: TG_B \rightarrow TG_A$. For a morphism $p_A: P_A \rightarrow G$ and a solution $Q_A$ for $p_A \vdash ac_{P_A}$, the restriction $Q_B$ of $Q_A$ along $t$, written $Q_B = \text{Restr}_t(Q_A)$, is defined inductively as follows:

- If $Q_A$ is empty then also $Q_B$ is empty,
- if $ac_{P_A} = \exists (a: P_A \rightarrow C_A, ac_{C_A})$ and $Q_A = (q_A, Q_{CA})$, then $Q_B = (q_B, Q_{CB})$ such that $Q_B$ and $Q_{CB}$ are restrictions of $q_A$ respectively $Q_{CA}$, and
- if $ac_{P_A} = \bigwedge_{i \in \mathcal{I}} ac_{P_A,i}$ or $ac_{P_A} = \bigvee_{i \in \mathcal{I}} ac_{P_A,i}$ and $Q_A = (Q_{A,i})_{i \in \mathcal{I}}$, then $Q_B = (Q_{B,i})_{i \in \mathcal{I}}$ such that $Q_{B,i}$ is a restriction of $Q_{A,i}$ for all $i \in \mathcal{I}$.

**Fact 3.4 (Restriction of Solutions for Positive Nested Conditions).** Given a positive nested condition $ac_{P_A}$ and a match $p_A: P_A \rightarrow G_A$ over $TG_A$ with restrictions $ac_{P_B} = \text{Restr}_t(ac_{P_A})$, $p_B = \text{Restr}_t(p_A)$ along $t: TG_B \rightarrow TG_A$. Then for a solution $Q_A$ of $p_A \vdash ac_{P_A}$, there is a solution $Q_B = \text{Restr}_t(Q_A)$ for $p_B \vdash ac_{P_B}$.
4 Amalgamation

The amalgamation of typed objects allows to combine objects of different types provided that they agree on a common subtype. This concept is already known in the context of different types of Petri net processes, such as open net processes [11] and algebraic high-level processes [7], which can be seen as special kinds of typed objects. In this section, we introduce a general definition for the amalgamation of typed objects. Moreover, we extend the concept to the amalgamation of positive nested conditions and their solutions.

As required for amalgamation, we discuss under which conditions morphisms can be composed via a span of restriction morphisms. Two morphisms $g_B$ and $g_C$ “agree” in a morphism $g_D$, if $g_D$ can be constructed as a common restriction and can be used as a composition interface for $g_B$ and $g_C$ as in [Def. 4.1]

**Definition 4.1** (Agreement and Amalgamation of Typed Objects). Given a span $TG_B \xrightarrow{tg_{DB}} TG_D \xleftarrow{tg_{DC}} TG_C$, with $tg_{DB}, tg_{DC} \in M$ and typed objects $G_B \xrightarrow{g_B} TG_B$, $G_C \xrightarrow{g_C} TG_C$ and $G_D \xrightarrow{g_D} TG_D$. We say $g_B, g_C$ agree in $g_D$, if $g_D$ is a restriction of $g_B$ and $g_C$, i.e., $\text{Restr}_{g_D}(g_B) = g_D = \text{Restr}_{g_D}(g_C)$.

Given pushout (1) below with all morphisms in $M$ and typed objects $g_B, g_C$ agreeing in $g_D$. A morphism $g_A : G_A \rightarrow TG_A$ is called amalgamation of $g_B$ and $g_C$ over $g_D$, written $g_A = g_B + g_D g_C$, if the outer square is a pushout and $g_B, g_C$ are restrictions of $g_A$.

![Diagram](https://via.placeholder.com/150)

Fact 4.2 is essentially based on the horizontal VK property.

**Fact 4.2** (Amalgamation of Typed Objects). Given pushout (1) with all morphisms in $M$ as in [Def. 4.1]

**Composition.** Given $g_B, g_C$ agreeing in $g_D$, then there exists a unique amalgamation $g_A = g_B + g_D g_C$.

**Decomposition.** Vice versa, given $g_A : G_A \rightarrow TG_A$, there are unique restrictions $g_B, g_C$, and $g_D$ of $g_A$ such that $g_A = g_B + g_D g_C$.

Here and in the following, uniqueness means uniqueness up to isomorphism.

**Proof.** Given $g_B, g_C$ agreeing in $g_D$, we have that the upper two trapezoids are pullbacks. Now we construct $G_A$ as pushout over $G_B$ and $G_C$ via $G_D$, such that the outer diamond is a pushout. This leads to a unique induced morphism $g_A : G_A \rightarrow TG_A$, such that the diagram commutes and via the horizontal VK property we get that the lower two trapezoids are pullbacks and therefore $g_A = g_B + g_D g_C$.

Vice versa, we can construct $G_B, G_C, G_D$ as restrictions such that the trapezoids become pullbacks, where $g_A : G_A \rightarrow TG_A$ and $TG_A, TG_B, TG_C, TG_D$ are given such that (1) is a pushout with $M$-morphisms only. Then the horizontal VK property implies that the outer diamond is a pushout and $g_A$ is unique because of the universal property and $g_A = g_B + g_D g_C$.

The uniqueness (up to isomorphism) of the amalgamated composition and decomposition constructions follows from uniqueness of pushouts and pullbacks up to isomorphism. □
Example 4.3 (Amalgamation of Typed Objects). Figure 3 shows a pushout of type graphs $T G_A$, $T G_B$, $T G_C$, and $T G_D$.

**Composition.** Consider the typed graphs $G_B$, $G_C$ and $G_D$ typed over $T G_B$, $T G_C$ and $T G_D$, respectively. Graph $G_D$, containing the same nodes as $G_B$ and $G_C$ and no edges, is the common restriction of $G_B$ and $G_C$. So, the type morphisms $g_B$ and $g_C$ agree in $g_D$, which by Fact 4.2 means that there is an amalgamation $g_A = g_B + g_D g_C$. It can be obtained by computing the pushout of $G_B$ and $G_C$ over $G_D$, leading to the graph $G_A$ that contains the $b$-edges of $G_B$ as well as the $c$-edges of $G_C$. The type morphism $g_A$ is induced by the universal property of pushouts, mapping all edges in the same way as $g_B$ and $g_C$.

**Decomposition.** Vice versa, consider the graph $G_A$ typed over $T G_A$. We can restrict $G_A$ to the type graphs $T G_B$ and $T G_C$, leading to typed graphs $G_B$ and $G_C$, containing only the $b$- respectively $c$-edges of $G_A$. Restricting the graphs $G_B$ and $G_C$ to type graph $T G_D$, we get in both cases the graph $G_D$ that contains no edges, and we have that $g_A = g_B + g_D g_C$.

We already defined the restriction of positive nested conditions (Def. 3.2) and their solutions (Def. 3.3). Now we want to consider the case that we have two conditions, which have a common restriction and can be amalgamated.

**Definition 4.4 (Agreement and Amalgamation of Positive Nested Conditions).** Given a pushout (1) below with all morphisms in $\mathcal{M}$. Two positive nested conditions $ac_{P_B}$ typed over $T G_B$ and $ac_{P_C}$ typed over $T G_C$ agree in $ac_{P_D}$ typed over $T G_D$ if $ac_{P_D}$ is a restriction of $ac_{P_B}$ and $ac_{P_C}$.

Given $ac_{P_B}$ and $ac_{P_C}$ agreeing in $ac_{P_D}$ then a positive nested condition $ac_{P_A}$ typed over $T G_A$ is called amalgamation of $ac_{P_B}$ and $ac_{P_C}$ over $ac_{P_D}$, written $ac_{P_A} = ac_{P_B} + ac_{P_D} ac_{P_C}$, if $ac_{P_B}$ and $ac_{P_C}$ are restrictions of $ac_{P_A}$ and $tPA = tPB + tPB tPC$. In particular, we have $true_A = true_B + true_D true_C$, short true = true + true true.
In the following Fact 4.5, we give a construction for the amalgamation of positive nested conditions and in Thm. 4.10 for the corresponding solutions.

**Fact 4.5 (Amalgamation of Positive Nested Conditions).** Given a pushout (1) as in Def. 4.4 with all morphisms in $\mathcal{M}$.

**Composition.** If there are positive nested conditions $ac_{P_A}$ and $ac_{P_C}$ typed over $T_G_B$ and $T_G_C$, respectively, agreeing in $ac_{P_B}$ typed over $T_G_D$, then there exists a unique positive nested condition $ac_{P_A}$ typed over $T_G_A$ such that $ac_{P_A} = ac_{P_B} + ac_{P_D} ac_{P_C}$.

**Decomposition.** Vice versa, given a positive nested condition $ac_{P_A}$ typed over $T_G_A$, there are unique restrictions $ac_{P_B}$, $ac_{P_C}$ and $ac_{P_D}$ of $ac_{P_A}$ such that $ac_{P_A} = ac_{P_B} + ac_{P_D} ac_{P_C}$.

The amalgamated composition and decomposition constructions are unique up to isomorphism.

**Remark 4.6.** Given an amalgamation $ac_{P_A} = ac_{P_B} + ac_{P_D} ac_{P_C}$ of positive nested conditions, we can conclude from the proof of Fact 4.5 (see App. A) that we also have corresponding amalgamations in each level of nesting.

**Example 4.7 (Amalgamation of Positive Nested Conditions).** Figure 4 shows a pushout of typed graphs $T_G_A$, $T_G_B$, $T_G_C$ and $T_G_D$, and four positive nested conditions $ac_{P_A}$, $ac_{P_B}$, $ac_{P_C}$ and $ac_{P_D}$ typed over $T_G_A$, $T_G_B$, $T_G_C$ and $T_G_D$, respectively. For simplicity, the figure contains only the type morphisms of the $P$s, but there are also corresponding type morphisms for the $C$s, mapping all $b$-edges to $b$ and all $c$-edges to $c$. There is $ac_{P_B} = \bigvee_{i\in\{1,2\}} ac_{C_{i,A}}$ with $ac_{C_{i,A}} = \exists (a_{i,A}, \text{true})$ for $i = 1,2$, and $ac_{P_B}$, $ac_{P_C}$ and $ac_{P_D}$ have a similar structure.

**Composition.** We have that $t_{P_B}$ is a common restriction of $t_{P_B}$ and $t_{P_C}$, and also that $a_{i,D}$ is a common restriction of $a_{i,B}$ and $a_{i,C}$ for $i = 1,2$. Thus, $ac_{P_B}$ is a common restriction of $ac_{P_B}$ and $ac_{P_D}$, which means that $ac_{P_B}$ and $ac_{P_D}$ agree in $ac_{P_B}$. So by Fact 4.5 there exists an amalgamation $ac_{P_A} = ac_{P_B} + ac_{P_D} ac_{P_C}$, and according to Rem. 4.6 it can be obtained as amalgamation of its components.
This means that we have an amalgamation $t_{P_A} = t_{P_B} \sqcup t_{P_C}$ with pushout of the $P$s as shown in Fig. 4 as well as amalgamations of the corresponding type morphisms of the $C$s, leading to the pushouts depicted in Fig. 4 by dotted arrows for the $C_1$s and by dashed arrows for the $C_2$s. The morphisms $a_{1A}$ and $a_{2A}$ are obtained by the universal property of pushouts.

**Decomposition.** The other way around, considering the condition $ac_{P_B}$, we can construct the restrictions $ac_{P_B}$ and $ac_{P_C}$ by deleting the $c$- respectively $b$-edges. Then, restricting $ac_{P_B}$ and $ac_{P_C}$ to $TG_D$ by deleting all remaining edges, we obtain the same condition $ac_{P_D}$ such that $ac_{P_A} = ac_{P_B} + ac_{P_C}$.

In order to answer the question, under which conditions such amalgamated positive nested conditions are satisfied, we need to define an amalgamation of their solutions. Afterwards, we show in the proof of Thm. 4.10 that a composition of two solutions via an interface leads to a unique amalgamated solution and that a given solution for an amalgamated positive nested condition is the amalgamation of its unique restrictions.

**Definition 4.8** (Agreement and Amalgamation of Solutions for Positive Nested Conditions). Given pushout (1) below with all morphisms in $M$, an amalgamation of typed objects $g_A = g_B +_{gD} g_C$, and an amalgamation of positive nested conditions $ac_{P_A} = ac_{P_B} + ac_{P_D} ac_{P_C}$ with corresponding matches $p_A = p_B +_{pC} p_D$.

1. Two solutions $Q_B$ for $p_B \models ac_{P_B}$ and $Q_C$ for $p_C \models ac_{P_C}$ agree in a solution $Q_D$ for $p_D \models ac_{P_D}$, if $Q_D$ is a restriction of $Q_B$ and $Q_C$.
2. Given solutions $Q_B$ for $p_B \models ac_{P_B}$ and $Q_C$ for $p_C \models ac_{P_C}$ agreeing in a solution $Q_D$ for $p_D \models ac_{P_D}$, then a solution $Q_A$ for $p_A \models ac_{P_A}$ is called amalgamation of $Q_B$ and $Q_C$ over $Q_D$, written $Q_A = Q_B +_{QD} Q_C$, if $Q_B$ and $Q_C$ are restrictions of $Q_A$.

![Diagram](https://example.com/diagram.png)

**Remark 4.9.** Note that by assumption $g_A = g_B +_{gD} g_C$ in the definition above we already have a pushout over the $G$s, and by $ac_{P_A} = ac_{P_B} + ac_{P_D} ac_{P_C}$ we also have a pushout over the $P$s.

**Theorem 4.10** (Amalgamation of Solutions for Positive Nested Conditions). Given pushout (1) as in Def. 4.8 with all morphisms in $M$, an amalgamation of typed objects $g_A = g_B +_{gD} g_C$, and an amalgamation of positive nested conditions $ac_{P_A} = ac_{P_B} + ac_{P_D} ac_{P_C}$ with corresponding matches $p_A = p_B +_{pC} p_D$.

**Composition.** Given solutions $Q_B$ for $p_B \models ac_{P_B}$ and $Q_C$ for $p_C \models ac_{P_C}$ agreeing in a solution $Q_D$ for $p_D \models ac_{P_D}$, then there is a solution $Q_A$ for $p_A \models ac_{P_A}$ constructed as amalgamation $Q_A = Q_B +_{QD} Q_C$.

**Decomposition.** Given a solution $Q_A$ for $p_A \models ac_{P_A}$, then there are solutions $Q_B$, $Q_C$ and $Q_D$ for $p_B \models ac_{P_B}$, $p_C \models ac_{P_C}$ and $p_D \models ac_{P_D}$, respectively, which are constructed as restrictions $Q_B$, $Q_C$ and $Q_D$ of $Q_A$ such that $Q_A = Q_B +_{QD} Q_C$.

The amalgamated composition and decomposition constructions are unique up to isomorphism.

**Remark 4.11.** From the proof of Thm. 4.10 (see App. A) we can conclude that for a given amalgamation of solutions $Q_A = Q_B +_{QD} Q_C$, we also have corresponding amalgamations of its components.
5 Compatibility of Initial Satisfaction with Restriction and Amalgamation

In this section we present our main result showing compatibility of initial satisfaction with amalgamation (Thm. 5.1) and restriction (Cor. 5.2) which are based on the amalgamation of solutions for positive nested conditions (Thm. 4.10). This main result allows to conclude the satisfaction of a constraint for a composed object from the satisfaction of the corresponding restricted constraints for the component objects. It is valid for initial satisfaction, but not for general satisfaction.

**Theorem 5.1 (Compatibility of Initial Satisfaction with Amalgamation).** Given pushout (1) below with all morphisms in \( M \), an amalgamation of typed objects \( g_A = g_B + g_D g_C \), and an amalgamation of positive constraints \( ac_A = ac_B + ac_D ac_C \). Then we have:

**Decomposition.** Given a solution \( Q_A \) for \( G_A \models ac_A \), then there are solutions \( Q_B \) for \( G_B \models ac_B \), \( Q_C \) for \( G_C \models ac_C \) and \( Q_D \) for \( G_D \models ac_D \) such that \( Q_A = Q_B + Q_D q C \).

**Composition.** Vice versa, given solutions \( Q_B \) for \( G_B \models ac_B \) and \( Q_C \) for \( G_C \models ac_C \) agreeing in a solution \( Q_D \) for \( G_D \models ac_D \), then there exists a solution \( Q_A \) for \( G_A \models ac_A \) such that \( Q_A = Q_B + Q_D q C \).

*Proof.*

**Decomposition.** By Def. 2.8 a solution \( Q_A \) for \( G_A \models ac_A \) is also a solution for \( i_{G_A} \models ac_A \), where \( i_{G_A} \) is the unique morphism \( i_{G_A} : I \to G_A \). Moreover, due to amalgamation \( g_A = g_B + g_D g_C \) the inner trapezoids in the diagram above are pullbacks. So by closure of \( M \) under pullbacks we have that \( g_{BA}, g_{CA}, g_{DB}, g_{DC} \in M \) which means that they are monomorphisms. Therefore, the outer trapezoids become pullbacks by standard category theory, which means that \( i_{G_B} : I \to G_B \) is a restriction of \( i_{G_A} \), \( i_{G_C} : I \to G_C \) is a restriction of \( i_{G_A} \), and \( i_{G_D} : I \to G_D \) is a restriction of \( i_{G_B} \) as well as of \( i_{G_C} \).

Furthermore, the outer square in the diagram is a pushout, implying that we have an amalgamation \( i_{G_A} = i_{G_B} + i_{G_D} i_{G_C} \). Thus, using Thm. 4.10 we obtain solutions \( Q_B \) for \( i_{G_B} \models ac_B \), \( Q_C \) for \( i_{G_C} \models ac_C \) and \( Q_D \) for \( i_{G_D} \models ac_D \) such that \( Q_A = Q_B + Q_D q C \), and by Def. 2.8 \( Q_B, Q_C \) and \( Q_D \) are solutions for \( G_B \models ac_B \), \( G_C \models ac.C \) and \( G_D \models ac_D \), respectively.

**Composition.** Now, given solutions \( Q_B, Q_C \) and \( Q_D \) for \( G_B \models ac_B \), \( G_C \models ac_C \) and \( G_D \models ac_D \), respectively. Then by Def. 2.8 we have that \( Q_B, Q_C \) and \( Q_D \) are solutions for \( i_{G_B} \models ac_B \), \( i_{G_C} \models ac_C \) and \( i_{G_D} \models ac_D \), respectively. As shown in item 1, there is \( i_{G_A} = i_{G_B} + i_{G_D} i_{G_C} \) and therefore, since \( Q_B \) and \( Q_C \) agree...
in \( Q_D \), by Theorem 4.10 we obtain a solution \( Q_A \) for \( i_{G_A} \Vdash ac_A \) such that \( Q_A = Q_B + Q_D Q_C \). Finally, Definition 2.8 implies that \( Q_A \) is a solution for \( G_A \Vdash ac_A \).

**Corollary 5.2** (Compatibility of Initial Satisfaction with Restriction). Given type restriction \( t : TG_B \rightarrow TG_A \in \mathcal{M} \), object \( G_A \) typed over \( TG_A \) with restriction \( G_B \), and a positive constraint \( ac_A \) over initial object \( I \) typed over \( TG_A \) with restriction \( ac_B \). Then \( G_A \Vdash ac_A \) implies \( G_B \Vdash ac_B \). Moreover, if \( Q_A \) is a solution for \( G_A \Vdash ac_A \) then \( Q_B = Restr_{t}(Q_A) \) is a solution for \( G_B \Vdash ac_B \).

**Proof.** Consider the diagram in Theorem 5.1 with \( G_C = G_A, G_D = G_B, ac_C = ac_A \) and \( ac_D = ac_B \). Then by standard category theory we have that all rectangles in the diagram are pushouts and the trapezoids are pullbacks. Thus, we have \( s_A = g_B + g_D s_A \) and, analogously, \( ac_A = ac_B + ac_D ac_A \) with corresponding matches \( i_{G_A} = i_{G_B} + i_{G_D} i_{G_A} \). So, given a solution \( Q_A \) for \( G_A \Vdash ac_A \), by item 1 of Theorem 5.1 there is a solution \( Q_B \) for \( G_B \Vdash ac_B \) with \( Q_A = Q_B + Q_D Q_A \) such that by Definition 4.4, \( Q_B \) is a restriction of \( Q_A \).

**Example 5.3** (Compatibility of Initial Satisfaction with Amalgamation). Figure 5 shows the amalgamation of typed graphs \( g_A = g_B + g_D g_C \) from Example 4.7 and an amalgamation of positive nested structures \( ac_A = ac_B + ac_C ac_C \). Note that we have \( ac_A = \exists (i_{P_A}, ac_A) \) and \( ac_B, ac_C \) and \( ac_D \) with similar structure, where the amalgamation \( ac_P = ac_B + ac_D ac_P \) is presented in Example 4.7.

**Composition.** For \( G_B \Vdash ac_B \) we have the solution \( Q_B = (q_B, (Q_{1,B}, Q_{2,B})) \) with \( Q_{1,B} = (q_{1,B}, \emptyset) \) and \( Q_{2,B} = \emptyset \). Moreover, we have similar solutions \( Q_C \) for \( G_C \Vdash ac_C \) and \( Q_D \) for \( G_D \Vdash ac_D \). According to Remark 4.11, the amalgamation \( Q_A = Q_B + Q_D Q_C \) can be constructed by amalgamation of the components.

First, we explain in detail the amalgamation \( q_{1,A} = q_{1,B} + q_{1,D} q_{1,C} \). Note that the graphs \( G_A, G_B, G_C \) and \( G_D \) can be considered as type graphs such that e.g. \( G_{1,D} \) is typed over \( G_D \) by \( q_{1,D} \). So, since \( q_{1,D} \) is a common restriction of \( q_{1,B} \) and \( q_{1,C} \), we have that \( q_{1,B} \) and \( q_{1,C} \) agree in \( q_{1,D} \). This means that there is an amalgamation of typed objects \( q_{1,A} = q_{1,B} + q_{1,D} q_{1,C} \), where the inclusion \( q_{1,A} \) maps all nodes and edges in the same way as \( q_{1,B} \) and \( q_{1,C} \).

Moreover, for the empty solutions we have an empty solution as amalgamation, and thus, we have amalgamations of solutions \( Q_{1,A} = Q_{1,B} + Q_{1,D} Q_{1,C} = (q_{1,A}, \emptyset) \) and \( Q_{2,A} = Q_{2,B} + Q_{2,D} Q_{2,C} = \emptyset \). The amalgamation \( q_A = q_B + q_D q_C \) can be obtained analogously as described for \( q_{1,A} \), and hence, we have \( Q_A = Q_B + Q_D Q_C = (q_A, (Q_{1,A}, Q_{2,A})) \), which is a solution for \( G_A \Vdash ac_A \).

**Decomposition.** For \( G_A \Vdash ac_A \) we have a solution \( Q_A = (q_A, (Q_{1,A}, Q_{2,A})) \) with \( Q_{1,A} = (q_{1,A}, \emptyset) \) and \( Q_{2,A} = \emptyset \) where \( q_A \) and \( q_{1,A} \) are inclusions. The restrictions \( Q_B, Q_C \) and \( Q_D \) of \( Q_A \) are given by restrictions of the components. By computing the restrictions \( q_{1,B}, q_{1,C} \) and \( q_{1,D} \) of \( q_{1,A} \), and similar the restrictions of \( q_A \) and \( \emptyset \), we get as result again the solutions \( Q_B \) for \( G_B \Vdash ac_B \), \( Q_C \) for \( G_C \Vdash ac_C \), and \( Q_D \) for \( G_D \Vdash ac_D \) as described in the composition case above.

From Corollary 5.2, we know that initial satisfaction is compatible with restriction of typed objects and constraints. In contrast, general satisfaction and restriction are not compatible in general. As the following example illustrates, it is possible that a typed object generally satisfies a constraint while the same does not hold for their restrictions.
Example 5.4 (Restriction of General Satisfaction Fails in General). Figure 6 shows a restriction $G_B$ of the typed graph $G_A$ and a restriction $ac_{P_B}$ of constraint $ac_{P_A}$. There are two possible matches $p_{1,A}, p_{2,A} : P_A \rightarrow G_A \in \mathcal{M}$ where $p_{1,A}$ is an inclusion and $p_{2,A}$ maps $b_1$ to $b_2$ and $c_1$ to $c_2$. Since for each of the matches the graph $G_A$ contains the required edges in the inverse direction, both of the matches satisfy $ac_{P_A}$. For $p_{1,A}$ we have $q_{i,A}$ with $q_{i,A} \circ a_A = p_{i,A}$ for $i = 1, 2$. Thus, we have that $G_A \models ac_{P_A}$.

For the constraint $ac_{P_B}$ there is a match $p_B : P_B \rightarrow G_B \in \mathcal{M}$ mapping edge $b_1$ identically and node 3 to node 4. We have that $p_B \not\models ac_{P_B}$ because there is no edge from node 4 to node 2 in $G_B$, which means that $G_B \not\models ac_{P_B}$. This is due to the fact that there is no match $p_A : P_A \rightarrow G_A \in \mathcal{M}$ such that $p_B$ is the restriction of $p_A$.

6 Related Work

The framework of $\mathcal{M}$-adhesive categories [8] generalizes various kinds of categories for high level replacement systems, e.g. adhesive [17], quasi-adhesive [18], partial VK square adhesive [15], and weak-
adhesive categories [6]. Therefore, the results of this paper are applicable to all of them, where the category of typed attributed graphs is a prominent example.

The concepts of nested graph conditions [13] and first-order graph formulas [4] are shown to be expressively equivalent in [14] using the translation between first-order logic and predicates on edge-labeled graphs without parallel edges [20].

Multi-view modelling is an important concept in software engineering. Several approaches have been studied and used, e.g. focussing on aspect oriented techniques [12]. In this line, graph transformation (GT) approaches have been extended to support view concepts based on the integration of type graphs. For this purpose, the concept of restriction along type morphisms has been studied and used intensively [11, 5] including GT systems using the concept of inheritance and views [5, 16]. Instead of restriction of constraints considered in this paper, only more restrictive forward translations of view constraints have been studied in [5] for the case of atomic constraints with general satisfaction leading to a result similar to Thm. 5.1. The notions of initial and general satisfaction for nested conditions can be transformed one into the other [14], but this transformation uses the Boolean operator negation that is not present in positive constraints, for which, however, our main result on the compatibility of restriction and initial satisfaction holds. Moreover, we have shown by counterexample that general satisfaction is not compatible with restriction in general, even if only positive constraints are considered.

7 Conclusion

Nested application conditions for rules and constraints for graphs and more general models have been studied already in the framework of $\mathcal{M}$-adhesive transformation systems [6, 9]. The new contribution of this paper is to study compatibility of satisfaction with restriction and amalgamation. This is important for large typed systems respectively objects, which can be decomposed by restriction and composed by amalgamation. The main result in this paper shows that initial satisfaction of positive constraints is compatible with restriction and amalgamation (Thm. 5.1 and Cor. 5.2). The amalgamation construction is based on the horizontal van Kampen (VK) property, which is required in addition to the vertical VK property of $\mathcal{M}$-adhesive categories. To our best knowledge, this is the most interesting result for $\mathcal{M}$-adhesive transformation systems which is based on the horizontal VK property. Note that the main result is not valid for general satisfaction of positive constraints nor for initial satisfaction of general constraints. For future work, it is important to obtain weaker versions of the main result, which are valid for general satisfaction and constraints, respectively.

References


### A Remaining Proofs

In this appendix, we give the proofs for [Fact 3.4](#), [Fact 4.5](#) and [Thm. 4.10](#).

#### Fact 3.4 (Restriction of Solutions for Positive Nested Conditions)

Given a positive nested condition $ac_{P_A}$ and a match $p_A : P_A \rightarrow G_A$ over $TG_A$ with restrictions $ac_{P_B} = \text{Restr}_i(ac_{P_A})$, $p_B = \text{Restr}_i(p_A)$ along $t : TG_B \rightarrow TG_A$. Then for a solution $Q_A$ of $p_A \models ac_{P_A}$ there is a solution $Q_B = \text{Restr}_i(Q_A)$ for $p_B \models ac_{P_B}$.

**Proof.**

- For $ac_{P_A} = \text{true}$ the implication is trivial, because $Q_A$ is empty which means that also $Q_B$ is empty and thus a solution for $p_B \models ac_{P_B}$ is empty, because $ac_{P_B}$ is also true.

- For $ac_{P_A} = \exists (a, ac_{C_A})$ we have that $Q_A = (q_A, Q_{CA})$ such that $q_A : C_A \rightarrow G_A \in \mathcal{M}$ with $q_A \circ a = p_A$ and $Q_{CA}$ is a solution for $q_A \models ac_{C_A}$. Then by $q_B = \text{Restr}_i(q_A) : C_B \rightarrow G_B$, we have $q_B \in \mathcal{M}$ and we also have $t_G : G_B \rightarrow G_A \in \mathcal{M}$, because $t \in \mathcal{M}$ (see Fig. 7). So for $ac_{P_B} = \exists (b, ac_{C_B})$ we have

  $$ t_G \circ q_B \circ b = q_A \circ t_C \circ b = q_A \circ a \circ t_P = p_A \circ t_P = t_G \circ p_B, $$

  which by monomorphism $t_G$ implies $q_B \circ b = p_B$.

Moreover, the fact that $Q_{CA}$ is a solution for $q_A \models ac_{C_A}$ implies that $Q_{CB} = \text{Restr}_i(Q_{CA})$ is a solution for $q_B \models ac_{C_B}$ by induction hypothesis and hence the restriction $Q_B = (q_B, Q_{CB})$ of $Q_A$ is a solution for $p_B \models ac_{P_B}$.

- Now, for $ac_{P_A} = \bigwedge_{i \in \mathcal{I}} ac_{P_A,i}$ we have $ac_{P_B} = \bigwedge_{i \in \mathcal{I}} \text{Restr}_i(ac_{P_A,i})$. By the fact that $Q_A$ is a solution for $p_A \models ac_{P_A}$, we have that $Q_A = (Q_{A,i})_{i \in \mathcal{I}}$ such that $Q_{A,i}$ is a solution for $p_A \models ac_{A,i}$ for all $i \in \mathcal{I}$. Thus, by induction hypothesis, we have restrictions $Q_{B,i} = \text{Restr}_i(Q_{A,i})$ that are solutions for $p_B \models \text{Restr}_i(ac_{P_A,i})$ for all $i \in \mathcal{I}$. Hence, the restriction $Q_B = (Q_{B,i})_{i \in \mathcal{I}}$ of $Q_A$ is a solution for $p_B \models ac_{P_B}$.

- Finally, for $ac_{P_A} = \bigvee_{i \in \mathcal{I}} ac_{P_A,i}$ we have $ac_{P_B} = \bigvee_{i \in \mathcal{I}} \text{Restr}_i(ac_{P_A,i})$. By the fact that $Q_A$ is a solution for $p_A \models ac_{P_A}$, we have that $Q_A = (Q_{A,i})_{i \in \mathcal{I}}$ such that for one $j \in \mathcal{I}$ there is a solution $Q_{A,j}$ for $p_A \models ac_{A,j}$ and for all $k \neq j$ we have that $Q_{A,k} = \emptyset$. Thus, by induction hypothesis, the restriction $Q_{B,j}$ of $Q_{A,j}$ is a solution for $p_B \models \text{Restr}_i(ac_{P_A,i})$. Hence, we also have that the restriction $Q_B = (Q_{B,i})_{i \in \mathcal{I}}$ is a solution for $p_B \models ac_{P_B}$ with $Q_{B,k} = \emptyset$ for $k \neq j$.

\[\square\]

#### Fact 4.5 (Amalgamation of Positive Nested Conditions)

Given a pushout (1) as in [Def. 4.4](#) with all morphisms in $\mathcal{M}$.

![Figure 7: Restriction of solution $q_A$ for $p_A \models \exists (a, ac_{C_A})$](#)
Proof. The amalgamated composition and decomposition constructions are unique up to isomorphism. We perform an induction over the structure of $\text{Composition}$.

If there are positive nested conditions $ac_{P_B}$ and $ac_{P_C}$ typed over $T_G B$ and $T_G C$, respectively, agreeing in $ac_{P_B}$ typed over $T_G D$ then there exists a unique positive nested condition $ac_{P_A}$ typed over $T_G A$ such that $ac_{P_A} = ac_{P_B} + ac_{P_D} ac_{P_C}$.

**Decomposition.** Vice versa, given a positive nested condition $ac_{P_A}$ typed over $T_G A$, there are unique restrictions $ac_{P_B}$, $ac_{P_C}$ and $ac_{P_D}$ of $ac_{P_A}$ such that $ac_{P_A} = ac_{P_B} + ac_{P_D} ac_{P_C}$.

The amalgamated composition and decomposition constructions are unique up to isomorphism.

**Composition.** We perform an induction over the structure of $ac_{P_0}$:

- $ac_{P_D}$ is true.
  Then we also have $ac_{P_B} = true$ and $ac_{P_C} = true$, and the amalgamation $ac_{P_A}$ is trivially given by $ac_{P_A} = true$.
- $ac_{P_D} = \exists (d, ac_{C_D})$ with $d : P_D \rightarrow C_D$.
  The assumption that $ac_{P_B}$ and $ac_{P_C}$ agree in $ac_{P_B}$ means that $ac_{P_D}$ is a restriction of $ac_{P_B}$ and $ac_{P_C}$ and thus, by Def. 3.2 we have that $ac_{P_B} = \exists (b, ac_{C_B})$ with $b : P_B \rightarrow C_B$, $ac_{P_C} = \exists (c, ac_{C_C})$ with $c : P_C \rightarrow C_C$, $d$ is a restriction of $b$ and $c$, and $ac_{C_D}$ is a restriction of $ac_{C_B}$ and $ac_{C_C}$. This in turn means that $ac_{C_D}$ and $ac_{C_C}$ agree in $ac_{C_B}$ according to Def. 4.4. So, by induction hypothesis, we obtain an amalgamation $ac_{C_A} = ac_{C_B} + ac_{C_D} ac_{C_C}$, which implies that $ac_{C_A} = t_{GB} ac_{C_B} + t_{GDC} ac_{C_C}$, i.e., diagrams (2)-(5) below are pullbacks. By closure of $\mathcal{M}$ under pullbacks, we obtain from $t_{GBA}, t_{GCA} \in \mathcal{M}$ that also $c_{BA}, c_{CA} \in \mathcal{M}$.
  Moreover, the fact that $d$ is a restriction of $b$ and $c$ means that (6)+(2) and (7)+(3) are pullbacks, which by pullback decomposition implies that (6) and (7) are pullbacks. Note that $b$, $c$ and $d$ can be considered as typed over $C_B$, $C_C$ and $C_D$, respectively. So, according to Def. 4.1 we obtain that $b$ and $c$ agree in $d$ with respect to the pushout of the Cs, leading to an amalgamation $a = b + d c : P_A \rightarrow C_A$ with pullbacks (8) and (9) by Fact 4.2. Hence, $ac_{P_A} = \exists (a, ac_{C_A})$ is the required amalgamation.

- $ac_{P_D} = \bigwedge_{i \in I} ac_{P_D,i}$.
  Since $ac_{P_D}$ is a restriction of $ac_{P_B}$ and $ac_{P_C}$, they must be of the form $ac_{P_B} = \bigwedge_{i \in I} ac_{P_B,i}$ and $ac_{P_C} = \bigwedge_{i \in I} ac_{P_C,i}$. Moreover, since $ac_{P_B}$ and $ac_{P_C}$ agree in $ac_{P_D}$, we obtain that also $ac_{P_B,i}$ and $ac_{P_C,i}$ agree in $ac_{P_D,i}$ for all $i \in I$. So, by induction hypothesis, there are amalgamations $ac_{P_A,i} = ac_{P_B,i} + ac_{P_D,i} ac_{P_C,i}$ such that $ac_{P_B,i}$ and $ac_{P_C,i}$ are restrictions of $ac_{P_A,i}$ for all $i \in I$. Hence, $ac_{P_A} = \bigwedge_{i \in I} ac_{P_A,i}$ is the required amalgamation.
• The remaining case for disjunction works analogously to the case for conjunction.

The uniqueness of the amalgamation follows from the fact that we have an amalgamation in each level of nesting and the amalgamation of typed objects is unique by Fact 4.2.

**Decomposition.** We do an induction over the structure of $ac_{P_A}$:

- $ac_{P_A} = \text{true}$.
  - This case is trivial because $\text{true} = \text{true} + \text{true} \cdot \text{true}$.
- $ac_{P_A} = \exists (a, ac_{C_A})$ with $a : P_A \to C_A$.
  - Then by induction hypothesis, there exist restrictions $ac_{C_B}, ac_{C_C}$ and $ac_{C_D}$ of $ac_{C_A}$ such that $ac_{C_A} = ac_{C_B} + ac_{C_D} \cdot ac_{C_C}$. Moreover, by Fact 4.2 there are unique restrictions $b, c$ and $d$ of $a$ such that $a = b + d \cdot c$. Hence, we have restrictions $ac_{P_B} = \exists (b, ac_{C_B}), ac_{P_C} = \exists (c, ac_{C_C})$ and $ac_{P_D} = \exists (d, ac_{C_D})$ of $ac_{P_A}$, and, as shown for the case of composition before, the fact that $ac_{C_A} = ac_{C_B} + ac_{C_D} \cdot ac_{C_C}$ and $a = b + d \cdot c$ implies that $ac_{P_A} = ac_{P_B} + ac_{P_D} \cdot ac_{P_C}$.
- $ac_{P_A} = \bigwedge_{i \in I} ac_{P_A,i}$.
  - Then by induction hypothesis, there exist restrictions $ac_{P_B,i}, ac_{P_{C,i}}$ and $ac_{P_{D,i}}$ of $ac_{P_A,i}$ such that $ac_{P_A,i} = ac_{P_B,i} + ac_{P_D,i} \cdot ac_{P_{C,i}}$ for all $i \in I$. Hence, $ac_{P_B} = \bigwedge_{i \in I} ac_{P_B,i}, ac_{P_C} = \bigwedge_{i \in I} ac_{P_{C,i}}$ and $ac_{P_D} = \bigwedge_{i \in I} ac_{P_{D,i}}$ are restrictions of $ac_{P_A}$ such that $ac_{P_A} = ac_{P_B} + ac_{P_D} \cdot ac_{P_C}$.
- The remaining case for disjunction works analogously to the case for conjunction.

The uniqueness of the decomposition follows from the uniqueness of restrictions by pullback construction.

\[\square\]

**Theorem 4.10 (Amalgamation of Solutions for Positive Nested Conditions).** Given pushout (1) as in Def. 4.8 with all morphisms in $\mathcal{M}$, an amalgamation of typed objects $g_A = g_B + g_D g_C$, and an amalgamation of positive nested conditions $ac_{P_A} = ac_{P_B} + ac_{P_D} \cdot ac_{P_C}$ with corresponding matches $p_A = p_B + p_D P_C$.

**Composition.** Given solutions $Q_B$ for $p_B \models ac_{P_B}$ and $Q_C$ for $p_C \models ac_{P_C}$ agreeing in a solution $Q_D$ for $p_D \models ac_{P_D}$, then there is a solution $Q_A$ for $p_A \models ac_{P_A}$ constructed as amalgamation $Q_A = Q_B + Q_D Q_C$.

**Decomposition.** Given a solution $Q_A$ for $p_A \models ac_{P_A}$, then there are solutions $Q_B, Q_C$ and $Q_D$ for $p_B \models ac_{P_B}, p_C \models ac_{P_C}$ and $p_D \models ac_{P_D}$, respectively, which are constructed as restrictions $Q_B, Q_C$ and $Q_D$ of $Q_A$ such that $Q_A = Q_B + Q_D Q_C$.

The amalgamated composition and decomposition constructions are unique up to isomorphism.

**Proof.**

**Composition.** We perform an induction over the structure of $ac_{P_A}$.

- $ac_{P_A} = \text{true}$.
  - Then also $ac_{P_B}, ac_{P_C}, ac_{P_D}$ are true and we have empty solutions $Q_A, Q_B, Q_C$ and $Q_D$. Since the restriction of an empty solution is empty, we have that $Q_B$ and $Q_C$ are restrictions of $Q_A$.
- $ac_{P_A} = \exists (a, ac_{C_A})$ with $a : P_A \to C_A$.
  - By Fact 4.5 (Composition), we have the following diagram, where all rectangles are pushouts and all trapezoids are pullbacks, and all horizontal and vertical morphisms are in $\mathcal{M}$.
In order to show that \( q \) implies that we also have that \( q \), together with the pushout over the \( Q \) and analogously \( q \), moreover, we have

\[
\begin{align*}
\text{ac}_P &\triangleright P_a, \quad \text{ac}_C &\triangleright C_a, \\
p_B &\triangleright P_B, \quad pc &\triangleright \text{ac}_R, \\
p_C &\triangleright P_C, \quad \text{ac}_P &\triangleright \text{ac}_P, \\
p_D &\triangleright P_D, \quad \text{ac}_P &\triangleright \text{ac}_P.
\end{align*}
\]

Now, we consider solutions \( Q_B = (q_B, Q_CB) \), \( Q_C = (q_C, Q_CC) \) and \( Q_D = (q_D, Q_CD) \) for \( p_B \equiv \text{ac}_B \), \( p_C \equiv \text{ac}_R \) and \( p_D \equiv \text{ac}_P \), respectively, such that \( Q_D \) is a restriction of \( Q_B \) and \( Q_C \). Then we also have that \( q_D \) is a restriction of \( q_B \) and \( q_C \), and thus

\[
g_{BA} \circ q_B \circ c_{DB} = g_{BA} \circ g_{DB} \circ q_D = g_{CA} \circ g_{DC} \circ q_D = g_{CA} \circ q_C \circ c_{DC}.
\]

Together with the pushout over the Cs, this implies a unique morphism \( q_A : C_A \rightarrow G_A \) with \( q_A \circ c_{BA} = g_{BA} \circ q_B \) and \( q_A \circ c_{CA} = g_{CA} \circ q_C \).

Moreover, we have

\[
q_A \circ a \circ p_B = q_A \circ c_{BA} \circ b = g_{BA} \circ q_B \circ b = g_{BA} \circ p_B = p_A \circ p_B
\]

and analogously \( q_A \circ a \circ p_C = p_A \circ p_CA \). Since \( p_BA \) and \( p_CA \) are jointly epimorphic, this implies that \( q_A \circ a = p_A \).

In order to show that \( q_A \in M \), we consider the following diagram in the left:

\[
\begin{align*}
G_A &\leftarrow RCA \quad G_C &\leftarrow QC \quad C_C, \\
G_B &\leftarrow GDB \quad G_D &\leftarrow QD \quad C_D, \\
C_B &\leftarrow CDB \quad C_D &\leftarrow CD \quad C_D
\end{align*}
\]
We have that (6) is a pushout with all morphisms in $\mathcal{M}$ and thus also a pullback. Diagrams (7) and (8) are pullbacks by restriction, and (9) is a pullback because $q_D \in \mathcal{M}$ is a monomorphism. Hence, by composition of pullbacks, we obtain that the complete diagram is a pullback along $\mathcal{M}$-morphisms $g_{BA} \circ q_B$ and $g_{CA} \circ q_C$, which means that the pushout of the Cs is effective (see [Def. 2.3]), implying that $q_A \in \mathcal{M}$.

It remains to show that $q_B$ and $q_C$ are restrictions of $q_A$. In the following diagram, we have that (10) and (11) are pullbacks by restrictions, the Cs and the Gs form pushouts (see [Rem. 4.9]) and all morphisms in (10)-(13) are in $\mathcal{M}$. So, the horizontal as well as the vertical VK property implies that also (12) and (13) are pullbacks, which means that $q_B$ and $q_C$ are restrictions of $q_A$.

$$\begin{array}{c}
\text{acc}_A \triangleright C_A \\
\text{acc}_B \triangleright C_B \\
\text{acc}_C \triangleright C_C
\end{array}$$

Finally, $Q_D$ being a restriction of $Q_B$ and $Q_C$ means that $Q_{CD}$ is a restriction of $Q_{CB}$ and $Q_{CC}$ by induction hypothesis, this implies a solution $Q_{CA}$ of $q_A \models \text{acc}_A$ such that $Q_{CB}$ and $Q_{CC}$ are restrictions of $Q_{CA}$. Hence, $Q_A = (q_A, Q_{CA})$ is a solution for $p_A \models ac_A$ such that $Q_B$ and $Q_C$ are restrictions of $Q_A$.

- $ac_{p_A} = \bigwedge_{i \in \mathcal{I}} ac_{p_A,i}$.
  
  We have $ac_{p_B} = \bigwedge_{i \in \mathcal{I}} ac_{p_B,i}$, $ac_{p_C} = \bigwedge_{i \in \mathcal{I}} ac_{p_C,i}$ and $ac_{p_D} = \bigwedge_{i \in \mathcal{I}} ac_{p_D,i}$ such that for all $i \in \mathcal{I}$ there is $ac_{p_A,i}$ a restriction of $ac_{p_B,i}$ and $ac_{p_C,i}$.

  Moreover, given solutions $Q_B$, $Q_C$ and $Q_D$ of $p_B \models ac_{p_B}$, $p_C \models ac_{p_C}$ and $p_D \models ac_{p_D}$, respectively, we have $Q_B = (Q_{B,i})_{i \in \mathcal{I}}$, $Q_C = (Q_{C,i})_{i \in \mathcal{I}}$ and $Q_D = (Q_{D,i})_{i \in \mathcal{I}}$ such that for all $i \in \mathcal{I}$ we have that $Q_{B,i}, Q_{C,i}$ and $Q_{D,i}$ are solutions for $p_B \models ac_{p_B,i}$, $p_C \models ac_{p_C,i}$ and $p_D \models ac_{p_D,i}$, respectively, and $Q_{D,i}$ is a restriction of $Q_{B,i}$ and $Q_{C,i}$.

  Then, by induction hypothesis, there are solutions $Q_{A,i}$ for $p_A \models ac_{p_A,i}$ for all $i \in \mathcal{I}$ such that $Q_{B,i}$ and $Q_{C,i}$ are restrictions of $Q_{A,i}$. Hence, $Q_A = (Q_{A,i})_{i \in \mathcal{I}}$ is the required solution for $p_A \models ac_{p_A}$.

- $ac_{p_A} = \bigvee_{i \in \mathcal{I}} ac_{p_A,i}$.

  We have $ac_{p_B} = \bigvee_{i \in \mathcal{I}} ac_{p_B,i}$, $ac_{p_C} = \bigvee_{i \in \mathcal{I}} ac_{p_C,i}$ and $ac_{p_D} = \bigvee_{i \in \mathcal{I}} ac_{p_D,i}$ such that for all $i \in \mathcal{I}$ there is $ac_{p_A,i}$ a restriction of $ac_{p_B,i}$ and $ac_{p_C,i}$.

  Moreover, given solutions $Q_B$, $Q_C$ and $Q_D$ of $p_B \models ac_{p_B}$, $p_C \models ac_{p_C}$ and $p_D \models ac_{p_D}$, respectively, we have $Q_B = (Q_{B,i})_{i \in \mathcal{I}}$, $Q_C = (Q_{C,i})_{i \in \mathcal{I}}$ and $Q_D = (Q_{D,i})_{i \in \mathcal{I}}$ such that for some $j_B, j_C, j_D \in \mathcal{I}$ we have that $Q_{B,j_B}, Q_{C,j_C}$ and $Q_{D,j_D}$ are solutions for $p_B \models ac_{p_B,j_B}$, $p_C \models ac_{p_C,j_C}$ and $p_D \models ac_{p_D,j_D}$, respectively, and for all $k_B, k_C, k_D \in \mathcal{I}$ with $k_B \neq j_B$, $k_C \neq j_C$ and $k_D \neq j_D$ we have that $Q_{B,k_B}, Q_{C,k_C}$ and $Q_{D,k_D}$ are empty. Furthermore, $Q_{D,i}$ is a restriction of $Q_{B,i}$ and $Q_{C,i}$ for all $i \in \mathcal{I}$.

Case 1. $Q_{D,j_D} = \emptyset$.

Then we have $Q_{D,j} = \emptyset$ for all $j \in \mathcal{I}$. According to [Def. 3.3] only the restriction of
an empty solution is empty, implying that we also have \( Q_{B,j} = Q_{C,j} = \emptyset \) for all \( j \in \mathcal{I} \).
Moreover, since \( Q_{D,j_0} \) is a solution for \( p_0 \models acp_{P,D,j_0} \), we can conclude that \( acp_{P,D,j_0} = \text{true} \), and by the fact that \( acp_{D,j_0} \) is a restriction of \( acp_{A,j_0} \), \( acp_{B,j_0} \) and \( acp_{C,j_0} \) it follows that also \( acp_{A,j_0} = \text{true} \), \( acp_{B,j_0} = \text{true} \) and \( acp_{C,j_0} = \text{true} \). So, as shown above, there is a solution \( Q_{A,j_0} = \emptyset \) for \( p_0 \models acp_{A,j_0} \). Hence, \( Q_A = \{ Q_{A,i} \}_{i \in \mathcal{I}} \) with \( Q_{A,i} = \emptyset \) for all \( i \in \mathcal{I} \) is a solution for \( p_A \models acp_A \) such that \( Q_B \) and \( Q_C \) are restrictions of \( Q_A \).

**Case 2.** \( Q_{D,j_0} \neq \emptyset \).

Then according to [Def. 3.3] there are also \( Q_{B,j_0} \neq \emptyset \) and \( Q_{C,j_0} \neq \emptyset \) which means that \( j_B = j_C = j_D \). So, by induction hypothesis, there is a solution \( Q_{A,j_0} \) for \( p_A \models acp_{A,j_0} \) such that \( Q_{B,j_0} \) and \( Q_{C,j_0} \) are restrictions of \( Q_{A,j_0} \). Hence, \( Q_A = \{ Q_{A,i} \}_{i \in \mathcal{I}} \) with \( Q_{A,k} = \emptyset \) for all \( k \in \mathcal{I} \) with \( k \neq j_D \) is a solution for \( p_A \models acp_A \), and we have that \( Q_B \) and \( Q_C \) are restrictions of \( Q_A \).

In the first case \( (acp_A = \text{true}) \), the uniqueness of the amalgamation follows from the fact that an empty solution can only be the restriction of another empty solution. In the second case \((acp_A = \exists (a,acc_A))\), the uniqueness of \( Q_A = \{ q_A, Q_{CA} \} \) follows from the uniqueness of \( q_A \) by universal pushout property, and by uniqueness of \( Q_{CA} \) by induction hypothesis. Finally, in the cases of conjunction and disjunction, the uniqueness of the solution follows from uniqueness of its components by induction hypothesis.

**Decomposition.** Again, we perform an induction over the structure of \( acp_A \).

- **\( acp_A = \text{true} \).**
  
  Then we also have that \( acp_B, acp_C \) and \( acp_D \) are true. Moreover, we have that \( Q_A \) is empty, leading to empty restrictions \( Q_B, Q_C \) and \( Q_D \) that are solutions for \( p_B \models acp_B, p_C \models acp_C \) and \( p_D \models acp_D \), respectively.

- **\( acp_A = \exists (a,acc_A) \) with \( a : P_A \to C_A \).**
  
  Then we have \( acp_B = \exists (b,acc_B) \), \( acp_C = \exists (c,acc_C) \) and \( acp_D = \exists (d,acc_D) \). By amalgamation \( g_A = g_B + g_B g_C \), we have pullbacks (2)-(5) below. Moreover, by restrictions \( acp_B, acp_C \) and \( acp_D \) of \( acp_A \), we have restrictions \( b, c \) and \( d \) of \( a \), implying pullbacks (6)-(9) below. According to [Rem. 4.6] we have an amalgamation of positive nested conditions \( acc_A = acc_B + acc_D \) \( acc_C \), which implies an amalgamation of typed objects \( t_{CA} = t_{CB} + t_{CD} t_{CC} \) by [Def. 4.4].
Now, given a solution \( Q_A = (q_A, Q_{CA}) \) for \( p_A \models acp_A \), there is \( q_A : C_A \rightarrow G_A \in \mathcal{M} \) with \( q_A \circ a = p_A \).

Furthermore, we have
\[
g_A \circ q_A \circ c_{BA} = t_{CA} \circ c_{BA} = t g_{BA} \circ t_{CB}.
\]

which implies a unique morphism \( q_B : C_B \rightarrow G_B \) by pullback (2) such that \( g_B \circ q_B = t_{CB} \) and \( g_{BA} \circ q_B = q_A \circ c_{BA} \). Due to amalgamation \( t_{CA} = t_{CB} + t_{CD} \), we have that \( t_{CB} \) is a restriction of \( t_{CA} \) and thus (10)+(2) is a pullback. So, together with pullback (2), we obtain that also (10) is a pullback by pullback decomposition and, thus, \( q_B \) is a restriction of \( q_A \).

Moreover, by \( q_A, t g_{BA} \in \mathcal{M} \) and closure of \( \mathcal{M} \) under pullbacks, we know that \( q_B \circ g_{BA} \in \mathcal{M} \). Hence, by
\[
g_{BA} \circ p_B = p_A \circ p_{BA} = q_A \circ a \circ p_B = q_A \circ c_{BA} \circ b = g_{BA} \circ q_B \circ b,
\]

we obtain \( p_B = q_B \circ b \) because \( g_{BA} \in \mathcal{M} \) is a monomorphism.

Analogously, due to pullback (3) and restriction \( t_{CD} \) of \( t_{CA} \), there is a unique restriction \( q_C : C_C \rightarrow G_C \in \mathcal{M} \) of \( q_A \) with pullback (11) such that \( p_C = q_C \circ c \), and due to pullback (4) and restriction \( t_{CD} \) of \( t_{CB} \), there is a unique restriction \( q_D : C_D \rightarrow G_D \in \mathcal{M} \) of \( q_B \) with pullback (12) such that \( p_D = q_D \circ d \). Then, since \( t_{CD} \) is a restriction of \( t_{CC} \), (5)+(13) is a pullback which implies that also (13) is a pullback by pullback decomposition and pullback (5). Thus, \( q_D \) is also a restriction of \( q_C \), which means that we have \( q_A = q_B + q_D q_C \).

So, by induction hypothesis, there are solutions \( Q_{CB} \) for \( q_B \models ac_{CB} \), \( Q_{CC} \) for \( q_C \models ac_{CC} \), and \( Q_{CD} \) for \( q_D \models ac_{CD} \) such that \( Q_{CA} = Q_{CB} + Q_{CD} \). Hence, for \( Q_B = (q_B, Q_{CB}) \), \( Q_C = (q_C, Q_{CC}) \) and \( Q_D = (q_D, Q_{CD}) \) we obtain that \( Q_A = Q_B + Q_D Q_C \).

\( acp_A = \bigwedge_{i \in I} acp_{A,i} \).

Then we also have \( acp_B = \bigwedge_{i \in I} acp_{B,i} \), \( acp_C = \bigwedge_{i \in I} acp_{C,i} \), and \( acp_D = \bigwedge_{i \in I} acp_{D,i} \). Now, given a solution \( Q_A = (Q_{A,i})_{i \in I} \) for \( p_A \models acp_A \), then \( Q_{A,i} \) is a solution for \( p_A \models acp_{A,i} \) for all \( i \in I \).

Thus, by induction hypothesis for all \( i \in I \), there are solutions \( Q_{B,i} \) for \( p_B \models acp_{B,i} \), \( Q_{C,i} \) for \( p_C \models acp_{C,i} \), and \( Q_{D,i} \) for \( acp_{D,i} \) such that \( Q_{A,i} = Q_{B,i} + Q_{D,i} \). This in turn means that for all \( i \in I \) there are \( Q_{B,i} \) and \( Q_{C,i} \) restrictions of \( Q_{A,i} \), and \( Q_{D,i} \) is a restriction of \( Q_{B,i} \) and \( Q_{C,i} \).

Hence, for \( Q_B = (Q_{B,i})_{i \in I} \), \( Q_C = (Q_{C,i})_{i \in I} \) and \( Q_D = (Q_{D,i})_{i \in I} \) we have that \( Q_B \) and \( Q_C \) are restrictions of \( Q_A \), and \( Q_D \) is a restriction of \( Q_B \) and \( Q_C \), implying \( Q_A = Q_B + Q_D Q_C \).

\( acp_B = \bigvee_{i \in I} acp_{B,i} \).

Then we also have \( acp_B = \bigvee_{i \in I} acp_{B,i} \), \( acp_C = \bigvee_{i \in I} acp_{C,i} \), and \( acp_D = \bigvee_{i \in I} acp_{D,i} \). Given a solution \( Q_A = (Q_{A,i})_{i \in I} \) for \( p_A \models acp_A \), then there is \( j \in I \), such that \( Q_{A,j} \) is a solution for \( p_A \models acp_{A,j} \), and for all \( k \in I \) with \( k \neq j \) there is \( Q_{A,k} = \emptyset \).

By induction hypothesis, there are solutions \( Q_{B,j} \) for \( p_B \models acp_{B,j} \), \( Q_{C,j} \) for \( p_C \models acp_{C,j} \), and \( Q_{D,j} \) for \( p_D \models acp_{D,j} \) such that \( Q_{A,j} = Q_{B,j} + Q_{D,j} Q_{C,j} \). Hence, for \( Q_B = (Q_{B,i})_{i \in I} \), \( Q_C = (Q_{C,i})_{i \in I} \) and \( Q_D = (Q_{D,i})_{i \in I} \), where for all \( k \in I \) with \( k \neq j \) there is \( Q_{B,k} = Q_{C,k} = Q_{D,k} = \emptyset \), we have that \( Q_A = Q_B + Q_D Q_C \).

The uniqueness of the solutions follows from uniqueness of restrictions by pullback constructions.