Filtered compression for Kappa

Jonathan Hayman[1]

Computer Laboratory, University of Cambridge, United Kingdom

Abstract

Kappa is a language geared towards the modelling of the complex systems of reactions that can occur between proteins inside cells. It is supported by sophisticated simulation and static analysis techniques which can be applied to the models to study their emergent behaviour. Part of this is the construction by the simulator of causal histories explaining the generation of patterns of connectivity that the user specifies to be of interest, for example the existence of links between specified kinds of protein. Standard notions of independence of rule applications fail to provide adequately concise causal histories, leading to the earlier formulation of strong and weak forms of trajectory compression for Kappa. In this paper, we give a simple categorical account of how forms of compression can be uniformly obtained. This generalisation also describes a way for the user to specify their own levels of compression between weak and strong, which we call filtered compression. This is based on the idea of the user specifying the part of the type graph that represents the the structure which the compression technique should track through the trace.

1 Introduction

The rule-based approach to modelling biochemical systems taken by the Kappa [10] and BioNetGen [3] languages is motivated by the need to produce concise, maintainable models for the complex systems of reactions that can occur between proteins inside cells. They are supported by a range of tools for the stochastic simulation [7] and analysis [6] of the systems that they represent.

The Kappa simulator, KaSim, is capable of producing a *causal flow* to explain the pathway to the occurrence of patterns that the use specifies as interesting, for example the existence of a link between two of a particular kind of protein. As we shall see, in practice, it turns out that techniques from traditional concurrency theory based on the independence of events do not generate the kinds of pathway that a biochemist would expect: they often

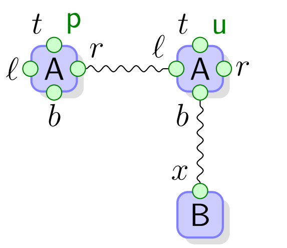
[1] The support of the ERC through the Advanced Grant ECSYM and the ANR Chair d'Excellence AbstractCell is gratefully acknowledged

contain sequences of events that it would be natural to omit from the account. To address this, KaSim allows the causal flow to be *compressed*. At present, the Kappa simulator allows a weak form of compression, though previous implementations have also allowed stronger forms of compression to take place and it is anticipated that these will be translated to the current simulator.

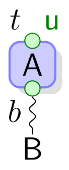
In [5], a non-stochastic semantics is given to Kappa in categories of graph-like structures. The application of rules to the mixture of entities inside the cell is described using the single-pushout approach [15] by taking a pushout in a category of partial maps. The forms of weak and strong compression are then described in [5] for Kappa, inspired by *simple* compression which has a much simpler and more abstract categorical account. In this paper, we work towards a more general formulation of compression by showing how a 'change of type' adjunction can be used to describe various levels of compression. The idea is that we take the original contact graph representing the structure permitted during rewriting and then generate a new type graph included in the old one by removing the parts that we do not wish to track the identity of during compression. For weak compression, we do not wish to track the identity of links between proteins and for strong compression we do not wish to track any structure at all. The lemma around which the compression technique revolves, Theorem 6.4, is then obtained for purely abstract reasons. Furthermore, the more abstract account allows forms of compression between weak and strong to be specified. We describe in the final section a simple example based on quasi-processive phosphorylation where this may be useful.

2 Kappa

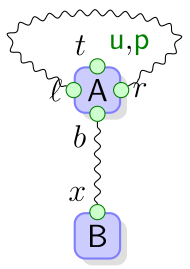
We begin by giving a brief overview of the structures involved in the graphical account of Kappa given in [5], to which we refer the reader for full definitions. The fundamental structures involved are called Σ-*graphs*. Σ-graphs are made up of *agents*, *sites*, *links* and *properties*: agents typically represent proteins, and sites reside on them to represent points of possible interaction, such as weak non-covalent bonds, with sites on other (or possibly the same) agents; these are represented by links. Apart from the existence of links, other properties can hold at sites, for example the site being phosphorylated or having positive or negative charge. A number of Σ-graphs are drawn in Figure 1, using squares for agents, circles for sites, lines for links and sans-serif labels for properties. Note that we label each agent and each site with its type; there can in general be more than one agent with a given type in a Σ-graph, but no two sites with the same type can be present on a single agent. As described in [5], the labels that can be used for the types of agents and sites are determined by the signature Σ. In the example, the signature specifies that A-type agents can have ℓ and r sites and B-agents can have x sites.



(a) A mixture. Every agent has every site consistent with the signature drawn.



(b) A site graph. A pattern matched by an A-agent with site t carrying the u property and, at site b, a link to some agent of type B. Note that the states of the sites ℓ and r is not specified; they are omitted from the pattern.



(c) A contact graph. A graph representing constraints on the structure of graphs: the site t may carry either u or p properties; the sites ℓ and r on A-type agents may be linked but may not carry any other property; and there can be links between b on A-type agents and x on B-type agents.

Fig. 1. Examples of Σ-graph, where the signature Σ allows two agent types, A and B, and specifies that A-type agents may have t-, b-, ℓ- and r-type sites and B-type agents may have x-type sites. Sites may carry u or p properties.

Special kinds of Σ-graph play a variety of rôles: *Mixtures* are the Σ-graphs that are typically rewritten, representing the kinds of structure that are physically meaningful; every agent has the state of every appropriate (according to the signature) site fully specified, and each site can be linked to at most one other site. *Site graphs* are used as patterns in the specification of transformation rules and are more loosely constrained than mixtures: they do not need to mention the state of every site on every agent, and may also possess 'one-ended' *anonymous* links to represent the existence of a link to *some* site, possibly on an agent of a specified type. These are of use when we wish patterns to be matched injectively on agents but do not necessarily wish to impose this constraint on the target of certain links. *Contact graphs* are used to represent the possible connections that might exist between sites on agents. They have at most one agent of every given agent type, but more than one link can originate at each site.

Homomorphisms of Σ-graphs embed the structure of one Σ-graph into that of another. The formal details are given in [5], but essentially a homomorphism f from G to H is a function from the agents, sites, links and properties of G to those of H. It is required to preserve structure in the sense that it must preserve the types of agents and sites; for any site s on an agent n in G, the site $f(s)$ in H must be on $f(n)$; the image of a link between sites s and s' must connect $f(s)$ and $f(s')$; the image of an anonymous link can be either an anonymous link or a link connecting two sites, but in either case the image must satisfy any constraints on the link in G such as to being a site on an agent of a given type; and finally, the image of a property on a site must be the same property on the image of the site. As examples of homomorphisms, there are homomorphisms from the Σ-graphs drawn in Figure 1a and Figure 1b to that in Figure 1c. We write $\Sigma\mathbb{G}$ for the category of Σ-graphs connected by homomorphisms.

3 Rewriting

We now turn to a higher-level view of how categories of structures can describe rewriting (cf HLR rewriting systems [11]).

3.1 Partial maps

We begin by assuming a category $\mathcal{C}$ consisting of the objects that will be involved in rewriting (for example, graphs to represent the structures being rewritten, the patterns in rules and graphs to represent types). In the case of Kappa, this will be the category $\Sigma\mathbb{G}$.

Partial maps are introduced with respect to a class of *inclusions*; we require that we are given a subcategory $\mathcal{I}$ of $\mathcal{C}$ specifying them. We write $\hookrightarrow$ to indicate arrows in $\mathcal{I}$. The category $\mathcal{I}$ is required to satisfy the following *admissibility* constraints.

Definition 3.1 [Admissible inclusions] The subcategory $\mathcal{I}$ is *admissible* if:

(i) For every object X in $\mathcal{C}$, the identity id_X is in $\mathcal{I}$

(ii) Every morphism in $\mathcal{I}$ is a mono in $\mathcal{C}$

(iii) For every morphism $f : X \to Z$ in $\mathcal{C}$ and every $i : Y \hookrightarrow Z$ in $\mathcal{I}$, there is a pullback in $\mathcal{C}$ of the form

$$\begin{tikzcd} P \arrow[r, hook, "j"] \arrow[d, "g"'] \arrow[dr, phantom, "\lrcorner", very near start] & X \arrow[d, "f"] \\ Y \arrow[r, hook, "i"] & Z \end{tikzcd}$$

where j is in $\mathcal{I}$.

(iv) For any pair of morphisms $f : X \hookrightarrow Y$ and $f' : X' \hookrightarrow Y$ in $\mathcal{I}$ and isomorphism $\phi : X \to X'$ in $\mathcal{C}$, if $f' \circ \phi = f$ then $X = X'$ and $\phi = \mathsf{id}_X$

For Kappa, we say that a Σ-graph G_0 is a sub-graph of G if the sets of agents, sites, links and properties of G_0 are included in those of G. It is straightforward to verify that the category of sub-graph inclusions is admissible.

A partial map from X to Y shall be constructed first by defining an inclusion $i : D \hookrightarrow X$, where D specifies where the partial map is defined, called its the *domain of definition*; we then define a morphism in $\mathcal{C}$ from D to Y.

Definition 3.2 A partial map from X to Y is a span of the form (i, D, h) where $i : D \hookrightarrow X$ is a morphism in $\mathcal{I}$ and $h : D \to Y$ is a morphism in $\mathcal{C}$.

Partial maps $(i, D, f) : X \to Y$ and $(j, E, g) : Y \to Z$ compose by taking the pullback described in part (iii) of the admissibility constraint on $\mathcal{I}$:

$$\begin{tikzcd}
 & & P \arrow[dl, hook, "j'"'] \arrow[dr, "f'"] \arrow[dd, phantom, "\vee"] & & \\
X & D \arrow[l, hook', "i"'] \arrow[dr, "f"'] & & E \arrow[r, "g"] & Z \\
 & & Y & E \arrow[l, phantom] &
\end{tikzcd}$$

$$\begin{tikzcd}
 & P \arrow[dl, hook, "j'"'] \arrow[dr, "f'"] & \\
D \arrow[dr, "f"'] & & E \arrow[dl, hook', "j"] \\
 & Y &
\end{tikzcd}$$

Note that this is well-defined since the pullback object described in (iii) of the admissibility constraint is necessarily unique (*not* just up to isomorphism):

Lemma 3.3 *If the category $\mathcal{I}$ is admissible then for every $f : X \to Z$ in $\mathcal{C}$ and $i : Y \hookrightarrow Z$ in $\mathcal{I}$, there is a unique pullback as follows for which j is in $\mathcal{I}$.*

$$\begin{tikzcd}
P \arrow[r, hook, "j"] \arrow[d, "g"'] \arrow[dr, phantom, "\lrcorner", very near start] & X \arrow[d, "f"] \\
Y \arrow[r, hook, "i"] & Z
\end{tikzcd}$$

Associativity of composition is a consequence of the final constraint. The category of partial maps is written $\mathcal{C}_*^{\mathcal{I}}$ or, where $\mathcal{I}$ is obvious from the context, just $\mathcal{C}_*$. Of course, every map in $\mathcal{C}$ induces a map in $\mathcal{C}_*$ where we take the inclusion to be the identity; we call such maps in $\mathcal{C}_*$ *total.* The following lemma establishes that composition in $\mathcal{C}$ and $\mathcal{C}_*$ coincide.

Lemma 3.4 *The inclusion $\mathcal{C} \hookrightarrow \mathcal{C}_*$ preserves pullbacks and preserves monos.*

We note here that the admissibility constraints are similar to those in [17] apart from the addition of the final constraint and the requirement in the third part that the map j be in $\mathcal{I}$. These additions allow us to avoid the complexities of having to take domains of definition to be isomorphism classes of graphs. For example, with the definition here, we can take the domain of definition of a partial map between Σ-graphs to be a sub-graph of the source instead of being the *isomorphism class* of a sub-graph. This restriction will simplify the following account when we come to describe compression. However, if a compelling reason for reverting to the more standard definition were to emerge, we believe that the development here could be straightforwardly translated (though the coming adjunction would then involve a pseudo-functor).

3.2 Rewriting and trajectories

We now describe the structure that we assume to describe rewriting. Briefly, we shall describe a rewrite rule as a partial map in $\mathcal{C}_*$. Only certain partial maps might be permitted to describe rules; we call such rules *action maps.* A rule can be applied to rewrite a structure when its domain *matches* the structure being rewritten. Action maps and matchings will be represented by partial maps in $\mathcal{C}_*$ (often, matchings are total, but the extra generality adds no complication). The produced structure is obtained by taking a pushout in $\mathcal{C}_*$. Action maps and matchings for Kappa are specified in [5].

Formally, we assume that we are given categories $\mathcal{M}$ and $\mathcal{A}$ of matchings and action maps, respectively, that are both subcategories of $\mathcal{C}_*$ such any span $S \overset{m}{\leftarrow} L \overset{a}{\rightarrow} R$ where m is in $\mathcal{M}$ and a is in $\mathcal{A}$ has a pushout in $\mathcal{C}_*$.

Given a set of rules specified by the user represented as a subset of action maps $\mathbf{A}$ and an initial object G, a *trajectory* is a finite sequence of pushouts of action maps in $\mathbf{A}$ from the object G:

$$\begin{tikzcd}
L_0 \arrow[r, "\alpha_0"] \arrow[d, "e_0"'] & R_0 \arrow[d, "p_0"] & \cdots & L_i \arrow[r, "\alpha_i"] \arrow[d, "e_i"'] & R_i \arrow[d, "p_i"] & \cdots & L_n \arrow[r, "\alpha_n"] \arrow[d, "e_n"'] & R_n \arrow[d, "p_n"] \\
G = G_0 \arrow[r, "\beta_i"] & G_1 & \cdots & G_i \arrow[r, "\beta_i"] & G_{i+1} & \cdots & G_n \arrow[r, "\beta_n"] & G_{n+1}
\end{tikzcd}$$

4 Compression

It is often useful to be able to describe how it was possible for a selected (typically the final) rule in a trajectory to be applied. This is, for example, a central aspect of the automatic generation of biochemical *pathways* through simulation, where the final rule application represents the observation of a pattern of interest called an *observable.* The whole trajectory is one possible answer, but frequently trajectories contain unnecessary rule applications that we would wish to elide in our response.

A first attempt to answer this question is based on the well-known notion of sequential concurrency in graph rewriting [15]. Though it will turn out to fail to remove enough rule applications from the trace to be practically useful for the study of biochemical pathways, we describe it since the notions introduced can be used in the presentation of the results of later forms of compression.

4.1 Mazurkiewicz compression

Suppose that we have two consecutive rule applications generated by pushouts of action maps α_1, α_2 against matchings e_1, e_2:

$$\begin{tikzcd}
L_1 \arrow[r, "\alpha_1"] \arrow[d, "e_1"'] & R_1 \arrow[d, "p_1"'] & L_2 \arrow[r, "\alpha_2"] \arrow[dl, "e_2"] & R_2 \arrow[d, "p_2"] \\
G_1 \arrow[r, "\beta_1"] & G_2 \arrow[rr, "\beta_2"] & & G_3
\end{tikzcd}$$

There is no *immediate causal dependency* between the two rule applications if there exists a matching $e_2' : L_2 \to G_1$ such that $e_2 = \beta_1 \circ e_2'$. From the pushout existence condition on the the selection of subcategories of action maps and matchings, there must exist a pushout of α_2 against e_2' as follows:

$$\begin{tikzcd} L_2 \arrow[r, "\alpha_2"] \arrow[d, "e_2'"'] & R_2 \arrow[d, "p_2'"] \\ G_1 \arrow[r, "\beta_2'"'] \arrow[ur, phantom, "\ulcorner" near end] & G' \end{tikzcd}$$

We say that the two rule applications are *independent* if there is no immediate causal dependency between them and, for some pushout (β_2', G, p_2') as drawn above, the composition $\beta_2' \circ e_1$ is a matching. It is not hard to show that the following is a pushout in $\mathcal{C}_*$, demonstrating the commutation of consecutive independent actions:

$$\begin{tikzcd} L_1 \arrow[r, "\alpha_1"] \arrow[d, "r_2' \circ e_1"'] & R_1 \arrow[d, "p_1'"] \\ G' \arrow[r, "\beta_1'"'] \arrow[ur, phantom, "\ulcorner" near end] & G_2 \end{tikzcd}$$

Furthermore, $\beta_1' \circ p_2' = p_2$ and $\beta_2 \circ p_1 = p_1'$, meaning that the permutation of the rule applications preserves the right-hand side of each rule.

Essentially, we have reconstructed the well-known notion of *sequential concurrency* within the framework of action maps and matchings. The absence of immediate causal dependency shows that the rule α_2 could have been applied to the same part of the structure before the occurrence of β_1, and the fact that $\beta_2' \circ e_1$ is an embedding shows that β_2' does not inhibit the occurrence of α_1 in the resulting state.

Given a trajectory from an initial object, we can permute consecutive independent rule applications to generate other valid trajectories with the goal of making a selected rule application occur as early as possible. Viewing trajectories as equivalent if they differ only in the order of consecutive independent rule applications, there will be a unique such trajectory. Anything following the selected rule application can be removed from the account of how the selected rule application was produced to yield what we call the *maximally Mazurkiewicz compressed* trajectory.

Unfortunately, this technique very often does not yield adequately concise trajectories for the analysis of biochemical pathways. There are two issues to contend with (both explained with examples in [5]). As an example of the first, suppose that the production of an observable depends on an earlier rule application that deletes some part of the mixture that is tested by its immediately prior rule application. The observable does not depend on the rule application that tests the mixture, but the test and the deletion rule applications cannot be permuted. Therefore, the test rule application will always

$$\begin{tikzcd}
G'_n \arrow[rrrr, "\gamma_n"] & & & & G_{n+1} \\
G'_m \arrow[u, "\beta'_n"'] \arrow[rrr, "\gamma_m"] & & & G_{m+1} \arrow[r, "\beta_{n-1} \circ \ldots \circ \beta_{m+1}"] & G_n \arrow[u, "\beta_n"'] \\
G'_j \arrow[u, dotted, dash] \arrow[rr, "\gamma_j"] & & G_{j+1} \arrow[r, "\beta_{k-1} \circ \ldots \circ \beta_{j+1}"] & G_k \arrow[u, dotted, dash] & \\
G'_i \arrow[u, "\beta'_j"] \arrow[r, "\gamma_i"] & G_{i+1} \arrow[r, "\beta_{j-1} \circ \ldots \circ \beta_{i+1}"] & G_j \arrow[u, "\beta_j"'] & & \\
G_0 \arrow[u, "\beta'_i"] \arrow[r, "\beta_{i-1} \circ \ldots \circ \beta_0"] & G_i \arrow[u, "\beta_i"'] & & &
\end{tikzcd}$$

Fig. 2. Steps of compression, each step justified by an application of Lemma 4.1

be in the maximally Mazurkiewicz compressed trajectory. This suggests that symmetric forms of independence such as the one given above sometimes fail to be appropriate. Secondly, and in practice more importantly, we need to consider the *composition* of rule applications. A typical example may be that the production of an observable depends on a rule application that tests for the absence of a link between two sites. Immediately prior to this, such a link is created and then destroyed (perhaps with some other effect). In many situations, it is appropriate to remove these two rule applications from the compressed trajectory since their combined effect towards the production of the observable is nil. However, no Mazurkiewicz compression is possible since no consecutive pair of rule applications is independent.

4.2 Simple compression

Simple compression relaxes the requirements of Mazurkiewicz compression to allow rule applications to be moved to occur earlier in the trace. It begins with the following simplification of the permutation property from Mazurkiewicz compression:

Lemma 4.1 *Suppose a partial map $\beta_0 : G_0 \to G_1$, an action map $\alpha : L \to R$ and a matching $e : L \to G_1$, all in $\mathcal{C}_*$. If there is a matching $e' : L \to G_0$ such that $\beta_0 \circ e' = e$ then there is a unique partial map $\gamma : G'_0 \to G_2$ in $\mathcal{C}_*$ such that $\gamma \circ p' = p$ and $\gamma \circ \beta'_0 = \beta_1 \circ \beta_0$, where the pushouts are as drawn to the right.*

$$\begin{tikzcd}
G'_0 \arrow[rr, "\gamma"] & & G_2 \\
& R \arrow[ul, "p'"] \arrow[ur, "p"'] & \\
& L \arrow[u, "\alpha"'] \arrow[dl, "e'"'] \arrow[dr, "e"] & \\
G_0 \arrow[uuu, "\beta'_0"] \arrow[rr, "\beta_0"'] & & G_1 \arrow[uuu, "\beta_1"']
\end{tikzcd}$$

Note that the map β_0 may represent the composition of a number of earlier rule applications, and that there is no guarantee that the map γ, which is called the *residual*, is itself derivable through a series of rule applications. For these reasons, compression using this lemma must take a global view of the trajectory being compressed rather than being based on the local (within the trajectory) permutation of consecutive rule applications.

Consider a trajectory $G_0 \xrightarrow{\beta_0} G_1 \xrightarrow{\beta_1} \cdots \xrightarrow{\beta_{n-1}} G_n \xrightarrow{\beta_n} G_{n+1}$, each step justified by a pushout as drawn at the end of Section 3.2. Intuitively, the goal of compression is to remove as many rule applications as possible to yield a valid trajectory from G_0 culminating in a rule application corresponding to β_n. Formally, a simple compression is a selection of indices $\{i, j, k, \ldots, m\} \subseteq \{1, \ldots, n-1\}$, taking $i < j < k < \cdots < m$, for which there is a sequence of applications of Lemma 4.1 as drawn in Figure 2. The lowest step is generated by a matching $e'_i : L_i \to G_0$ commuting with the original matching through the composition of the intermediate rule applications, *i.e.* such that $e_i = \beta_{i-1} \circ \cdots \circ \beta_0 \circ e'_i$; this generates a partial map $\beta'_i : G_0 \to G'_i$ representing the rule application and a residual $\gamma_i : G'_i \to G_{i+1}$. Later steps require the existence of a matching that commutes with the original matching through the composition of the prior rule applications and the previous residual. There is a final application of Lemma 4.1 to establish that the same rule as that generating β_n can be applied in G'_m, with the matching commuting with that in G_n through the composition of the residual γ_m and the composition of the intermediate rule applications.

We say that the compression if *maximal* if the set $\{i, j, k, \ldots, m\}$ is a minimal such set; in general, there may be more than one maximal compression of a given trajectory.

4.3 Weak compression

Returning to Kappa, in requiring the commutation of the lower triangle in the simple compression lemma (Lemma 4.1), we require that anything required to exist in G_1 according to the matching e is present in G_0 and preserved by β_0. For example, the morphism β_0 could represent the composition of a pair of rule applications that create and then delete a link between sites on two agents. Suppose that the observable that we are interested in is that there is no link between the agents; this is tested using fact that matchings in Kappa have to preserve the absence of links on unlinked sites: a negative application condition [12]. The graphs L and R are equal and consist of just two agents and the appropriate sites with no link between them, and the application of the rule to G_1 generates the rule application β_1. There would be no matching of L into the intermediate mixture between G_0 and G_1 where the link had been created but not deleted, but there would be a matching e' making the lower triangle commute since the two agents are preserved through β_0.

This reveals a subtle issue with simple compression: compression would not be able to take place in the opposite situation where the rule β_0 represents the composition of rule applications that successively delete and then create a link between two agents and the observable, represented by the rule application generating β_1, is that the link exists. In this situation, the lower triangle would

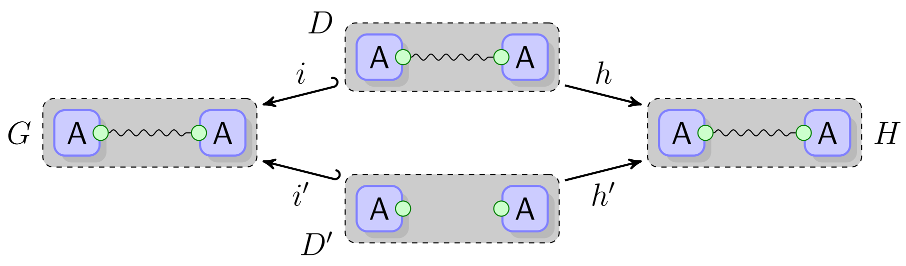


Fig. 3. Partial maps equivalent on agents, $(i, D, h) \sim_{\mathsf{ag}} (i', D', h')$ (site labels omitted). Each map sends the left agent in its source to the left agent in its target and right in its source to right in its target.

not commute: the morphism e would have the link in its domain of definition but the composition $\beta_0 \circ e'$ would not. For this reason, in [5], we considered *weak* compression, which required commutation of the lower square only on agents. For example, the partial maps (i, D, h) and (i', D', h') in Figure 3 are regarded as equivalent on agents.

Lemma 4.2 (Lemma 9 from [5])
Suppose a partial map $\beta_0 : G_0 \to G_1$, an action map $\alpha : L \to R$ and a matching $e : L \to G_1$, all in $\Sigma\mathbb{G}_*$. If there is a matching $e' : L \to G_0$ such that $\beta_0 \circ e' \sim_{\mathsf{ag}} e$ then there is a partial map $\gamma : G'_0 \to G_2$ in $\Sigma\mathbb{G}_*$ such that $\gamma \circ p' \sim_{\mathsf{ag}} p$ and $\gamma \circ \beta'_0 \sim_{\mathsf{ag}} \beta_1 \circ \beta_0$, where the pushouts are as drawn to the right. Furthermore, for any $\gamma' : G'_0 \to G_2$ in $\Sigma\mathbb{G}_*$ such that $\gamma \circ p' \sim_{\mathsf{ag}} p$ and $\gamma \circ \beta'_0 \sim_{\mathsf{ag}} \beta_1 \circ \beta_0$, we have $\gamma \sim_{\mathsf{ag}} \gamma'$.

$$\begin{tikzcd}
G'_0 \arrow[rr, "\gamma"] & \sim_{\mathsf{ag}} & G_2 \\
& R \arrow[ul, "p'"'] \arrow[ur, "p"] & \\
& L \arrow[u, "\alpha"] \arrow[dl, "e'"'] \arrow[dr, "e"] & \\
G_0 \arrow[uuu, "\beta'_0"] \arrow[rr, "\beta_0"'] & \sim_{\mathsf{ag}} & G_1 \arrow[uuu, "\beta_1"']
\end{tikzcd}$$

Note that Lemma 4.2 is specifically for categories of Σ-graphs, in contrast to Lemma 4.1 which is for arbitrary rewriting categories: the proof of Lemma 4.2 relies on a concrete construction of a partial map γ of Σ-graphs, whereas Lemma 4.1 can be proved abstractly. The reason for this is the absence of a categorical interpretation for commutation on agents. This reveals the fundamental issue in attempting to gain a more abstract understanding of compression to be tackled in the remainder of this paper, namely how to abstractly generate equivalences on partial maps that make the compression lemma hold.

5 Typed reduction

In this section, we briefly describe typed reduction in Kappa. In Kappa, a graph called the *contact graph* describes the links that are permitted to exist between agents of a given type [5]; it is used in both the simulation [8,7] and static analysis algorithms [6]. The contact graph describes the *type* of structure that can be generated, for example that a link may exist between given sites on given types of agent. Here, as mentioned in the introduction, inclusions of the type graph will be used in the following section to generate equivalence on partial maps, allowing us to parameterize the form of compression that we wish to use by restricting to the parts of the structure that are of interest.

We first review a few essential components of categories over a given object.

Definition 5.1 [Slice category] Let C be any object of a category $\mathcal{C}$. The slice category $\mathcal{C}/C$ consists of morphisms $g : G \to C$ in $\mathcal{C}$. A morphism $\alpha : g \to h$ in $\mathcal{C}/C$ from $g : G \to C$ to $h : H \to C$ is a morphism $\alpha : G \to H$ of $\mathcal{C}$ satisfying $h \circ \alpha = g$.

The partial map category over the slice $(\mathcal{C}/C)_*$ has the same objects as $\mathcal{C}/C$ but its morphisms are spans (i, D, α) of morphisms in $\mathcal{C}$ such that the diagram to the right commutes and $i : D \to G$ is in $\mathcal{I}$.

$$\begin{tikzcd}
 & D \arrow[dl, hook', "i"'] \arrow[dr, "\alpha"] & \\
G \arrow[dr, "g"'] & & H \arrow[dl, "h"] \\
 & C &
\end{tikzcd}$$

In what follows, we adopt the convention of using lower-case letters for morphisms into C, the corresponding upper case letter for their domain and Greek letters for morphisms in the slice category.

There is a forgetful functor from $(\mathcal{C}/C)_*$ to $\mathcal{C}_*$ that takes objects g in $(\mathcal{C}/C)_*$ to their domain G and acts as the identity on morphisms. The partial map category over a slice might, in general, have different categorical properties to the original category. Composition of partial maps is unaffected thanks to the following straightforward result about pullbacks of total maps.

Lemma 5.2 *Suppose that the image under the forgetful functor of a co-span of total morphisms $g \xrightarrow{\theta} k \xleftarrow{\phi} h$ in $(\mathcal{C}/C)_*$ has a pullback $G \xleftarrow{\phi'} P \xrightarrow{\theta'} H$ in $\mathcal{C}$. Taking $p = g \circ \phi'$, the span $h \xleftarrow{\phi'} p \xrightarrow{\theta'} h$ is a pullback in both $\mathcal{C}/C$ and $(\mathcal{C}/C)_*$.*

The situation for pushouts is more subtle: pushouts in $\mathcal{C}_*$ and $(\mathcal{C}/C)_*$ only coincide under certain circumstances.

Definition 5.3 Say that partial pushouts and partial slice pushouts over C *coincide* for a category $\mathcal{C}$ and object C in $\mathcal{C}$ iff the forgetful functor from $(\mathcal{C}/C)_*$ to $\mathcal{C}_*$ preserves pushouts and any span in $(\mathcal{C}/C)_*$ has a pushout in $(\mathcal{C}/C)_*$ if its image under the forgetful functor into $\mathcal{C}_*$ has a pushout.

This property allows us to view pushouts in the (partial map) slice category and pushouts in the original partial map category interchangeably. It implies a sense of type-preservation property, that taking a pushout of a pair of type-preserving maps (rewriting) yields an object consistent with the type.

Pushouts in the partial map category and the partial map category over the slice do not always coincide. However, the slice of $\Sigma\mathbb{G}$ over any contact graph is an example where they do. We believe that the Mipmap categories of [13] will provide a more abstract characterization, though (as discussed in the conclusion) a full development is beyond the scope of the current paper.

6 Filters

To address the situation where we do not wish to require the lower triangle in the simple compression lemma to commute 'on the nose', we can introduce notions of equivalence on partial maps between the same source and target structures. In this section, we introduce the idea of commutation up to a filter induced by an inclusion into the type graph; this will allow the expression of, for example, commutation only on agents (as seen to be useful for weak compression). We then see that this immediately extends, for abstract reasons, to give an analogue of the weak compression lemma. However, by giving the user flexibility to define their own inclusion with respect to which compression occurs, more refined kinds of compression can be obtained.

Given an inclusion $c : C_0 \hookrightarrow C$ in $\mathcal{I}$, as we have seen, we can construct the categories $(\mathcal{C}/C_0)_*$ and $(\mathcal{C}/C)_*$. We define functors between the categories as follows, yielding an adjunction.

$$\begin{tikzcd}
(\mathcal{C}/C_0)_* \arrow[r, bend left, "c_*"] \arrow[r, phantom, "\perp"] & (\mathcal{C}/C)_* \arrow[l, bend left, "c^*"]
\end{tikzcd}$$

- The functor $c_* : (\mathcal{C}/C_0)_* \to (\mathcal{C}/C)_*$ is the post-composition functor: it sends an object $g : G \to C_0$ of $(\mathcal{C}/C_0)_*$ to $g \circ c : G \to C$ and acts as the identity on morphisms.
- The functor $c^* : (\mathcal{C}/C)_* \to (\mathcal{C}/C_0)_*$ is the inverse image functor: it sends an object $g : G \to C$ of $(\mathcal{C}/C)_*$ to the morphism $c^*(g)$ drawn in the following pullback diagram in the category $\mathcal{C}$ where ϵ_g is an inclusion.

$$\begin{tikzcd}
c^*(G) \arrow[r, hook, "\epsilon_g"] \arrow[d, "c^*(g)"'] \arrow[dr, phantom, "\lrcorner", very near start] & G \arrow[d, "g"] \\
C_0 \arrow[r, hook, "c"'] & C
\end{tikzcd}$$

Note that this is a pullback in the category $\mathcal{C}_*$ by Lemma 3.4. This is well-defined since there is a unique such pullback according to Lemma 3.3; had we not imposed the additional constraints on the admissible inclusions, we would have had to define $c^*(g)$ to be the isomorphism class of pullbacks, yielding a pseudo-functor.

On morphisms, the functor c^* sends $\alpha : G \to H$ to the mediating morphism $c^*(\alpha)$ drawn in the following diagram in $\mathcal{C}_*$; this uniquely defines $c^*(\alpha)$ since $c^*(H)$ is a pullback in $\mathcal{C}$ and hence in $\mathcal{C}_*$.

$$\begin{tikzcd}
c^*(G) \arrow[r, hook, "\epsilon_g"] \arrow[dr, dotted, "c^*(\alpha)"] \arrow[ddr, bend right, "c^*(g)"'] & G \arrow[dr, "\alpha"] & \\
& c^*(H) \arrow[r, hook, "\epsilon_h"] \arrow[d, "c^*(h)"'] \arrow[dr, phantom, "\lrcorner", very near start] & H \arrow[d, "h"] \\
& C_0 \arrow[r, hook, "c"'] & C
\end{tikzcd}$$

Theorem 6.1 *The functor c_* is left-adjoint to the functor c^*.*

The central idea of the generalisation presented in this paper is to view partial maps to be equivalent if they have the same source and target and have the same image under the functor c^*.

Definition 6.2 For partial maps $\alpha, \beta : g \to h$ in $(\mathcal{C}/C)_*$, write $\alpha \sim \beta$ iff $c^*(\alpha) = c^*(\beta)$.

We have seen that for any $h : H \to C$, the morphism $\epsilon_h : c^*(H) \to H$ is also an inclusion; from this, we derive an alternative characterization of equivalence:

Lemma 6.3 *For any object h in $(\mathcal{C}/C)_*$, let $\overrightarrow{\epsilon_h} : H \to c^*(H)$ denote the partial map $(\epsilon_h, c^*(H), \mathsf{id}_{c^*(H)})$. Then for any $\alpha : g \to h$ in $(\mathcal{C}/C)_*$,*

$$c^*(\alpha) = \overrightarrow{\epsilon_h} \circ \alpha \circ \epsilon_g.$$

Furthermore, for partial maps $\alpha, \alpha' : g \to h$,

$$c^*(\alpha) = c^*(\alpha') \iff \overrightarrow{\epsilon_h} \circ \alpha = \overrightarrow{\epsilon_h} \circ \alpha' \iff \alpha \circ \epsilon_g = \alpha' \circ \epsilon_g.$$

The partial map $\overrightarrow{\epsilon_h}$ is undefined on the part of H that under h is outside the image of C_0 in C; elsewhere, it acts as the identity. It acts like a *filter*, removing the distinction between morphisms on the part where it is undefined. We are now able to give our more abstract form of the compression lemma, making no reference to Σ-graphs.

Theorem 6.4 *Suppose a partial map $\beta_0 : m_0 \to m_1$, an action map $\alpha : \ell \to r$ and a matching $e_1 : \ell \to m_1$, all in $(\mathcal{C}/C)_*$. If there is a matching $e_0 : \ell \to m_0$ such that $\beta_0 \circ e_0 \sim e_1$ then there is a partial map $\gamma : m_0' \to m_2$ in $(\mathcal{C}/C)_*$ such* *that $\gamma \circ n_0 \sim n_1$ and $\gamma \circ \beta'_0 \sim \beta_1 \circ \beta_0$, where the pushouts are as drawn below:*

$$\begin{tikzcd}
m'_0 \arrow[rr, "\gamma"] & \sim & m_2 \\
 & r \arrow[ul, "n_0"'] \arrow[ur, "n_1"] & \\
 & \ell \arrow[u, "\alpha"'] \arrow[dl, "e_0"'] \arrow[dr, "e_1"] & \\
m_0 \arrow[uuu, "\beta'_0"] \arrow[rr, "\beta_0"'] & \sim & m_1 \arrow[uuu, "\beta_1"']
\end{tikzcd}$$

Furthermore, any other partial map $\gamma' : m'_0 \to m_2$ in $(\mathcal{C}/C)_*$ such that $\gamma' \circ n_0 \sim n_1$ and $\gamma' \circ \beta'_0 \sim \beta_1 \circ \beta_0$ satisfies $\gamma \sim \gamma'$.

The compression 'lemma', Theorem 6.4, can be applied multiple times, as described for simple compression, to give maximally compressed (with respect to the given inclusion) trajectories leading to the production of the observable.

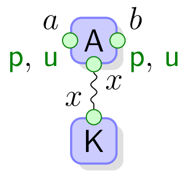
Example 6.5 [Weak compression] The weak compression lemma can be obtained in the following way. Let C be the contact graph for a given Kappa system. The contact graph C_0 consists of the agents and sites of C but no links or properties on sites and c is the inclusion of C_0 in C. For any $h : H \to C$ in $\Sigma\mathbb{G}/C$, the Σ-graph $c^*(H)$ consists of the agents and sites of H but no links or properties. Applying Lemma 6.3, any pair of morphisms $f, f' : g \to h$ in $(\Sigma\mathbb{G}/C)_*$ have the same image under c^* iff they satisfy $f \sim_{\mathsf{ag}} f'$.

Example 6.6 [Strong compression] Strong compression for Kappa [5] is obtained by taking the contact graph C_0 to be the empty Σ-graph. In this case, any two morphisms $f, f' : G \to H$ in $(\Sigma\mathbb{G}/C)_*$ have the same image under c^*.

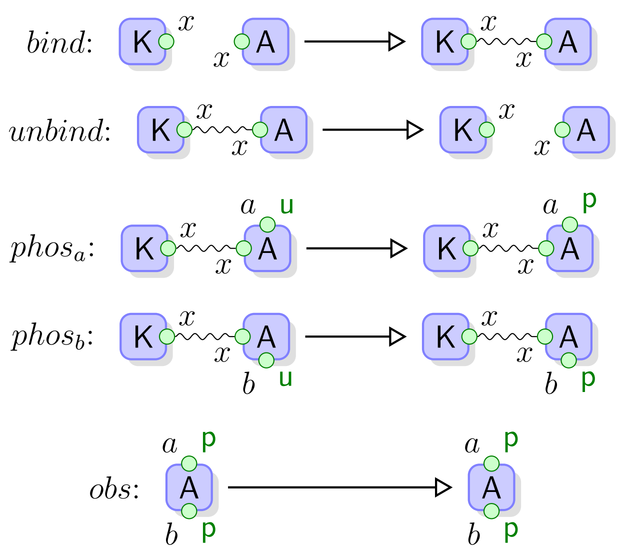
6.1 An example: processive phosphorylation

We now give a more sophisticated example of filtered compression, showing how it can be used to derive *quasi-processive* accounts of the phosphorylation of sites [1,16]. There are two mechanisms, *processive* and *distributive*, by which a number of sites on a single protein can become phosphorylated. Processive phosphorylation occurs when the sites become phosphorylated through the action of a single kinase. Distributive phosphorylation occurs when possibly different kinase molecules each bind, phosphorylate a single site and then unbind. As described in [1], it is appropriate under certain circumstances to view a distributive phosphorylation events as being processive, generating a quasi-processive account. For the production of causal accounts of the production of observables, this will typically be when the concentration of kinase molecules is high and we do not wish to track their identity in a pathway.

A simple model involves agents of type A with sites a, b and x and agents of type K with site x. The rules and contact graph for the system being



(a) Contact graph



(b) Rules. Domain of definition in each case is the greatest sub-graph of both the source and target drawn.

Fig. 4. Contact graph and rules for the phosphorylation of two sites on a single agent.

considered are presented in Figure 4. The observable, represented as a rule *obs*, tests that both sites a and b carry p labels (are phosphorylated). An example trajectory is presented in Figure 5. We wish to compress this trajectory to give a processive account for the production of the observable on the right-hand A-agent. This will involve generating a trajectory where just one kinase acts to phosphorylate both site a and site b. Note that an account for the production of the observable on *any* agent, as would be given by strong compression, would be trivial since the observable is matched by the left-hand A-agent; this reveals the importance of being able to select which kinds of agent have their identity tracked at the level of compression specified by the filter.

Letting C be the contact graph in Figure 4, we define the inclusion $c : C_0 \to C$ in Figure 5. Two partial maps $\alpha, \alpha' : g \to h$ in $(\Sigma\mathbb{G}/C)_*$ are $\sim$-equivalent iff they are the same (in terms of both being in the domain of definition and corresponding where defined) on agents of type A. Under this equivalence, two applications of the compression lemma, Theorem 6.4, yield the required sequence of compressions presented in Figure 5.

7 Conclusion and related work

The single-pushout semantics [15] for Kappa was given in [5] along with an account of weak and strong compression. Typed reduction for a side-effect free form of Kappa is studied in [9], where it is shown how rewriting can be constrained to match type constraints. It is worth mentioning that types for the constrained language used in [9] can require the existence of links

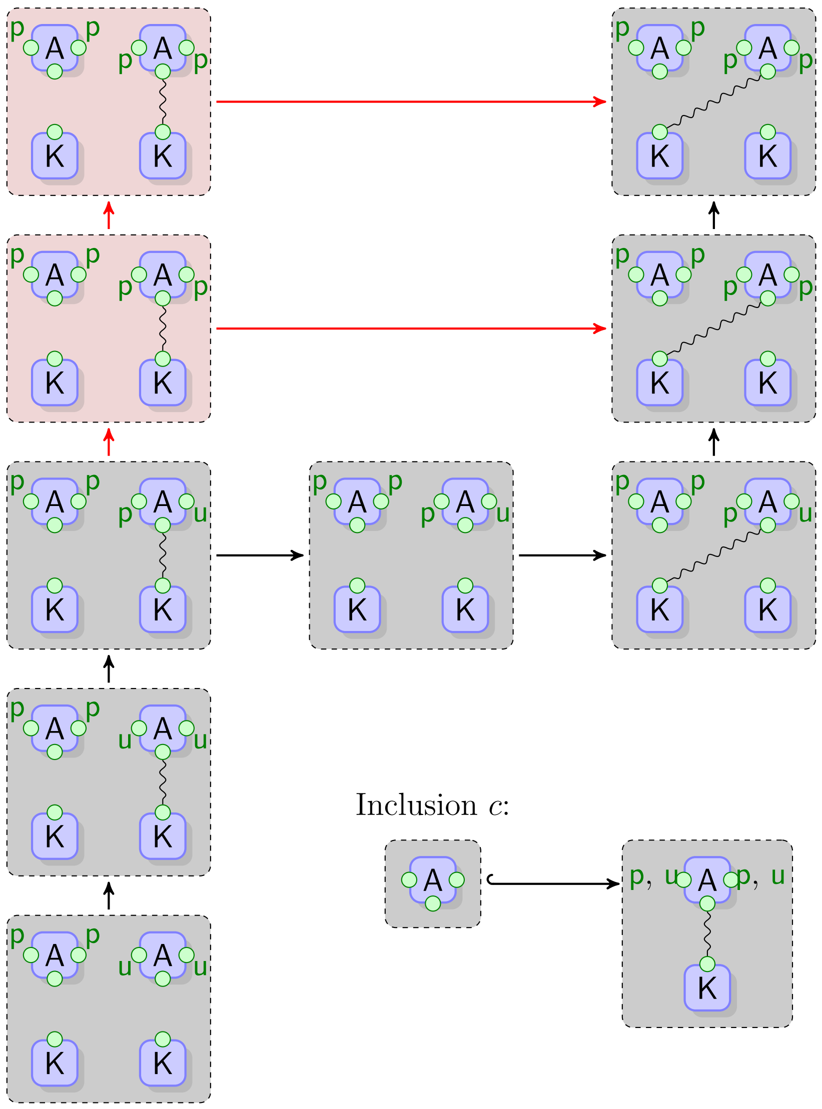


Fig. 5. Inclusion to define filtered equivalence and sequence of compression steps. Mixtures in the original trajectory have a grey background. Steps of compression highlighted with red background. On A-agents, the left site is a, the right site is b and the lower site is x. On K-agents, the site is x.

at given sites due to its explicit representation of unlinkedness, unlike the homomorphisms used here. For that reason, reduction in the slice category and in the untyped categories presented in [9] will not coincide.

The introduction of weak and strong compression here and in [5] is justified by comparison to Mazurkiewicz compression, a basic form of causal analysis based on the permutation of independent events. We plan in the future to give a fuller account of how more detailed models for concurrency compare to compression as presented here, for example with existing work on asymmetric conflict (e.g. [2]) and to establish connections with [14], which can be viewed as allowing compression steps when the residual is the composition of valid rule applications.

The framework for specifying levels of compression by an inclusion of the type graph presented here provides a more abstract understanding of weak compression and suggests forms of compression between weak and strong. The only property required specifically for Kappa was the coincidence of pushouts in the partial map and partial map slice categories. We believe that this result arises for much more abstract reasons and hope to show that this property holds for any (sub)category satisfying the Mipmap constraints presented in [13], which are necessary and sufficient constraints for the existence of pushouts of partial maps in categories where spans have cocones. Such a study would then allow the immediate translation of the results to other models.

More concretely, the implementation in KaSim of filtered compression is an obvious area for further work, as is the extension of compression and the production of causal flows to the tools for BioNetGen.

Finally, there are extensions of filtered compression that would be worth further investigation. These include the study of context-sensitive [4] inclusions to generate filters instead of using inclusion of the contact graph and forms of compression where it is possible to use alternative rules in compressed trajectories in a way specified by the user.

References

[1] K. Aoki, M. Yamada, K. Kunida, S. Yasuda, and M. Matsuda. Processive phosphorylation of erk map kinase in mammalian cells. *Proceedings of the National Academy of Sciences*, 2011.

[2] P. Baldan, A. Corradini, and U. Montanari. Contextual petri nets, asymmetric event structures, and processes. *Information and Computation*, 171(1):1–49, 2001.

[3] M. L. Blinov, J. Yang, J. R. Faeder, and W. S. Hlavacek. Graph theory for rule-based modeling of biochemical networks. In *Proc. CSB*, number 4230 in LNCS, 2006.

[4] F. Camporesi, J. Feret, and J. Hayman. Context-sensitive flow analyses: a hierarchy of model reductions. Submitted.

[5] V. Danos, J. Feret, W. Fontana, R. Harmer, J. Hayman, J. Krivine, C. Thompson-Walsh, and G. Winskel. Graphs, rewriting and pathway reconstruction for rule-based models. In *Proc. FSTTCS*, volume 18 of *LIPICS*, 2012.

[6] V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. Abstracting the differential semantics of rule-based models: exact and automated model reduction. In *Proc. LICS*, 2010.

[7] V. Danos, J. Feret, W. Fontana, and J. Krivine. Scalable simulation of cellular signaling networks. In *Proc. APLAS*, 2007.

[8] V. Danos, J. Feret, W. Fontana, and J. Krivine. Abstract interpretation of cellular signalling networks. In *Proc. VMCAI*, 2008.

[9] V. Danos, R. Harmer, and G. Winskel. Constraining rule-based dynamics with types. *MSCS*, 2012.

[10] V. Danos and C. Laneve. Formal molecular biology. *TCS*, 325, 2004.

[11] H. Ehrig and M. Löwe. Categorial principles, techniques and results for high-level-replacement systems in computer science. *Applied Categorical Structures*, 1(1), 1993.

[12] A. Habel, R. Heckel, and G. Taentzer. Graph grammars with negative application conditions. *Fundamenta Informaticae*, 26(3/4), 1996.

[13] J. Hayman and T. Heindel. Pattern graphs and graph rewriting: the categorical framework. Submitted.

[14] F. Hermann. Permutation equivalence of DPO derivations with negative application conditions based on subobject transformation systems. *ECEASST*, 16, 2008.

[15] M. Löwe. Algebraic approach to single-pushout graph transformation. *TCS*, 109, 1993.

[16] P. Patwardhan and W. T. Miller. Processive phosphorylation: mechanism and biological importance. *Cellular signalling*, 19(11), 2007.

[17] E. Robinson and G. Rosolini. Categories of partial maps. *Information and Computation*, 79(2), 1988.

A Proofs

We give proofs of the key results in this appendix.
Theorem 6.1

Proof. A simple consequence of the cofreeness property of the counit of the adjunction. Note that c^* is full (and of course faithful) since c is mono. □

Lemma 6.3

Proof. It is easy to verify that $\overrightarrow{\epsilon_h} \circ \epsilon_h = \mathsf{id}_{c^*(H)}$ from the definition of composition and the fact that ϵ_h is in $\mathcal{I}$ and therefore mono. From the definition of $c^*(\alpha)$, we have

$$\epsilon_h \circ c^*(\alpha) = \alpha \circ \epsilon_g$$

and therefore

$$\overrightarrow{\epsilon_h} \circ \epsilon_h \circ c^*(\alpha) = c^*(\alpha) = \overrightarrow{\epsilon_h} \circ \alpha \circ \epsilon_g.$$

Now consider the compositions $\alpha \circ \epsilon_g$ and $\overrightarrow{\epsilon_h} \circ \alpha$. In $\mathcal{C}$, we have:

$$\begin{tikzcd}
\mathrm{def}(\alpha \circ \epsilon_g) \arrow[rr, hook, "x"] \arrow[d, "j"'] \arrow[dr, phantom, very near start, "\lrcorner"] & & \mathrm{def}(\alpha) \arrow[dl, "i"'] \arrow[dr, "\alpha_0"] & & \mathrm{def}(\overrightarrow{\epsilon_h \circ \alpha}) \arrow[ll, hook', "y"'] \arrow[d, "k"] \arrow[dl, phantom, very near start, "\llcorner"] \\
c^*(G) \arrow[r, hook, "\epsilon_g"] \arrow[d, "c^*(g)"'] \arrow[dr, phantom, very near start, "\lrcorner"] & G \arrow[dr, "g"'] & = & H \arrow[dl, "h"] & c^*(H) \arrow[l, hook', "\epsilon_h"'] \arrow[d, "c^*(h)"] \arrow[dl, phantom, very near start, "\llcorner"] \\
C_0 \arrow[rr, hook, "c"'] & & C & & C_0 \arrow[ll, hook', "c"]
\end{tikzcd}$$

The inner square commutes because the morphism α is in $(\mathcal{C}/C)_*$ and the morphisms ϵ_g and x are both in $\mathcal{I}$ from the definitions of $c^*(g)$ and composition; similarly, ϵ_h and y are both in $\mathcal{I}$. Since pullbacks compose and are unique up to isomorphism, there exists an isomorphism $\phi : \mathrm{def}(\alpha \circ \epsilon_g) \to \mathrm{def}(\overrightarrow{\epsilon_h} \circ \alpha)$ such that $x = y \circ \phi$. From the final constraint in Definition 3.1, we have $\mathrm{def}(\overrightarrow{\epsilon_h} \circ \alpha) = \mathrm{def}(\alpha \circ \epsilon_g)$ and $x = y$.

By drawing the similar diagram for $\alpha' \circ \epsilon_g$ and $\overrightarrow{\epsilon_h} \circ \alpha'$, a straightforward analysis shows that $\overrightarrow{\epsilon_h} \circ \alpha = \overrightarrow{\epsilon_h} \circ \alpha'$ iff $\alpha \circ \epsilon_g = \alpha' \circ \epsilon_g$.

From the first part of the lemma, it is obvious that $\alpha \circ \epsilon_g = \alpha' \circ \epsilon_g$ implies $c^*(\alpha) = c^*(\alpha')$. Considering the reverse direction, the morphisms $c^*(\alpha)$ and $c^*(\alpha')$ are obtained as in the following two diagrams. Note that the upper pullback morphisms are identities since $x = y$ and x is mono.

$$\begin{tikzcd}
 & & & \mathrm{def}(\overrightarrow{\epsilon_h} \circ \alpha \circ \epsilon_g) \arrow[dl, "\mathsf{id}"'] \arrow[dr, "\mathsf{id}"] \arrow[d, phantom, "\vee"] & & & \\
 & & \mathrm{def}(\alpha \circ \epsilon_h) \arrow[dl, "j"'] \arrow[dr, "x"] \arrow[rr, leftrightarrow, "\mathsf{id}"] \arrow[d, phantom, "\vee"] & {} & \mathrm{def}(\overrightarrow{\epsilon_h} \circ \alpha) \arrow[dl, "y"'] \arrow[dr, "k"] \arrow[d, phantom, "\vee"] & & \\
 & c^*(G) \arrow[dl, "\mathsf{id}"'] \arrow[dr, "\epsilon_g"] & {} & \mathrm{def}(\alpha) \arrow[dl, "i"'] \arrow[dr, "\alpha_0"] & {} & c^*(H) \arrow[dl, "\epsilon_h"'] \arrow[dr, "\mathsf{id}"] & \\
c^*(G) & & G & & H & & c^*(H)
\end{tikzcd}$$

$$\begin{tikzcd}
 & & & \mathrm{def}(\overrightarrow{\epsilon_h} \circ \alpha' \circ \epsilon_g) \arrow[dl, "\mathsf{id}"'] \arrow[dr, "\mathsf{id}"] \arrow[d, phantom, "\vee"] & & & \\
 & & \mathrm{def}(\alpha' \circ \epsilon_h) \arrow[dl, "j'"'] \arrow[dr, "x'"] \arrow[rr, leftrightarrow, "\mathsf{id}"] \arrow[d, phantom, "\vee"] & {} & \mathrm{def}(\overrightarrow{\epsilon_h} \circ \alpha') \arrow[dl, "y'"'] \arrow[dr, "k'"] \arrow[d, phantom, "\vee"] & & \\
 & c^*(G) \arrow[dl, "\mathsf{id}"'] \arrow[dr, "\epsilon_g"] & {} & \mathrm{def}(\alpha') \arrow[dl, "i'"'] \arrow[dr, "\alpha'_0"] & {} & c^*(H) \arrow[dl, "\epsilon_h"'] \arrow[dr, "\mathsf{id}"] & \\
c^*(G) & & G & & H & & c^*(H)
\end{tikzcd}$$

It is easy to see from these diagrams that if $c^*(\alpha) = c^*(\alpha')$ then $\epsilon_g \circ \alpha = \epsilon_g \circ \alpha'$, *i.e.* that $j = j'$ and $k = k'$ implies $j = j'$ and $\alpha_0 \circ x = \alpha'_0 \circ x'$. □

Theorem 6.4

Proof. We commence by constructing a morphism $\gamma_0 : m'_0 \to c^*(m_2)$ using the universal property of the left-hand pushout in the following way. Since $\beta_0 \circ e_0 \sim e_1$, we have $c^*(\beta_0 \circ e_0) = c^*(e_1)$. Post-composing with $c^*(\beta_1)$ and noting that the right-hand square commutes, we obtain:

$$c^*(\beta_1 \circ \beta_0 \circ e_0) = c^*(\beta_1) \circ c^*(\beta_0 \circ e_0) = c^*(\beta_1) \circ c^*(e_1) = c^*(\beta_1 \circ e_1) = c^*(n_1 \circ \alpha)$$

By Lemma 6.3, it follows that $\overrightarrow{\epsilon_{m_2}} \circ \beta_1 \circ \beta_0 \circ e_0 = \overrightarrow{\epsilon_{m_2}} \circ n_1 \circ \alpha$. Since the left-hand square is a pushout, there is a unique morphism $\gamma_0 : m'_0 \to c^*(m_2)$ in $(\mathcal{C}/C)_*$ such that $\gamma_0 \beta'_0 = \overrightarrow{\epsilon_{m_2}} \beta_1 \beta_0$ (1) and $\overrightarrow{\epsilon_{m_2}} n_1 = \gamma_0 n_0$ (2).

Defining $\gamma = \epsilon_{m_2} \circ \gamma_0$, a straightforward analysis shows that $\gamma n_0 \sim n_1$ and $\gamma \circ \beta'_0 \sim \beta_1 \circ \beta_0$. For the uniqueness-up-to-$\sim$ property, let $\gamma' : m'_0 \to m'_2$ be any partial map in $(\mathcal{C}/C)_*$ such that $\gamma' n_0 \sim n_1$ and $\gamma' \beta'_0 \sim \beta_1 \beta_0$. Define $\gamma'_0 = \overrightarrow{\epsilon_{m_2}} \gamma'$. It is straightforward to verify that $\gamma'_0 = \gamma_0$ from the uniqueness of γ_0. From the definition of γ and the fact that $\overrightarrow{\epsilon_{m_2}} \epsilon_{m_2} = \mathsf{id}_{c^*(m_2)}$, we have $\overrightarrow{\epsilon_{m_2}} \gamma = \overrightarrow{\epsilon_{m_2}} \epsilon_{m_2} \gamma_0 = \gamma_0 = \gamma'_0 = \overrightarrow{\epsilon_{m_2}} \gamma'$. Hence $\gamma \sim \gamma'$, as required. □