Bachelor's Thesis Computing Science

Radboud University Nijmegen

Extending String Diagrams of MDPs

Author:
Wessel van der Lans
s1084461

First assessor:
dr. Sebastian Junges

Second assessor:
dr. Jurriaan Rot

Weekly Supervisor:
Marck van der Vegt

January 21, 2025

Abstract

Using String Diagrams as a means of modelling MDPs is a relatively new concept. Currently, not a lot of tools exist that make it easy for researchers or other users to work with String Diagrams of MDPs. We introduce a UI which allows users to visualize, edit and export String Diagrams of MDPs. Additionally, we propose two extensions to these String Diagrams. These extensions increase the expressivity of String Diagrams and they make it possible to represent the String Diagrams more compactly. We demonstrate how these extensions work using our UI. Finally, we prove that Extended String Diagrams are at least as expressive as Basic String Diagrams, by showing that in any case an Extended String Diagram can be converted to a Basic String Diagram.

Contents

Chapter 1

Introduction

A well-known example of a decision problem of sequential decision making is that of a robot traveling between so-called "rooms". The robot starts in some initial room from which it can reach a subset of other rooms.

Example 1. A very simple example is displayed in Figure 1.1. In this example, the robot is a cleaning robot that can clean floors. The robot starts in the "Storage" state. Based upon some condition the robot moves to either the Living Room or the Kitchen. After cleaning the floor in the room it reached, the robot returns to the Storage.

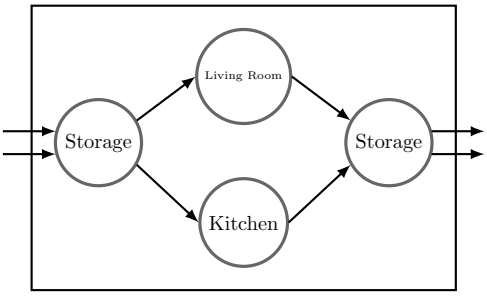

Figure 1.1: Robot traveling between rooms

For more complicated models, it could be interesting to determine how the robot can achieve the highest possible reward, in other cases it is more interesting to calculate the reachability probability of a certain room. A common way to model a problem like this is by using a Markov Decision Process (MDP). An MDP is a type of stochastic sequential decision process in which the cost and transition functions depend only on the current state of the system and the current action [6].

In practice, models tend to be a lot more complicated than that of the cleaning robot. Complicated real-world models may contain hundreds of millions of states. In these more complicated models, it is often infeasible

to calculate the reachability probabilities over a whole model. This can be because the model no longer fits in memory, or because the algorithms used for the calculations do not scale well enough for inputs this large. However, in a lot of cases it is possible to calculate the reachability probabilities when the model is partitioned into multiple subMDPs.

A String Diagram of MDPs is a type of model that consists of multiple such subMDPs. In String Diagrams these subMDPs are combined through sequential composition and by summing. Sequential composition can be seen as combining two subMDPs horizontally, by connecting exits of one subMDP with entrances of another subMDP. Summing can be seen as combining two subMDPs vertically, with the resulting subMDP's entrances/exits consisting of the combination of the entrances/exits of the two original subMDPs. A large MDP's structure can be captured by repeatedly sequentially composing and/or summing its subMDPs. In the next chapter, we formally define sequential composition and summing.

Example 2 (Sequential composition & Sum). We continue with our earlier example. In this example, each room can be seen as a subMDP. Sequential composition can be visualized as displayed in Figure 1.2. Summing can be visualized as displayed in Figure 1.3.

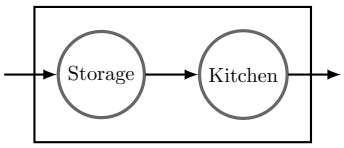

Figure 1.2: Sequential Composition

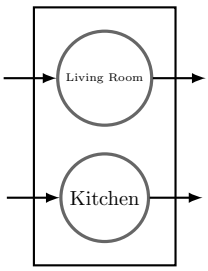

Figure 1.3: Sum

String Diagrams can be very useful for computing best-case probabilities of reaching a goal state for a given MDP, since they make it possible to perform the necessary calculations on just the components of the String Diagram as opposed to the entire state space [9]. Additionally, String Diagrams make it

possible to visualize models with a huge amount of states in a compact way. Large MDPs often have a lot of structure, and these structures can often be captured well by String Diagrams. A compact visualization can make it much easier for users to work with these large models.

Using String Diagrams to represent MDPs is a relatively new concept [8], and not a lot of tools currently exist to make String Diagrams of MDPs easy to work with. One of the tools that does exist is the probabilistic model checker Storm [2]. Storm-compose [8] [9] is a branch of Storm that supports the analysis of String Diagrams as we use them.

1.1 Problem Description

The current format of String Diagrams is limited. Additionally, no tool exists that allows a user to visualize and/or edit String Diagrams.

In this thesis we look at the following research question:

- How can we extend String Diagrams of MDPs?

In order to answer this question, we explore the following sub-questions:

1. What types of new information do we want to store using Extended String Diagrams?
2. How can Extended String Diagrams be converted into Basic String Diagrams without losing information?

We created a user interface that allows the user to import a String Diagram and visualize it. The user has the option to edit the diagram, and the UI allows the user to export the edited diagram once it is finished. The user interface will make it a lot easier for researchers or other users to be able to work with and analyze String Diagrams.

On top of this, we introduce two extensions to String Diagrams. The *repeat* extension allows the user to easily repeat any subMDP in our UI. Additionally, the repeat extension allows the JSON file that represents a diagram with repeated subMDPs to be more compact than a JSON file of the same diagram without the extension.

The *switch* extension allows the user to manually define how the sub-MDPs of a String Diagram are connected.

Example 3 (Switch). Figure 1.4 shows how Figure 1.1 can be edited using the switch-extension. For example, let's say the first exit of the initial storage state is chosen when it is morning, and the second exit is chosen when it is evening. If the user wanted to change the behavior of the model without having to edit the internal subMDPs or the order of the subMDPs, he can switch the destination of the outgoing transitions of the first storage state such that now the kitchen will be cleaned when it is morning, and the living room will be cleaned when it is evening.

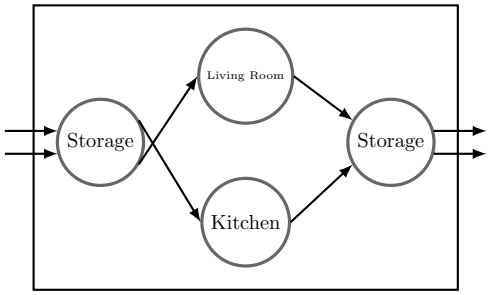

Figure 1.4: The switch-extension

We refer to String Diagrams that do not use these extensions as Basic String Diagrams (BSDs), and we refer to String Diagrams that do use these extensions as Extended String Diagrams (ESDs). We demonstrate the appicability of these extensions using our UI.

Although ESDs are more user-friendly and more compact than BSDs, they cannot be used as input for Storm. We therefore show that, if needed, ESDs can always be converted back to BSDs without losing information.

1.2 Related Work

String Diagrams have existed since the 1970's [4]. Since then, they have mostly been used in category theory [7]. As such, String Diagrams have found many applications both in and outside the Computing Science field. For example, String Diagrams have been used in Natural Language Processing [5], and as a way to represent concurrent systems [1].

String Diagrams as a representation of MDPs have been introduced in 2023 as a means of compositional probabilistic model checking [8]. This is very recent, and as a result there is not a very large amount of other related work on this subject. An even more recent paper that makes use of String Diagrams as a representation of MDPs is [9], where the authors use the compositionality of String Diagrams to accurately approximate reachability probabilities of sub-MDPs of the diagram. They then combine these intermediate results to eventually approximate the reachability probabilities of an entire String Diagram.

1.3 Outline

- Chapter 2 provides background information. In this chapter we define String Diagrams and their semantics, and we define some equivalence relations that are relevant for the operations on String Diagrams.
- Chapter 3 builds on Chapter 2 by defining our extensions of String Diagrams and the semantics of the resulting Extended String Diagrams. We additionally show how these Extended String Diagrams can be converted to Basic String Diagrams.
- In Chapter 4 we discuss everything related to the implementation of our UI. We show how String Diagrams are represented in JSON-format, and go over all the functionality we implemented.
- Chapters 5 and 6 wrap up the thesis, containing the related work and conclusions respectively.

Chapter 2

Preliminaries

String Diagrams originate from category theory. However, because this thesis is concerned with String Diagrams of MDPs, no knowledge of category theory is required. From here on out, we refer to String Diagrams of MDPs as simply String Diagrams for brevity.

In this chapter we explain what String Diagrams are, closely following the way they are defined by Watanabe et al. [9].

String Diagrams consist of one or more open Markov Decision Processes. We therefore introduce these oMDPs and define their semantics.

2.1 Notation

We go over some notation that will be used throughout this thesis.

- $\mathcal{D}(X)$: the set of distributions on X. A set of distributions refers to all possible ways to assign probabilities to each element in X such that these probabilities add up to 1.
- $[m]$: the sequence of numbers $\{1, 2, ..., m\}$
- $\uplus$: disjoint union. Let A and B be sets. The disjoint union $A \uplus B$ of A and B is the set made up from the elements of A and B in which each element is labeled with the name of the set from which it comes. In some cases we leave the labeling implicit when there is no risk of ambiguity.
  For two functions $f : x \to y$ and $g : a \to b$ the disjoint union is defined as:
  $$(f \uplus g)(k) = \begin{cases} y & \text{if } k \in x \\ b & \text{if } k \in a \end{cases}$$
- δ_c: a function which returns 1 when the condition c is true, and 0 otherwise

2.2 Markov Decision Processes

Definition 1 (open MDP)**.** *An open MDP (oMDP)* $\mathcal{A} = (S, A, P, E)$ *is a tuple with a finite set* S *of states, finite set* A *of actions, a partial transition function* P*:* $S \times A \rightharpoonup \mathcal{D}(S) + [n_r + m_l]$*, and an entry function* E*:* $[m_r + n_l] \to S + [n_r + m_l]$*, which maps each entrance in* $[m_r + n_l]$ *to either a state in* S *or to an exit in* $[n_r + m_l]$*.*

In the definition above, $m = (m_r, m_l)$ and $n = (n_r, n_l)$ refer to the oMDP's left- and right-arity. m and n are both pairs of natural numbers. Elements of $[m_r + n_l]$ are the oMDP's entrances, and elements of $[n_r + m_l]$ are its exits.

Example 4 (Left- and right-arity). Figure 2.1 has two right-facing entrances which are represented by m_r, one left-facing exit which is represented by m_l, one right-facing exit which is represented by n_r, and three left-facing entrances which are represented by n_l. The example's left-arity $m = (2, 1)$ and the example's right-arity $n = (1, 3)$.

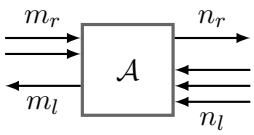

Figure 2.1: Left- and right-arity of an oMDP

The possible actions in a state s are A(s) $= \{a \in A \mid P(s, a) \neq \bot\}$. A *terminal state* is a state s for which A(s) $= \emptyset$.
We think of the distribution of states $P(s_0, a_0)$ as a function from states to probabilities. In the rest of this paper we use the notation $P(s_0, a_0, s_1)$ instead of $P(s_0, a_0)(s_1)$.

Example 5. An example oMDP is shown in Figure 2.2. In this figure:
$S = \{s_0, s_1, s_2\}$, $A = \{a_0\}$,
$P(s_0, a_0, s_2) = 0.8$, $P(s_0, a_0, s_1) = 0.2$, $P(s_1, a_0, s_2) = 1$,
$E(0) = s_0$, and $E(1) = s_1$

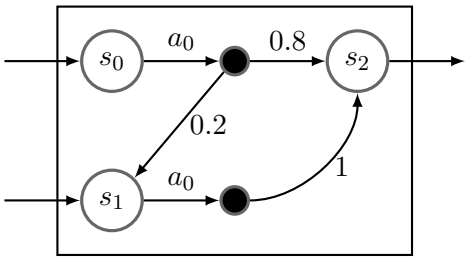

Figure 2.2: Example open MDP

Definition 2 (Path)**.** *A path* $\pi : s_0 \to a_0 \to s_1 \to a_1 \ldots$ *is an (in)finite sequence of alternating states and actions. For every index* i *it holds that* $s_i \in S$ *and* $P(s_i, a_i, s_{i+1}) \neq 0$*. Paths can be finite or infinite. The set of all finite paths is written as* $FPath_M$ *and the set of all infinite paths is written as* $IPath_M$*.*

Definition 3 (Markov Chain)**.** *A Markov Chain is an MDP for which* $|A(S)| \leq 1$ *for all* $s \in S$*.*

Since a Markov Chain is a special type of MDP, we can define a Markov Chain as an MDP by using a tuple $(S, \{\bot\}, P)$. If we want to use a Markov Chain an sich we can simply define it using a tuple (S, P).

2.3 String Diagrams

Now, let's consider MDPs which consist of a very large amount of states. For models like this it is often computationally infeasible to solve a problem by running the desired calculations over the entire model [9]. However, in many cases it is possible to solve such a problem when the model is partitioned into oMDPs. By using a divide-and-conquer approach, the desired calculations can be run on the oMDPs individually, which requires far less computational power. The results of the calculations on the oMDPs can then be combined to calculate the final result of the entire model.
When visualizing an oMDP, we only show its entrances and exits. This is because oMDPs become most useful when they make up a model that contains a very large amount of states, so showing all the internal states in the visualization would quickly become very unclear.

Example 6. Figure 2.3 shows how an oMDP is visualized when it is part of a String Diagram. It is important to realize that the oMDP may contain internal states even though they are not visualized. The whole aim of String Diagrams is to make it easier to work with huge models, so in a lot of practical settings, these oMDPs may contain a very large amount of internal states.

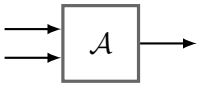

Figure 2.3: oMDP as part of a String Diagram

Definition 4 (String Diagram)**.** *A String Diagram $\mathcal{D}$ of oMDPs is a term adhering to the grammar*

$$\mathbb{D} := \mathcal{A} \mid \mathbb{D} \fatsemi \mathbb{D} \mid \mathbb{D} \oplus \mathbb{D}$$

where $\mathcal{A}$ can be any oMDP.

Intuitively, a String Diagram is a model that is entirely composed of oMDPs. For any oMDP in the model, its input states are connected to the output states of an oMDP in the previous layer, and its output states are connected to the input states of an oMDP in the next layer.

oMDPs like the one in Figure 2.3 can be combined in two ways: through sequential composition and by summing them.

2.3.1 Sequential Composition

The first operation we will look at is *sequential composition*. Sequential composition of two oMDPs $\mathcal{A}$ and $\mathcal{B}$ is depicted as follows:

$$\mathcal{A} \fatsemi \mathcal{B}$$

Definition 5 ($\fatsemi$ operator)**.** *Let $\mathcal{A}, \mathcal{B}$ be oMDPs, with matching arities and the same action set A. Applying sequential composition results in an oMDP (S, A, P, E), where: $S := S^{\mathcal{A}} \uplus S^{\mathcal{B}}$, $A := A$, $P := P^{\mathcal{A} \fatsemi \mathcal{B}}$, $E = E^{\mathcal{A} \fatsemi \mathcal{B}}$ and:*

$E^{\mathcal{A} \fatsemi \mathcal{B}}(i) = E^{\mathcal{A}}(i)$ if $E^{\mathcal{A}}(i) \in S^{\mathcal{A}}$ and $E^{\mathcal{A} \fatsemi \mathcal{B}}(i) = E^{\mathcal{B}}(E^{\mathcal{A}}(i))$ if $E^{\mathcal{A}}(i) \in [n_r]^{\mathcal{A}}$, and:

$$P^{\mathcal{A} \fatsemi \mathcal{B}}(s^{\mathcal{A}}, a, s') = \begin{cases} P^{\mathcal{A}}(s^{\mathcal{A}}, a, s') & \text{if s' } \in S^{\mathcal{A}} \\ \sum_{i \in [k]} P^{\mathcal{A}}(s^{\mathcal{A}}, a, i) \cdot \delta_{E^{\mathcal{B}}(i) = s'} & \text{otherwise} \end{cases}$$

$$P^{\mathcal{A} \fatsemi \mathcal{B}}(s^{\mathcal{B}}, a, s') = \begin{cases} P^{\mathcal{B}}(s^{\mathcal{B}}, a, s') & \text{if s' } \in S^{\mathcal{B}} + [n_r + m_l]^{\mathcal{B}} \\ 0 & \text{otherwise} \end{cases}$$

In this definition, k is any entrance in $\mathcal{B}$ that is reached from $\mathcal{A}$, and $[n_r + m_l]$ is any exit originally in $\mathcal{B}$.

Example 7 (Entry function combined with with sequential composition)*.* Figure 2.4 shows sequentially composed oMDPs $\mathcal{A}$ and $\mathcal{B}$. In this example:
$E^{\mathcal{A} \fatsemi \mathcal{B}}(0) = E^{\mathcal{A}}(0) = s_0^{\mathcal{A}}$, since $E^{\mathcal{A}}(0) \in S^{\mathcal{A}}$.
$E^{\mathcal{A} \fatsemi \mathcal{B}}(1) = E^{\mathcal{B}}(E^{\mathcal{A}}(1)) = E^{\mathcal{B}}(1) = s_0^{\mathcal{B}}$, since $E^{\mathcal{A}}(1) \in [n_r]^{\mathcal{A}}$.

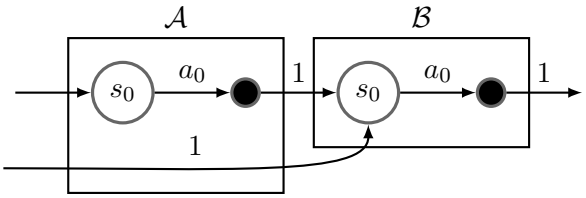

Figure 2.4: Sequential Composition example 1

Example 8 (Transition probabilities in sequential composition). Figure 2.5 shows sequentially composed oMDPs $\mathcal{A}$ and $\mathcal{B}$. In this example:
$P^{\mathcal{A};\mathcal{B}}(s_0^{\mathcal{A}}, a_0, s_1) = P^{\mathcal{A}}(s_0^{\mathcal{A}}, a_0, s_1) = 0.2$, since $s_1 \in S^{\mathcal{A}}$.
$P^{\mathcal{A};\mathcal{B}}(s_1, a_0, s_0^{\mathcal{B}}) = P^{\mathcal{A}}(s_1, a_0, s_0^{\mathcal{B}}) = 1$, since $E^{\mathcal{B}}(1) = s_0^{\mathcal{B}}$.
$P^{\mathcal{A};\mathcal{B}}(s_0^{\mathcal{B}}, a_0, 0^{\mathcal{B}}) = P^{\mathcal{B}}(s_0^{\mathcal{B}}, a_0, 0^{\mathcal{B}}) = 1$, since $0^{\mathcal{B}} \in [n_r]^{\mathcal{B}}$.

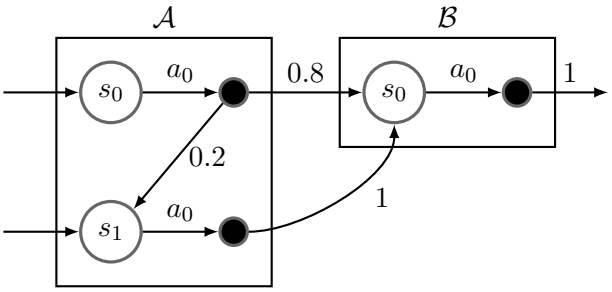

Figure 2.5: Sequential Composition example 2

2.3.2 Sum

The second operation on oMDPs is the sum operation. Summing oMDPs $\mathcal{A}$ and $\mathcal{B}$ is written as:

$$\mathcal{A} \oplus \mathcal{B}$$

Definition 6 ($\oplus$ operator)**.** *Let $\mathcal{A}, \mathcal{B}$ be oMDPs with the same action set A. Their sum $\mathcal{A} \oplus \mathcal{B}$ is an oMDP (S, A, P, E), where:*
$S := S^{\mathcal{A}} \uplus S^{\mathcal{B}}$, $A := A$, $E = E^{\mathcal{A}} \uplus E^{\mathcal{B}}$, *and P is defined as:*

$$P(s,a,s') := \begin{cases} P^{\mathcal{D}}(s,a,s') & \text{if } \mathcal{D} \in \{\mathcal{A},\mathcal{B}\}, s \in S^{\mathcal{D}}, a \in A^{\mathcal{D}}, s' \in S^{\mathcal{D}}, \\ 0 & \text{otherwise.} \end{cases}$$

In this definition, E' is the disjoint union of the entry functions of $\mathcal{A}$ and $\mathcal{B}$. The disjoint union of two functions has been defined in Section 2.1.

Example 9. Summing two oMDPs results in one larger oMDP. Visually, this means the oMDPs are stacked on top of each other. The sum's entrances and exits consist of the combination of the entrances and exits of the summed oMDPs. An example of two summed oMDPs is displayed in Figure 2.6.

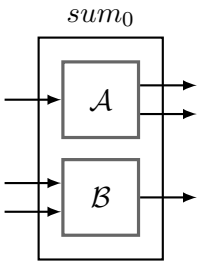

Figure 2.6: Sum of two oMDPs

A general term for an oMDP or a sum that is part of a sequence is a layer. The reason why sums are so important is that it often happens that the number of entrances and exits of adjacent layers do not match initially. In such cases summing is necessary in order to be able to match the exit states of one layer to the entrance states of the next layer. Figure 2.7 demonstrates this by showing a full String Diagram containing multiple layers which are combined through both sequential composition and sums.

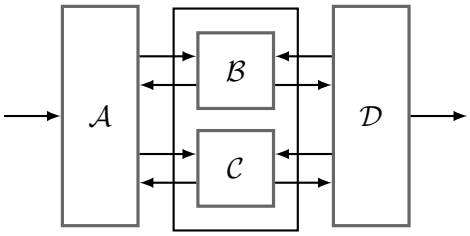

Figure 2.7: String Diagram constructed using both sequential composition and sums

2.4 Equivalence Relations

In this section we define some equivalence relations that are relevant for sequential composition and summing.

2.4.1 Identity

Definition 7 (Identity)**.** *Let m and n be natural numbers. The identity $\mathcal{I}_m$ on m (over the action set A) is an oMDP defined as $\mathcal{I}_m = (\emptyset, A, !, E)$, where $E(i) = i$ for each $i \in [m]$.*

The "!" denotes that the transition probabilities for this oMDP are undefined/not relevant. This is the case because the identity does not contain any states, so the domain of P is empty.

Theorem 1 (Identity)*.*

$$\mathcal{I} \fatsemi \mathcal{A} = \mathcal{A} = \mathcal{A} \fatsemi \mathcal{I}$$

Proof. See Appendix A.1.1 □

Since the identity component does not have any states, it effectively extends the outgoing arrows of a component. This is useful when creating or visualizing a String Diagram, because it allows the user to change the length of arrows.

2.4.2 Associativity

Theorem 2 (Associativity)*.*

$$\mathcal{A} \fatsemi (\mathcal{B} \fatsemi \mathcal{C}) = (\mathcal{A} \fatsemi \mathcal{B}) \fatsemi \mathcal{C} \tag{2.1}$$

$$\mathcal{A} \oplus (\mathcal{B} \oplus \mathcal{C}) = (\mathcal{A} \oplus \mathcal{B}) \oplus \mathcal{C} \tag{2.2}$$

Proof. See Appendix A.1.2 and Appendix A.1.3 □

The associativity relation resolves any ambiguity that might occur when parsing a file that represents a String Diagram. How this works will be discussed in Section 4.2.1.

2.4.3 Bifunctoriality

Theorem 3 (Bifunctoriality).

$$(\mathcal{A} \oplus \mathcal{B}) \fatsemi (\mathcal{C} \oplus \mathcal{D}) = (\mathcal{A} \fatsemi \mathcal{C}) \oplus (\mathcal{B} \fatsemi \mathcal{D})$$

Proof. See Appendix A.1.4 □

The bifunctoriality relation shows that a String Diagram consisting of two sequentially composed layers of two summed oMDPs can be created in two different ways, which both lead to the same result. This is demonstrated in Figure 2.8. In the first approach, the top-left oMDP is summed with the bottom-left oMDP, and the top-right oMDP is summed with the bottom-right oMDP. Sequentially composing these layers leads to the same String Diagram that is the result of the second approach. In the second approach, the top-left oMDP is first sequentially composed with the top-right oMDP, and the bottom-left oMDP is first sequentially composed with the bottom-right oMDP. Summing these top and bottom oMDPs results in the same String Diagram as the one resulting from the first approach.

The bifunctoriality relation is important in regards to our UI. In our UI it is not possible to sum an entire sequence with another sequence at once. However, knowing that the bifunctoriality relation holds, the user can sum the oMDPs of the first sequence individually with each oMDP of the second sequence. The bifunctoriality relation guarantees that despite the different order of operations the resulting diagram will be the same.

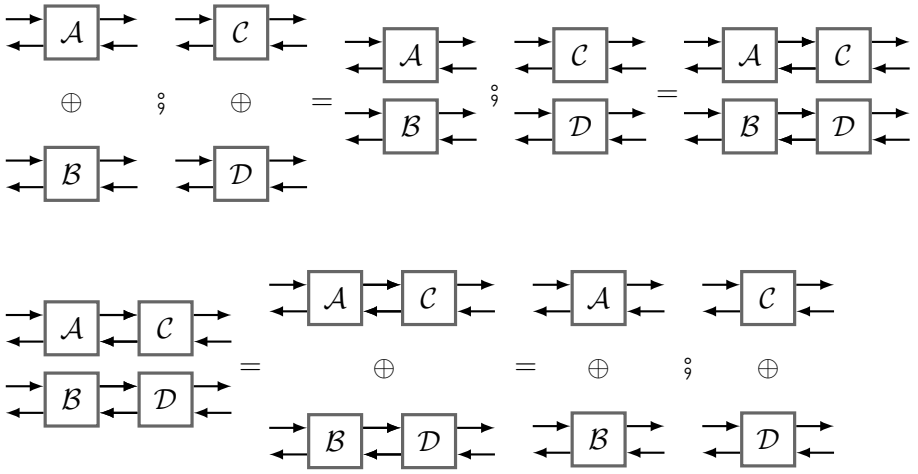

Figure 2.8: The bifunctoriality relation

Chapter 3

Extended String Diagrams

In this chapter we introduce several extensions that can be made to Basic String Diagrams (BSDs) in order to make it easier for users to work with String Diagrams.

Definition 8 (Extended String Diagram)**.** *An Extended String Diagram (ESD)* $\mathbb{D}$ *of oMDPs is a term adhering to the grammar*

$$\mathbb{D} := [m_1]\mathcal{A}[m_2] \mid \mathbb{D} \fatsemi \mathbb{D} \mid \mathbb{D} \oplus \mathbb{D} \mid \mathcal{R}(\mathbb{D}, n)$$

where $\mathcal{A}$ *can be any oMDP and* n *is the number of times the component is repeated. Left- and right-mapping* m_1 *and* m_2 *(formally defined in Section 3.2) are mappings between the exit states of* $\mathcal{A}$ *and the entrance states of the layer before and after* $\mathcal{A}$*.*

An ESD is different from a BSD in two ways. ESDs include the *repeat* component $\mathcal{R}(\mathbb{D}, n)$ and the *switch* operation. These extensions improve usability and allow for a more compact representation of String Diagrams. Using these extensions does not decrease the expressivity of the diagrams, since it is always possible to convert an ESD back to a BSD. In the following sections we formally define the extensions and their semantics.

3.1 Repeat

In some cases a user may want to sequentially compose the same component multiple times in a row. A real-world example where this could happen is a model of a network protocol with multiple rounds. In order to repeat a layer in the basic format its name must be manually added to the file representing the String Diagram for every repetition. This might not be a problem if the layer is only repeated a few times. However, repeating the layer many times using this method causes problems. For example, it becomes inconvenient to determine the amount of repetitions. Additionally, a large amount of repetitions may clutter the file, making the file unorganized and less convenient to edit.

In order to make it easier to work with repeated layers, we propose the *repeat* extension.

Definition 9 (Repeat)**.** *Let $\mathcal{A}$ be an oMDP. The repeat operation $\mathcal{R}(\mathcal{A}, n)$, where $n \geq 1$, is defined as:*

$$\mathcal{R}(\mathcal{A}, n) = \begin{cases} \mathcal{A} & \text{if } n = 1 \\ \mathcal{R}(\mathcal{A}, n-1) \fatsemi \mathcal{A} & \text{otherwise.} \end{cases}$$

Example 10 (Repeat). Let's say the cleaning robot from our example in the introduction needs to clean the kitchen floor three times throughout the day before returning to the storage. Without the repeat-extension, this example would be modeled as displayed in Figure 3.1. With the repeat-extension, however, the example can be visualized in a more compact way, as visualized in Figure 3.2.

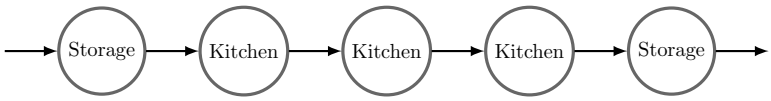

Figure 3.1: Example 10 without the repeat-extension

Introducing the repeat operation solves the problems we described earlier. It is now very easy to determine how often a layer is repeated. The file representing a String Diagram with repeat components will look a lot more organized than one without repeat components, and it will be a lot easier to edit layers around the repeated layers.

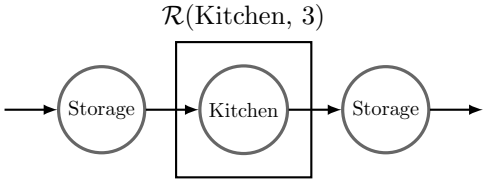

Figure 3.2: Example 10 with the repeat-extension

3.2 Switch

Basic String Diagrams do not explicitly define how the exit states of a layer are connected to the entrance states of the next layer. Instead, it is left implicit that the first exit state of the first layer is connected to the first entrance state of the second layer, that the second exit state of the first layer is connected to the second entrance state of the second layer, and so forth. While this sounds simple and efficient, it is problematic in cases where users would want to specify themselves how the states between layers should be connected.

The main problem is that in BSDs it is not possible in some cases to connect two exit states from one component in a layer to two entrance states in two separate components in the next layer. This is because components cannot overlap within a layer and states are assumed to be connected as described in the previous paragraph. At first glance, a solution for this could be to allow the user to reorder the entrance/exit states of a component and/or layer. Unfortunately, this simple solution is not possible, since components can occur multiple times in the diagram. Reordering the entrance/exit states of a component in the diagram's JSON file would reorder the entrance/exit states of all components with that name. This would change the behavior of the diagram beyond the user's intended changes, and is therefore not an acceptable solution.

To allow users to connect states between layers in any way they want, we introduce our second extension: the *switch* operation.

Definition 10 (Switch)**.** *Let $\mathcal{A}$ be an oMDP. Applying left mapping $[m_1]$ and right mapping $[m_2]$ to $\mathcal{A}$ results in an oMDP $[m_1]\mathcal{A}[m_2] = (S, A, P, E)$, where:*
$S := S^{\mathcal{A}}$, $A := A^{\mathcal{A}}$, $P := P^{\mathcal{A}}$, $E = E_1 \uplus E_2$,
$E_1 = m_1(E^{\mathcal{A}}(i)), E_2 = m_2(E^{\mathcal{A}}(i))$,
$m_1 : [m_l]^{\mathcal{A}} \to [m_l]^{\mathcal{A}}$ *is a mapping from each left-facing exit of $\mathcal{A}$ to a potentially new position, and:*
$m_2 : [n_r]^{\mathcal{A}} \to [n_r]^{\mathcal{A}}$ *is a mapping from each right-facing exit of $\mathcal{A}$ to a potentially new position.*

Example 11 (Switch). Figure 3.3 shows oMDPs $\mathcal{A}$ and $\mathcal{B}$. In this example, $\mathcal{A}$ is defined as:
$m_1 = []$, since $\mathcal{A}$ has no left-facing exits,
$m_2 = [1, 0]$
$S = \{s_0, s_1\}$, $A = a_0$, $P =!$
$E_1 = !$, since $\mathcal{A}$ has no left-facing entrances.
$E_2(0) = m_2(E^{\mathcal{A}}(0)) = m_2(s_0) = s_1^{\mathcal{B}}$
$E_2(1) = m_2(E^{\mathcal{A}}(1)) = m_2(s_1) = s_0^{\mathcal{B}}$

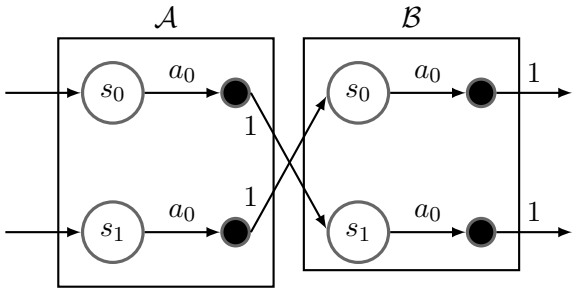

Figure 3.3: Switch example

By default, a mapping [m] is the identity mapping. From now on, we omit [m] whenever it is an identity mapping. When both $[m_1]$ and $[m_2]$ are the default mappings, $\mathcal{A}$ therefore behaves the same as if it were a BSD component. By using custom mappings the user no longer experiences the problems described earlier in this section.

3.3 Converting ESDs to BSDs

While Extended String Diagrams are convenient and more expressive than Basic String Diagrams, they cannot be used as input for Storm. It is especially important for a user to be able to check a model's properties after having edited said model. For this reason, this section discusses how ESDs can be converted to BSDs such that newly created or edited models can be used in the model checker.

3.3.1 Converting Repeat to BSD

Theorem 4. Let $\mathcal{A}$ be an oMDP. We write $\mathcal{A}^n$ when $\mathcal{A}$ is repeated n times without using the repeat component. For any natural number $n \geq 1$, it holds that:

$$\mathcal{A}^n = \mathcal{R}(\mathcal{A}, n)$$

Proof. See Appendix A.2.1 □

3.3.2 Converting Switch to BSD

Theorem 5. Let $[m_1]\mathcal{A}[m_2]$ be an oMDP with custom mappings m_1 and m_2. It holds that:

$$[m_1]\mathcal{A}[m_2] = S_1 \fatsemi \mathcal{A} \fatsemi S_2$$

$S_1 = (S, A, P, E_1)$ is an oMDP where:
$S := \emptyset, A := A^{\mathcal{A}}, P :=!, E_1(i) = m_1(E^{\mathcal{A}}(i))$, and
$S_2 = (S, A, P, E_2)$ is an oMDP where:
$S := \emptyset, A := A^{\mathcal{A}}, P :=!, E_2(i) = m_2(E^{\mathcal{A}}(i))$.

Proof. See Appendix A.2.2 □

Example 12. Figure 3.5 shows how oMDP $\mathcal{A}$ with mapping $m_2 = [1, 0]$ from Figure 3.4 is converted to the BSD format.

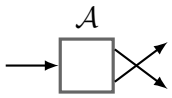

Figure 3.4: oMPD $\mathcal{A}$ with mapping $m_2 = [1, 0]$

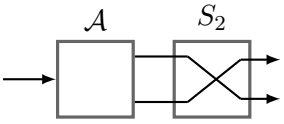

Figure 3.5: Converting switch to BSD

Chapter 4

Implementation

In this chapter we discuss everything related to the implementation of our UI. The UI code has been uploaded to `https://gitlab.science.ru.nl/thesis-supplemental-material/wessel-vanderlans`.
We start this chapter by explaining some of the practical decisions we made. Next, we show how both BSDs and ESDs are represented in JSON format. This is important because our tool imports and exports String Diagrams to and from this JSON format. In the following section, we go over all the functionality in our UI. The last section of this chapter is again about converting ESDs to BSDs, only this time our solutions to this problem are presented through a more practical perspective as opposed to the more theoretical perspective from the previous chapter.

4.1 Design Decisions

In this section we go over some practical decisions we made regarding the implementation of the user interface.

We chose to implement the UI in Python. We chose Python because this is the programming language we were most familiar and comfortable with. Within Python, we chose the GUI toolkit *tkinter* to develop our UI. Tkinter is relatively simple, which was a big advantage, as we had little UI programming experience. Nonetheless, tkinter has more than enough features to be able to create an effective UI.

4.2 String Diagrams in JSON Format

In this section we show what String Diagrams look like in JSON format when they are imported by our UI. We start in subsection one by describing the existing format of (Basic) String Diagrams in JSON. In the second subsection we show how our proposed extensions to String Diagrams can be included in this existing format.

4.2.1 BSDs in JSON Format

Basic String Diagrams in JSON-format must adhere to a specific format. The structure of this format is depicted in Figure 4.1.

```
{
    "root": "root",
    "components": {
        "layer_0": {
            (layer data)
        },

        "layer_1": {
            (layer data)
        },

        ...

        "layer_n": {
            (layer data)
        },

        "root": {
            "type": "sequence",
            "values": [
                "layer_0",
                "layer_1",
                ...
                "layer_n"
            ]
        }
    }
}
```

Figure 4.1: JSON format of Basic String Diagrams

Type	Fields	Notation
prism	“path”, “ $>\|$ ”, “ $<\|$ ”, “ $\|>$ ”, “ $\|<$ ”	$\mathcal{A}$
sum	“values”	$\mathcal{A}_1 \oplus \mathcal{A}_2 \oplus \cdots \oplus \mathcal{A}_n$
sequence	“values”	$\mathcal{A}_1 \fatsemi \mathcal{A}_2 \fatsemi \cdots \fatsemi \mathcal{A}_n$

Table 4.1: Types and fields of BSDs

A BSD in JSON format consists of a root component and a dictionary of components. These components can have three possible types: “prism”, “sum” or “sequence”. The types of BSD components in JSON format are summarized in Table 4.1.

PRISM

The first type is “prism”. PRISM is a format with which MDPs are commonly defined [3]. Models in PRISM format are accepted by Storm.

Each PRISM component is an oMDP like the one in Figure 2.3. The structure of PRISM components is depicted in Figure 4.2.

The “path” value specifies the filename of the PRISM file which the component represents. The PRISM file should be in the same folder as the JSON file. The four symbols below the path specify the right-facing entrances, the left-facing exits, the right-facing exits and the left-facing entrances. Each of these entrance/exit attributes contain a list of zero or more names of states, which are the entrances/exits with which the component is connected to the rest of the diagram.

Sum

The second type is “sum”. The sum type represents a sum of components as we have described them in Chapter 2.

Sum components have an attribute called “values”, which contains the names of all the components that make up the sum. All of these components must be of the “prism” type.

Sums may contain more than two components. For the resulting diagram, it does not matter whether the components are summed starting from the left/top or from the right/bottom. This is because the sum operation is associative (see Section 2.4.2).

```
"component_name": {
    "type": "prism",
    "path": "path.prism",
    ">|": [
        "s_0",
        "s_1",
        ...
    ],
    "<|": [
        "s_2",
        "s_3",
        ...
    ],
    "|>": [
        "s_4",
        "s_5",
        ...
    ],
    "|<": [
        "s_6",
        "s_7",
        ...
    ]
}
```

Figure 4.2: PRISM component in JSON format

```
"sum_name": {
    "type": "sum",
    "values": [
        "comp_0",
        "comp_1",
        ...
        "comp_n"
    ]
}
```

Figure 4.3: Sum component in JSON format

```
"sequence_name": {
    "type": "sequence",
    "values": [
        "layer_0",
        "layer_1",
        ...
        "layer_n"
    ]
}
```

Figure 4.4: Sequence in JSON format

Sequence

The last type is "sequence". The sequence type represents a sequence of sequentially composed components.

As with the sum type, the sequence type has an attribute called "values", which contains the names of all layers that make up the sequence. For sequences it also holds that it does not matter if its layers are sequentially composed from left to right or from right to left because of the associativity of sequential composition (see Section 2.4.2).

Every String Diagram contains exactly one root sequence. The name of this root sequence is defined by the value in the very first "root" attribute in Figure 4.1.

Example 13. A complete example of a small diagram in JSON format is shown in Figure 4.5. This example shows what the expression "$\mathcal{A} \fatsemi (\mathcal{B} \oplus \mathcal{C}) \fatsemi \mathcal{A}$" looks like in JSON format.

```
{
  "root: "root",
  "components": {
      "A": {
        "type": "prism",
        "path": "a.prism",
        ">|": ["s_0"],
        "<|": ["s_1"],
        "|>": ["s_2"],
        "|<": ["s_4"]
      },
      "B": {
        "type": "prism",
        "path": "b.prism",
        ">|": ["s_5"],
        "|>": ["s_6"],
      },
      "C": {
        "type": "prism",
        "path": "c.prism",
        "<|": ["s_7"],
        "|<": ["s_8"]
      },
      "sum_0": {
        "type": "sum",
        "values": [
            "B",
            "C"
        ]
      }
      "root": {
        "type": "sequence",
        "values": [
            "A",
            "sum_0",
            "A"
        ]
      }
  }
}
```

Figure 4.5: The expression "$\mathcal{A} \fatsemi (\mathcal{B} \oplus \mathcal{C}) \fatsemi \mathcal{A}$" in JSON format

Type	Fields	Notation
prism	"path", " $>\|$ ", " $<\|$ ", "$\|>$ ", "$\|<$ ", **"maps"**	$\mathcal{A}$
sum	"values", **"maps"**	$\mathcal{A}_1 \oplus \mathcal{A}_2 \oplus \cdots \oplus \mathcal{A}_n$
sequence	"values"	$\mathcal{A}_1 \fatsemi \mathcal{A}_2 \fatsemi \cdots \fatsemi \mathcal{A}_n$
repeat	"value", "amount"	$\mathcal{R}(\mathcal{A}, n)$

Table 4.2: Types and fields of ESDs. Extensions are highlighted in **bold**.

4.2.2 ESDs in JSON Format

In the previous chapter we introduced two extensions to BSDs: the repeat-extension and the switch-extension. Table 4.2 summarizes the types of components in ESDs.

Repeat

Repeat components in an Extended String Diagram are structured as depicted in Figure 4.6.

```
"name": {
    "type": "repeat",
    "value": "layer_name",
    "amount": number
}
```

Figure 4.6: Repeat component in JSON format

In this structure, "name" is the name of all the repeated layers combined. The "layer_name" in the "value" attribute is the name of the layer that is to be repeated. Of course, "number" denotes the amount of times the layer should be repeated.

Switch

The switch extension does not introduce a new type of component. Instead, it extends components of the "prism" and the "sum" type with a "maps" field. Sequence- and repeat-components do not get a "maps" field. This is because in every case both the first and the last component of a sequence or a repeat is a PRISM- or sum-component. Therefore, if a mapping is applied to a repeat or a sequence, this mapping is stored in the repeat's or sequence's first or last layer, which is either a PRISM- or a sum-component.

The maps field contains four lists, one for each type of entrance/exit. Each of these lists contains a list of indices, such that there is an index for

every entrance/exit in the oMDP/sum. This index is the index in the previous or next layer to which the entrance/exit it belongs to maps. Essentially, the mapping creates a permutation of each of the entrance/exit lists, where the new order of the indices defines the new mapping.

Example 14. Figure 4.7 shows a simple example diagram where non-default arrows have been used. The corresponding maps field of the "initial" component is displayed in Figure 4.8.

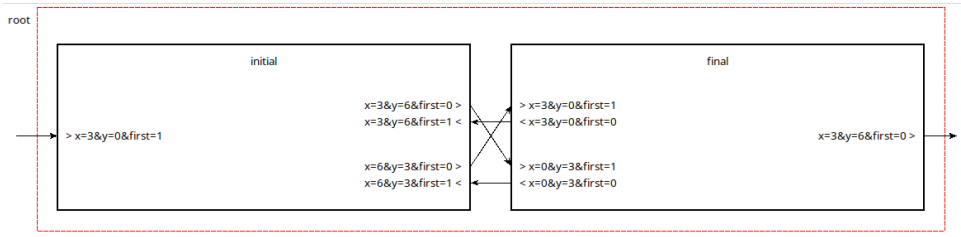

Figure 4.7: Custom layer connections

```
"maps": [
    [0],
    [],
    [1, 0],
    [0, 1]
]
```

Figure 4.8: Custom "maps" field

4.3 The UI

4.3.1 The Component Tree

In the previous section we discussed what String Diagrams look like in JSON format. Upon starting the UI, the user is prompted with a pop-up window asking him to select such a String Diagram in JSON format to import into the program. When a file is selected, the program parses the JSON file and draws the diagram that it represents on the canvas. It does not matter whether the input file is an ESD or a BSD, because the BSD format is a strict subset of the ESD format.

In order to be able to visualize the diagram and to allow the user to make changes to the diagram, the program keeps an internal state of all the components that are present in the diagram. We implemented this as a "Component Tree". In almost every case, the root of the tree is the root sequence. The root component could also be a PRISM-, sum- or repeat-component, but this would not result in a very interesting diagram. The root sequence branches out into any number of nodes. Leaf nodes must always be PRISM-components. It is not possible for a sum- or repeat-component to have no children. If a user removes the last child of a Sum, the program therefore also removes the entire Sum itself and adjusts the visualization accordingly. If a Repeat with two children has its repeat count decreased to one, the Repeat will be converted to a PRISM-component.

For every object we store its name, x and y coordinates and height in order to be able to accurately display the diagram. Every component other than the root sequence has a parent object, which also means that every object other than PRISM-components have children objects. Additionally, every object type has its own buttons, which allow the user to edit the diagram in various ways. How exactly the diagram can be edited will be discussed in the next subsection. An object's name, rectangle, and buttons each have their own IDs. These IDs are used by the program to update or delete all parts of an object when this is necessary.

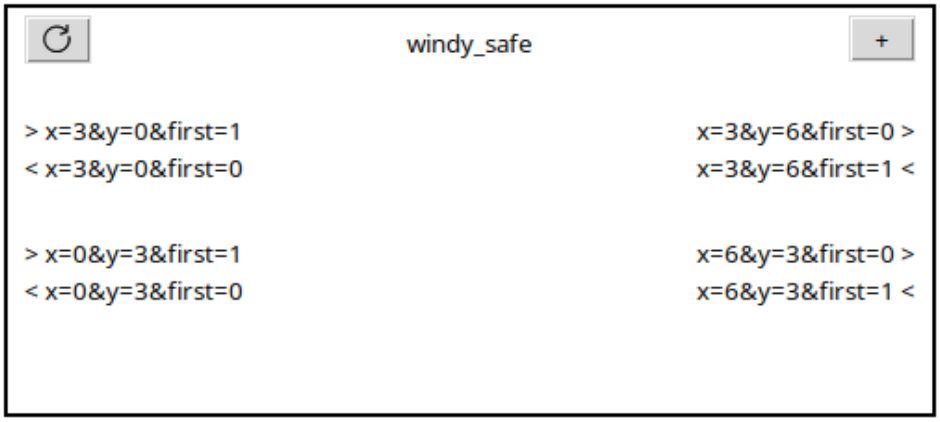

Figure 4.9: PRISM component in the UI

4.3.2 Visualization and Editing

We have four different types of objects: PRISM-components, sums, repeats and sequences. In this subsection we describe how each of these objects can be edited in our UI.

PRISM-components

A PRISM-component is visualized as in Figure 4.9. The name of the component is displayed in the top-middle. Its buttons are displayed in the top-left and the top-right corners of the component. The entrances and exits of the component are displayed on its left and right borders. The small arrowheads indicate the direction of the entrance/exit.
Other than completely deleting a component, there are two possible ways to change it. The component can be converted into either a Repeat or a Sum.

In order to convert a PRISM-component into a Repeat, the repeat-button in the top-left corner of the component must be clicked. Upon clicking the button, the user is prompted to enter the name of the new Repeat object. After choosing a name, the PRISM-component is repeated once and is then converted into a Repeat object.

Alternatively, the component can be converted into a Sum. This can be done by clicking the plus-button in the top-right corner of the component. Clicking this button opens up a drop-down menu containing the names of all the components present in the diagram's JSON-file. When a user selects one of these components, he will again first be prompted to enter a name for the new Sum. Once he has entered a name, the original component will be summed with the component selected in the drop-down menu.

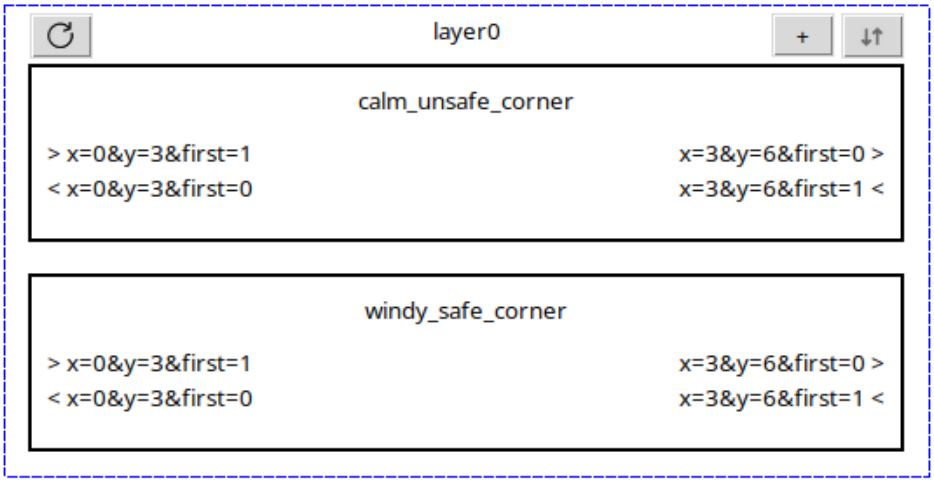

Figure 4.10: Sum component in the UI

Sums

An example of how a Sum is visualized in the UI is shown in Figure 4.10. Sums consist of two or more vertically stacked PRISM-components surrounded by a blue border. The Sum's name is displayed in the top-middle. Components within a Sum are, of course, already part of a Sum, so as a result they do not contain buttons to convert them into another object. The Sum itself contains a repeat-button in the top-left corner and a plus-button and reorder-button in the top-right corner. The repeat-button functions the same as the repeat-button for PRISM-components. The plus-button can be used to add another PRISM-component to the Sum. Adding another PRISM-component will insert the new component at the bottom of the Sum. Components can be removed from the Sum by clicking the middle mouse-button on the component's border. The order of the components can be changed by using the reorder-button. Clicking this button opens a popup window containing the names of the Sum's components. These names can be reordered by dragging them. Upon confirmation of the changes the Sum will be redrawn with its components in the new order.

Repeats

Both PRISM-components and Sums can be repeated. Figure 4.11 shows a Repeat of a PRISM-component.

Repeat components consist of two or more Components or Sums. Repeat components are visualized using a green border. The Repeat's name is displayed in its top-left corner. Its buttons are displayed above the top-right side of its first component.

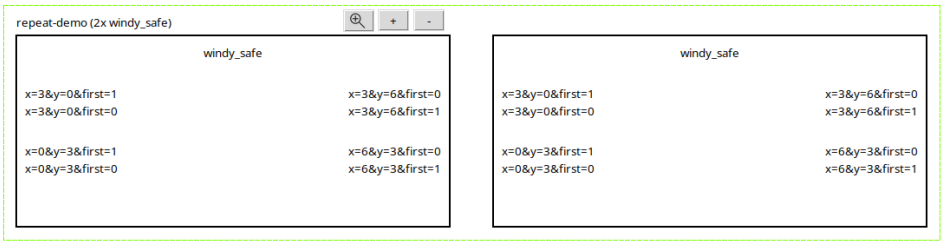

Figure 4.11: Repeat component in the UI

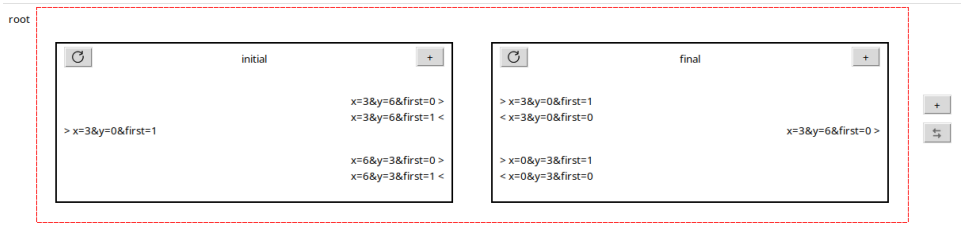

Figure 4.12: Sequence in the UI

The first button with the magnifying glass symbol is used to switch the Repeat between compact and expanded view. Components could be repeated a lot of times, and it is therefore not always practical to always visualize all the repeated components. Switching the Repeat to compact view therefore only visualizes its first component. The amount of times this component is repeated can still be seen in the top-left corner next to the name of the Repeat.

The plus- and minus-buttons can be used to add or remove a component to or from the repeat. Of course, these buttons work when the Repeat is in both compact and expanded view.

Sequences

A sequence consists of any number of sequentially composed PRISM components, sums and/or repeats. A very short example sequence of just two components is displayed in Figure 4.12.

The sequence's name is displayed outside its border on the top-left side. A sequence can be edited in three ways: layers can be removed, added and reordered.

Removing a layer can be done by clicking the layer's border with the middle mouse button. Adding a component to the sequence can be done by clicking the plus-button on the right-side border of the sequence. This works similarly to adding a component to a Sum: clicking the button opens up a drop-down menu allowing the user to select any component present in the diagram's JSON file. The selected component will be added as the last

layer in the sequence.

The reorder button can be found under the plus-button. Again, reordering layers in a sequence is very similar to reordering components in a Sum. Upon clicking the button, a popup window appears allowing the user to reorder the sequence's components by dragging them. Confirming the order in the popup window results in the sequence being redrawn in the provided order.

4.3.3 Connecting the Layers

When the user is done editing his diagram he can choose to create custom connections between the diagram's layers. As we discussed in the previous chapter, layers are connected with straight arrows by default. However, we do not display these default arrows in the editor, because it would be a mess to redraw all arrows when a user is rearranging, adding or deleting components, especially when a user is editing a diagram in multiple steps.

Therefore, when the user is done editing the components of the diagram, he can select the "Edit Arrows" command in the editor menu. After having selected this command, the user will be able to show the default arrows and/or replace the default arrows with custom arrows. Custom arrows can be created by right-clicking an exit and dragging the cursor over a valid entrance.

4.3.4 Exporting

Once the user is satisfied with the changes to the diagram, the visualization of the diagram can be finalized. This can be done from both the initial editor and the arrow-editor by selecting the "Finalize" command in the editor menu. The finalize-command visualizes the entire diagram, including all arrows and without the visual clutter of the editor-buttons. At this point the user can no longer edit the diagram. The user can, however, choose to export his newly created diagram. He can choose to export the diagram to both the Basic and the Extended String Diagram format, by selecting the corresponding commands in the file-menu.

4.4 Converting ESDs to BSDs

In the previous chapter we proved that ESDs can always be converted into BSDs without losing expressivity. In our UI users can choose to export their diagram to both the ESD- and BSD-format. If a user wants to export an ESD diagram to BSD-format, our program therefore also needs to be able to apply such a conversion. In this section we explain how we implemented the conversions from ESDs to BSDs in our UI.

4.4.1 Converting Repeat to BSD

It is straightforward to convert repeat components. In the repeat component's parent sequence the repeat component simply has to be replaced by the component that is repeated for the number of times in the "amount" field of the repeat component. An example is shown in Figure 4.13.

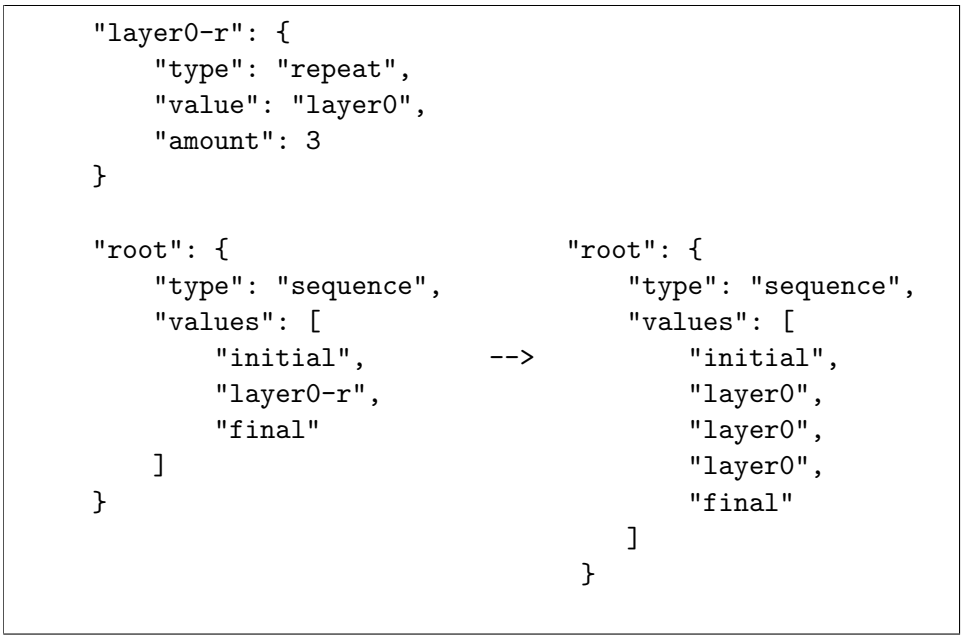

Figure 4.13: Converting a repeat component to BSD format

4.4.2 Converting Switch to BSD

Converting switch components is a little more complicated. As we described in the previous chapter, we insert a new PRISM-component before and after each component with a custom map.

In our UI, we call these newly added components "switchx", where x is a number, starting with 0 for the first newly inserted component and going up by one for each next component. The entrances of these new components are the same as the exits of the component with the map. The exits of these new components are a permutation of their entrances, such that their new order now corresponds to the indices of the mapping of the original component.

The new components are added to the dictionary of components in the JSON file, and their names are inserted before or after the name of the original component in the original component's parent sequence, whichever is applicable.

Chapter 5

Conclusions

In this thesis we proposed two extensions to String Diagrams of MDPs. The repeat extension makes it possible for String Diagrams to be more compactly represented and makes it easier for users to include, remove, or edit repetitions of components of the String Diagram. The switch extension allows the user to change how the layers of String Diagrams are connected. The extensions improve usability and allow the user to create the diagram they want more easily. We have proven that applying these extensions to a String Diagram does not decrease the diagram's expressivity, by proving that in any case an Extended String Diagram (ESD) can be converted to an equivalent Basic String Diagram (BSD).

Additionally, we have created a user interface that allows the user to import, edit, and export String Diagrams. This tool makes it easier for users to work and experiment with both BSDs and ESDs. We have demonstrated the applicability of our proposed extensions to String Diagrams using this tool.

In future work more extensions to String Diagrams can be proposed. Future work can also continue to expand the user interface. Additional features could include implementing the "trace" type, enabling more elaborate nesting and configurations of the diagram's sequences, sums, and repeats, and the implementation could be changed such that it can calculate and show how changing the model affects the model's reachability probabilities.

Bibliography

[1] Filippo Bonchi, Joshua Holland, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. Diagrammatic algebra: from linear to concurrent systems. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–28, 2019.

[2] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. The probabilistic model checker storm. *Int. J. Softw. Tools Technol. Transf.*, 24(4):589–610, 2022.

[3] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[4] Roger Penrose et al. Applications of negative dimensional tensors. *Combinatorial mathematics and its applications*, 1:221–244, 1971.

[5] Robin Piedeleu, Dimitri Kartsaklis, Bob Coecke, and Mehrnoosh Sadrzadeh. Open system categorical quantum semantics in natural language processing. In *6th Conference on Algebra and Coalgebra in Computer Science, CALCO 2015, June 24-26, 2015, Nijmegen, The Netherlands*, volume 35 of *LIPIcs*, pages 270–289. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

[6] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[7] Emily Riehl. *Category theory in context*. Courier Dover Publications, 2017.

[8] Kazuki Watanabe, Clovis Eberhart, Kazuyuki Asada, and Ichiro Hasuo. Compositional probabilistic model checking with string diagrams of mdps. In *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part III*, volume 13966 of *Lecture Notes in Computer Science*, pages 40–61. Springer, 2023.

[9] Kazuki Watanabe, Marck van der Vegt, Ichiro Hasuo, Jurriaan Rot, and Sebastian Junges. Pareto curves for compositionally model checking string diagrams of mdps. In *Tools and Algorithms for the Construction and Analysis of Systems - 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part II*, volume 14571 of *Lecture Notes in Computer Science*, pages 279–298. Springer, 2024.

Appendix A

Appendix

A.1 Proofs of Equivalence Relations

A.1.1 Identity

Proof. We need to prove the identity relation in two directions. That is, we need to prove $\mathcal{I} \fatsemi \mathcal{A} = \mathcal{A}$ and $\mathcal{A} \fatsemi \mathcal{I} = \mathcal{A}$.

$\mathcal{I} \fatsemi \mathcal{A} = \mathcal{A}$:

For $\mathcal{I} \fatsemi \mathcal{A}$ to be equal to $\mathcal{A}$ we need to show that:

1. $S^{\mathcal{I} \fatsemi \mathcal{A}} = S^{\mathcal{A}}$ (their states are equal)
2. $E^{\mathcal{I} \fatsemi \mathcal{A}}(i) = E^{\mathcal{A}}(i)$ (their entry functions are equal)
3. $P^{\mathcal{I} \fatsemi \mathcal{A}} = P^{\mathcal{A}}$ (their transition functions are equal)

$$
\begin{aligned}
S^{\mathcal{I} \fatsemi \mathcal{A}} &= S^{\mathcal{I}} \uplus S^{\mathcal{A}} && \text{(by the definition of the } \fatsemi\text{-operator)} \\
&= \emptyset \uplus S^{\mathcal{A}} && \text{(by the definition of the identity component)} \\
&= S^{\mathcal{A}}
\end{aligned}
$$

This proves 1.

$$
\begin{aligned}
E^{\mathcal{I} \fatsemi \mathcal{A}}(i) &= E^{\mathcal{A}}(E^{\mathcal{I}}(i)) && \text{(by the definition of the } \fatsemi\text{-operator)} \\
&= E^{\mathcal{A}}(i) && \text{(by the definition of the identity component)}
\end{aligned}
$$

This proves 2.

$$
\begin{aligned}
P^{\mathcal{I} \fatsemi \mathcal{A}}(s^{\mathcal{I}}, a, s') &= \begin{cases} P^{\mathcal{I}}(s^{\mathcal{I}}, a, s') & \text{if s' } \in S^{\mathcal{A}} \\ \sum_{i \in [k]} P^{\mathcal{I}}(s^{\mathcal{I}}, a, i) \cdot \delta_{E^{\mathcal{A}}(i) = s'} & \text{otherwise} \end{cases} \\
&= ! \quad \text{(by definition of } P^{\mathcal{I}}\text{)}
\end{aligned}
$$

$$P^{\mathcal{I}\fatsemi\mathcal{A}}(s^{\mathcal{A}}, a, s') = \begin{cases} P^{\mathcal{A}}(s^{\mathcal{A}}, a, s') & \text{if s' } \in S^{\mathcal{A}} + [n] \\ 0 & \text{otherwise} \end{cases}$$

This proves 3.

1, 2, and 3 hold, so $\mathcal{I} \fatsemi \mathcal{A} = \mathcal{A}$

$\mathcal{A} \fatsemi \mathcal{I} = \mathcal{A}$:

For $\mathcal{A} \fatsemi \mathcal{I}$ to be equal to $\mathcal{A}$ we need to show that:

1. $S^{\mathcal{A}\fatsemi\mathcal{I}} = S^{\mathcal{A}}$
2. $E^{\mathcal{A}\fatsemi\mathcal{I}}(i) = E^{\mathcal{A}}(i)$
3. $P^{\mathcal{A}\fatsemi\mathcal{I}} = P^{\mathcal{A}}$

$$\begin{aligned} S^{\mathcal{A}\fatsemi\mathcal{I}} &= S^{\mathcal{A}} \uplus S^{\mathcal{I}} && \text{(by the definition of the } \fatsemi\text{-operator)} \\ &= \mathcal{A} \uplus \emptyset && \text{(by the definition of the identity component)} \\ &= S^{\mathcal{A}} \end{aligned}$$

This proves 1.

$$E^{\mathcal{A}\fatsemi\mathcal{I}}(i) = E^{\mathcal{A}}(i) \qquad \text{(by the definition of the identity component)}$$

This proves 2.

$$P^{\mathcal{A}\fatsemi\mathcal{I}}(s^{\mathcal{A}}, a, s') = \begin{cases} P^{\mathcal{A}}(s^{\mathcal{A}}, a, s') & \text{if s' } \in S^{\mathcal{A}} \\ \sum_{i \in [k]} P^{\mathcal{A}}(s^{\mathcal{A}}, a, i) \cdot \delta_{E^{\mathcal{I}}(i) = s'} & \text{otherwise} \end{cases}$$

$$= \begin{cases} P^{\mathcal{A}}(s^{\mathcal{A}}, a, s') & \text{if s' } \in S^{\mathcal{A}} \\ 0 & \text{otherwise} \end{cases}$$

$$P^{\mathcal{A}\fatsemi\mathcal{I}}(s^{\mathcal{I}}, a, s') = \begin{cases} P^{\mathcal{I}}(s^{\mathcal{I}}, a, s') & \text{if s' } \in S^{\mathcal{I}} + [n] \\ 0 & \text{otherwise} \end{cases}$$

$$= ! \qquad \text{by definition of } P^{\mathcal{I}}$$

1, 2, and 3 hold, so $\mathcal{A} \fatsemi \mathcal{I} = \mathcal{A}$

We have proven that both $\mathcal{I} \fatsemi \mathcal{A} = \mathcal{A}$ and $\mathcal{A} \fatsemi \mathcal{I} = \mathcal{A}$, which concludes our proof of the identity relation. □

A.1.2 Associativity of Sequential Composition

Proof. We need to prove: $\mathcal{A} \fatsemi (\mathcal{B} \fatsemi \mathcal{C}) = (\mathcal{A} \fatsemi \mathcal{B}) \fatsemi \mathcal{C}$

For the above to hold we need to show that:

1. $S^{\mathcal{A} \fatsemi (\mathcal{B} \fatsemi \mathcal{C})} = S^{(\mathcal{A} \fatsemi \mathcal{B}) \fatsemi \mathcal{C}}$
2. $E^{\mathcal{A} \fatsemi (\mathcal{B} \fatsemi \mathcal{C})}(i) = E^{(\mathcal{A} \fatsemi \mathcal{B}) \fatsemi \mathcal{C}}(i)$
3. $P^{\mathcal{A} \fatsemi (\mathcal{B} \fatsemi \mathcal{C})} = P^{(\mathcal{A} \fatsemi \mathcal{B}) \fatsemi \mathcal{C}}$

$$
\begin{aligned}
S^{\mathcal{A} \fatsemi (\mathcal{B} \fatsemi \mathcal{C})} &= S^{\mathcal{A}} \uplus S^{\mathcal{B} \fatsemi \mathcal{C}} \\
&= S^{\mathcal{A}} \uplus S^{\mathcal{B}} \uplus S^{\mathcal{C}} \\
&= S^{\mathcal{A} \fatsemi \mathcal{B}} \uplus S^{\mathcal{C}} \\
&= S^{(\mathcal{A} \fatsemi \mathcal{B}) \fatsemi \mathcal{C}}
\end{aligned}
$$

This proves 1.

$$
\begin{aligned}
E^{\mathcal{A} \fatsemi (\mathcal{B} \fatsemi \mathcal{C})}(i) &= E^{\mathcal{B} \fatsemi \mathcal{C}}(E^{\mathcal{A}}(i)) \\
&= E^{\mathcal{C}}(E^{\mathcal{B}}(E^{\mathcal{A}}(i))) \\
&= E^{\mathcal{C}}(E^{\mathcal{A} \fatsemi \mathcal{B}}(i)) \\
&= E^{(\mathcal{A} \fatsemi \mathcal{B}) \fatsemi \mathcal{C}}(i)
\end{aligned}
$$

This proves 2.

$$
P^{\mathcal{A} \fatsemi (\mathcal{B} \fatsemi \mathcal{C})}(s^{\mathcal{A}}, a, s') = \begin{cases} P^{\mathcal{A}}(s^{\mathcal{A}}, a, s') & \text{if } s' \in S^{\mathcal{A}} \\ \sum_{i \in [k]} P^{\mathcal{A}}(s^{\mathcal{A}}, a, i) \cdot \delta_{E^{\mathcal{B} \fatsemi \mathcal{C}}(i) = s'} & \text{if } s' \in S^{\mathcal{B} \fatsemi \mathcal{C}} + [n] \end{cases}
$$

$$
P^{\mathcal{A} \fatsemi (\mathcal{B} \fatsemi \mathcal{C})}(s^{\mathcal{B} \fatsemi \mathcal{C}}, a, s') = \begin{cases} P^{\mathcal{B} \fatsemi \mathcal{C}}(s^{\mathcal{B} \fatsemi \mathcal{C}}, a, s') & \text{if } s' \in S^{\mathcal{B} \fatsemi \mathcal{C}} + [n] \\ 0 & \text{otherwise} \end{cases}
$$

$$
P^{\mathcal{A} \fatsemi (\mathcal{B} \fatsemi \mathcal{C})}(s^{\mathcal{B}}, a, s') = \begin{cases} P^{\mathcal{B}}(s^{\mathcal{B}}, a, s') & \text{if } s' \in S^{\mathcal{B}} \\ \sum_{i \in [k]} P^{\mathcal{B}}(s^{\mathcal{B}}, a, i) \cdot \delta_{E^{\mathcal{C}}(i) = s'} & \text{if } s' \in S^{\mathcal{C}} + [n] \end{cases}
$$

$$
P^{\mathcal{A} \fatsemi (\mathcal{B} \fatsemi \mathcal{C})}(s^{\mathcal{C}}, a, s') = \begin{cases} P^{\mathcal{C}}(s^{\mathcal{C}}, a, s') & \text{if } s' \in S^{\mathcal{C}} + [n] \\ 0 & \text{otherwise} \end{cases}
$$

$$P^{(\mathcal{A}\fatsemi\mathcal{B})\fatsemi\mathcal{C}}(s^{\mathcal{A}\fatsemi\mathcal{B}}, a, s') = \begin{cases} P^{\mathcal{A}\fatsemi\mathcal{B}}(s^{\mathcal{A}\fatsemi\mathcal{B}}, a, s') & \text{if } s' \in S^{\mathcal{A}\fatsemi\mathcal{B}} \\ \sum_{i\in[k]} P^{\mathcal{A}\fatsemi\mathcal{B}}(s^{\mathcal{A}\fatsemi\mathcal{B}}, a, i) \cdot \delta_{E^{\mathcal{C}}(i)=s'} & \text{if } s' \in S^{\mathcal{C}} + [n] \end{cases}$$

$$P^{(\mathcal{A}\fatsemi\mathcal{B})\fatsemi\mathcal{C}}(s^{\mathcal{A}}, a, s') = \begin{cases} P^{\mathcal{A}}(s^{\mathcal{A}}, a, s') & \text{if } s' \in S^{\mathcal{A}} \\ \sum_{i\in[k]} P^{\mathcal{A}}(s^{\mathcal{A}}, a, i) \cdot \delta_{E^{\mathcal{B}\fatsemi\mathcal{C}}(i)=s'} & \text{if } s' \in S^{\mathcal{B}\fatsemi\mathcal{C}} + [n] \end{cases}$$

$$P^{(\mathcal{A}\fatsemi\mathcal{B})\fatsemi\mathcal{C}}(s^{\mathcal{B}}, a, s') = \begin{cases} P^{\mathcal{B}}(s^{\mathcal{B}}, a, s') & \text{if } s' \in S^{\mathcal{B}} \\ \sum_{i\in[k]} P^{\mathcal{B}}(s^{\mathcal{B}}, a, i) \cdot \delta_{E^{\mathcal{C}}(i)=s'} & \text{if } s' \in S^{\mathcal{C}} + [n] \end{cases}$$

$$P^{(\mathcal{A}\fatsemi\mathcal{B})\fatsemi\mathcal{C}}(s^{\mathcal{C}}, a, s') = \begin{cases} P^{\mathcal{C}}(s^{\mathcal{C}}, a, s') & \text{if } s' \in S^{\mathcal{C}} + [n] \\ 0 & \text{otherwise} \end{cases}$$

From the above equations we can determine that:

$$P^{\mathcal{A}\fatsemi(\mathcal{B}\fatsemi\mathcal{C})}(s^{\mathcal{A}}, a, s') = P^{(\mathcal{A}\fatsemi\mathcal{B})\fatsemi\mathcal{C}}(s^{\mathcal{A}}, a, s')$$
$$P^{\mathcal{A}\fatsemi(\mathcal{B}\fatsemi\mathcal{C})}(s^{\mathcal{B}}, a, s') = P^{(\mathcal{A}\fatsemi\mathcal{B})\fatsemi\mathcal{C}}(s^{\mathcal{B}}, a, s')$$
$$P^{\mathcal{A}\fatsemi(\mathcal{B}\fatsemi\mathcal{C})}(s^{\mathcal{C}}, a, s') = P^{(\mathcal{A}\fatsemi\mathcal{B})\fatsemi\mathcal{C}}(s^{\mathcal{C}}, a, s')$$

This proves 3.

Since all three conditions are met, we can conclude that sequential composition is associative. □

A.1.3 Associativity of Summing

Proof. We need to prove: $\mathcal{A} \oplus (\mathcal{B} \oplus \mathcal{C}) = (\mathcal{A} \oplus \mathcal{B}) \oplus \mathcal{C}$

For the above to hold we need to show that:

1. $S^{\mathcal{A} \oplus (\mathcal{B} \oplus \mathcal{C})} = S^{(\mathcal{A} \oplus \mathcal{B}) \oplus \mathcal{C}}$
2. $E^{\mathcal{A} \oplus (\mathcal{B} \oplus \mathcal{C})}(i) = E^{(\mathcal{A} \oplus \mathcal{B}) \oplus \mathcal{C}}(i)$
3. $P^{\mathcal{A} \oplus (\mathcal{B} \oplus \mathcal{C})} = P^{(\mathcal{A} \oplus \mathcal{B}) \oplus \mathcal{C}}$

$$\begin{aligned} S^{\mathcal{A} \oplus (\mathcal{B} \oplus \mathcal{C})} &= S^{\mathcal{A}} \uplus S^{\mathcal{B} \oplus \mathcal{C}} \\ &= S^{\mathcal{A}} \uplus S^{\mathcal{B}} \uplus S^{\mathcal{C}} \\ &= S^{\mathcal{A} \oplus \mathcal{B}} \uplus S^{\mathcal{C}} \\ &= S^{(\mathcal{A} \oplus \mathcal{B}) \oplus \mathcal{C}} \end{aligned}$$

This proves 1.

$$
\begin{aligned}
E^{\mathcal{A} \oplus (\mathcal{B} \oplus \mathcal{C})}(i) &= E^{\mathcal{A}}(i) \uplus E^{\mathcal{B} \oplus \mathcal{C}}(i) \\
&= E^{\mathcal{A}}(i) \uplus E^{\mathcal{B}}(i) \uplus E^{\mathcal{C}}(i) \\
&= E^{\mathcal{A} \oplus \mathcal{B}}(i) \uplus E^{\mathcal{C}}(i) \\
&= E^{(\mathcal{A} \oplus \mathcal{B}) \oplus \mathcal{C}}(i)
\end{aligned}
$$

This proves 2.

$$
P^{\mathcal{A}\oplus(\mathcal{B}\oplus\mathcal{C})}(s^{\mathcal{A}}, a, s') = \begin{cases} P^{\mathcal{A}}(s^{\mathcal{A}}, a, s') & \text{if } a \in A^{\mathcal{A}} \text{ and } s' \in S^{\mathcal{A}} \\ 0 & \text{otherwise} \end{cases}
$$

$$
P^{\mathcal{A}\oplus(\mathcal{B}\oplus\mathcal{C})}(s^{\mathcal{B}\oplus\mathcal{C}}, a, s') = \begin{cases} P^{\mathcal{B}\oplus\mathcal{C}}(s, a, s') & \text{if } s, s' \in S^{\mathcal{B}\oplus\mathcal{C}} \text{ and } a \in A^{\mathcal{B}\oplus\mathcal{C}} \\ 0 & \text{otherwise} \end{cases}
$$

$$
P^{\mathcal{A}\oplus(\mathcal{B}\oplus\mathcal{C})}(s^{\mathcal{B}}, a, s') = \begin{cases} P^{\mathcal{B}}(s^{\mathcal{B}}, a, s') & \text{if } a \in A^{\mathcal{B}} \text{ and } s' \in S^{\mathcal{B}} \\ 0 & \text{otherwise} \end{cases}
$$

$$
P^{\mathcal{A}\oplus(\mathcal{B}\oplus\mathcal{C})}(s^{\mathcal{C}}, a, s') = \begin{cases} P^{\mathcal{C}}(s^{\mathcal{C}}, a, s') & \text{if } a \in A^{\mathcal{C}} \text{ and } s' \in S^{\mathcal{C}} \\ 0 & \text{otherwise} \end{cases}
$$

$$
P^{(\mathcal{A}\oplus\mathcal{B})\oplus\mathcal{C}}(s^{\mathcal{A}\oplus\mathcal{B}}, a, s') = \begin{cases} P^{\mathcal{A}\oplus\mathcal{B}}(s, a, s') & \text{if } s, s' \in S^{\mathcal{A}\oplus\mathcal{B}} \text{ and } a \in A^{\mathcal{A}\oplus\mathcal{B}} \\ 0 & \text{otherwise} \end{cases}
$$

$$
P^{(\mathcal{A}\oplus\mathcal{B})\oplus\mathcal{C}}(s^{\mathcal{A}}, a, s') = \begin{cases} P^{\mathcal{A}}(s^{\mathcal{A}}, a, s') & \text{if } a \in A^{\mathcal{A}} \text{ and } s' \in S^{\mathcal{A}} \\ 0 & \text{otherwise} \end{cases}
$$

$$
P^{(\mathcal{A}\oplus\mathcal{B})\oplus\mathcal{C}}(s^{\mathcal{B}}, a, s') = \begin{cases} P^{\mathcal{B}}(s^{\mathcal{B}}, a, s') & \text{if } a \in A^{\mathcal{B}} \text{ and } s' \in S^{\mathcal{B}} \\ 0 & \text{otherwise} \end{cases}
$$

$$
P^{(\mathcal{A}\oplus\mathcal{B})\oplus\mathcal{C}}(s^{\mathcal{C}}, a, s') = \begin{cases} P^{\mathcal{C}}(s^{\mathcal{C}}, a, s') & \text{if } a \in A^{\mathcal{C}} \text{ and } s' \in S^{\mathcal{C}} \\ 0 & \text{otherwise} \end{cases}
$$

From the above equations we can determine that:

$$
\begin{aligned}
P^{\mathcal{A}\oplus(\mathcal{B}\oplus\mathcal{C})}(s^{\mathcal{A}}, a, s') &= P^{(\mathcal{A}\oplus\mathcal{B})\oplus\mathcal{C}}(s^{\mathcal{A}}, a, s') \\
P^{\mathcal{A}\oplus(\mathcal{B}\oplus\mathcal{C})}(s^{\mathcal{B}}, a, s') &= P^{(\mathcal{A}\oplus\mathcal{B})\oplus\mathcal{C}}(s^{\mathcal{B}}, a, s') \\
P^{\mathcal{A}\oplus(\mathcal{B}\oplus\mathcal{C})}(s^{\mathcal{C}}, a, s') &= P^{(\mathcal{A}\oplus\mathcal{B})\oplus\mathcal{C}}(s^{\mathcal{C}}, a, s')
\end{aligned}
$$

This proves 3.

Since all three conditions are met, we can conclude that summing is associative. □

A.1.4 Bifunctoriality

Proof. We need to prove: $(\mathcal{A} \oplus \mathcal{B}) \fatsemi (\mathcal{C} \oplus \mathcal{D}) = (\mathcal{A} \fatsemi \mathcal{C}) \oplus (\mathcal{B} \fatsemi \mathcal{D})$

For the above to hold we need to show that:

1. $S^{(\mathcal{A}\oplus\mathcal{B})\fatsemi(\mathcal{C}\oplus\mathcal{D})} = S^{(\mathcal{A}\fatsemi\mathcal{C})\oplus(\mathcal{B}\fatsemi\mathcal{D})}$
2. $E^{(\mathcal{A}\oplus\mathcal{B})\fatsemi(\mathcal{C}\oplus\mathcal{D})}(i) = E^{(\mathcal{A}\fatsemi\mathcal{C})\oplus(\mathcal{B}\fatsemi\mathcal{D})}(i)$
3. $P^{(\mathcal{A}\oplus\mathcal{B})\fatsemi(\mathcal{C}\oplus\mathcal{D})} = P^{(\mathcal{A}\fatsemi\mathcal{C})\oplus(\mathcal{B}\fatsemi\mathcal{D})}$

$$\begin{aligned} S^{(\mathcal{A}\oplus\mathcal{B})\fatsemi(\mathcal{C}\oplus\mathcal{D})} &= S^{\mathcal{A}\oplus\mathcal{B}} \uplus S^{\mathcal{C}\oplus\mathcal{D}} \\ &= S^{\mathcal{A}} \uplus S^{\mathcal{B}} \uplus S^{\mathcal{C}} \uplus S^{\mathcal{D}} \\ &= S^{\mathcal{A}} \uplus S^{\mathcal{C}} \uplus S^{\mathcal{B}} \uplus S^{\mathcal{D}} \\ &= S^{\mathcal{A}\fatsemi\mathcal{C}} \uplus S^{\mathcal{B}\fatsemi\mathcal{D}} \\ &= S^{(\mathcal{A}\fatsemi\mathcal{C})\oplus(\mathcal{B}\fatsemi\mathcal{D})} \end{aligned}$$

This proves 1.

If $E^{\mathcal{A}}(i) \in S^{\mathcal{A}}$ and $E^{\mathcal{B}}(i) \in S^{\mathcal{B}}$:

$$\begin{aligned} E^{(\mathcal{A}\oplus\mathcal{B})\fatsemi(\mathcal{C}\oplus\mathcal{D})}(i) &= E^{\mathcal{A}\oplus\mathcal{B}}(i) \\ &= E^{\mathcal{A}}(i) \uplus E^{\mathcal{B}}(i) \end{aligned}$$

$$\begin{aligned} E^{(\mathcal{A}\fatsemi\mathcal{C})\oplus(\mathcal{B}\fatsemi\mathcal{D})}(i) &= E^{\mathcal{A}\fatsemi\mathcal{C}}(i) \uplus E^{\mathcal{B}\fatsemi\mathcal{D}}(i) \\ &= E^{\mathcal{A}}(i) \uplus E^{\mathcal{B}}(i) \end{aligned}$$

Otherwise: (if $E^{\mathcal{A}}(i) \notin S^{\mathcal{A}}$ and $E^{\mathcal{B}}(i) \notin S^{\mathcal{B}}$)

$$\begin{aligned} E^{(\mathcal{A}\oplus\mathcal{B})\fatsemi(\mathcal{C}\oplus\mathcal{D})}(i) &= E^{\mathcal{C}\oplus\mathcal{D}}(E^{\mathcal{A}\oplus\mathcal{B}}(i)) \\ &= (E^{\mathcal{C}}(i) \uplus E^{\mathcal{D}}(i))(E^{\mathcal{A}}(i) \uplus E^{\mathcal{B}}(i)) \\ &= E^{\mathcal{C}}(E^{\mathcal{A}}(i)) \uplus E^{\mathcal{D}}(E^{\mathcal{B}}(i)) \end{aligned}$$

$$\begin{aligned} E^{(\mathcal{A}\fatsemi\mathcal{C})\oplus(\mathcal{B}\fatsemi\mathcal{D})}(i) &= E^{(\mathcal{A}\fatsemi\mathcal{C})}(i) \uplus E^{\mathcal{B}\fatsemi\mathcal{D}}(i) \\ &= E^{\mathcal{C}}(E^{\mathcal{A}}(i)) \uplus E^{\mathcal{D}}(E^{\mathcal{B}}(i)) \end{aligned}$$

In both cases $E^{(\mathcal{A}\oplus\mathcal{B})\fatsemi(\mathcal{C}\oplus\mathcal{D})}(i) = E^{(\mathcal{A}\fatsemi\mathcal{C})\oplus(\mathcal{B}\fatsemi\mathcal{D})}(i)$, which proves 2.

$$P^{(\mathcal{A}\oplus\mathcal{B})\wr(\mathcal{C}\oplus\mathcal{D})}(s^{\mathcal{A}\oplus\mathcal{B}},a,s') = \begin{cases} P^{\mathcal{A}\oplus\mathcal{B}}(s^{\mathcal{A}\oplus\mathcal{B}},a,s') & \text{if } s'\in S^{\mathcal{A}\oplus\mathcal{B}} \\ \sum_{i\in[k]} P^{\mathcal{A}\oplus\mathcal{B}}(s^{\mathcal{A}\oplus\mathcal{B}},a,i)\cdot\delta_{E^{\mathcal{C}\oplus\mathcal{D}}(i)=s'} & \text{if } s'\in S^{\mathcal{C}\oplus\mathcal{D}}+[n] \end{cases}$$

$$P^{(\mathcal{A}\oplus\mathcal{B})\wr(\mathcal{C}\oplus\mathcal{D})}(s^{\mathcal{A}},a,s') = \begin{cases} P^{\mathcal{A}}(s^{\mathcal{A}},a,s') & \text{if } s'\in S^{\mathcal{A}} \\ \sum_{i\in[k]} P^{\mathcal{A}}(s^{\mathcal{A}},a,i)\cdot\delta_{E^{\mathcal{C}}(i)=s'} & \text{if } s'\in S^{\mathcal{C}}+[n] \end{cases}$$

$$P^{(\mathcal{A}\oplus\mathcal{B})\wr(\mathcal{C}\oplus\mathcal{D})}(s^{\mathcal{B}},a,s') = \begin{cases} P^{\mathcal{B}}(s^{\mathcal{B}},a,s') & \text{if } s'\in S^{\mathcal{B}} \\ \sum_{i\in[k]} P^{\mathcal{B}}(s^{\mathcal{B}},a,i)\cdot\delta_{E^{\mathcal{D}}(i)=s'} & \text{if } s'\in S^{\mathcal{D}}+[n] \end{cases}$$

$$P^{(\mathcal{A}\oplus\mathcal{B})\wr(\mathcal{C}\oplus\mathcal{D})}(s^{\mathcal{C}\oplus\mathcal{D}},a,s') = \begin{cases} P^{\mathcal{C}\oplus\mathcal{D}}(s^{\mathcal{C}\oplus\mathcal{D}},a,s') & \text{if } s'\in S^{\mathcal{C}\oplus\mathcal{D}}+[n] \\ 0 & \text{otherwise} \end{cases}$$

$$P^{(\mathcal{A}\oplus\mathcal{B})\wr(\mathcal{C}\oplus\mathcal{D})}(s^{\mathcal{C}},a,s') = \begin{cases} P^{\mathcal{C}}(s^{\mathcal{C}},a,s') & \text{if } s'\in S^{\mathcal{C}}+[n] \\ 0 & \text{otherwise} \end{cases}$$

$$P^{(\mathcal{A}\oplus\mathcal{B})\wr(\mathcal{C}\oplus\mathcal{D})}(s^{\mathcal{D}},a,s') = \begin{cases} P^{\mathcal{D}}(s^{\mathcal{D}},a,s') & \text{if } s'\in S^{\mathcal{D}}+[n] \\ 0 & \text{otherwise} \end{cases}$$

$$P^{(\mathcal{A}\wr\mathcal{C})\oplus(\mathcal{B}\wr\mathcal{D})}(s^{\mathcal{A}\wr\mathcal{C}},a,s') = \begin{cases} P^{\mathcal{A}\wr\mathcal{C}}(s^{\mathcal{A}\wr\mathcal{C}},a,s') & \text{if } s'\in S^{\mathcal{A}\wr\mathcal{C}} \\ 0 & \text{otherwise} \end{cases}$$

$$P^{(\mathcal{A}\wr\mathcal{C})\oplus(\mathcal{B}\wr\mathcal{D})}(s^{\mathcal{A}},a,s') = \begin{cases} P^{\mathcal{A}}(s^{\mathcal{A}},a,s') & \text{if } s'\in S^{\mathcal{A}} \\ \sum_{i\in[k]} P^{\mathcal{A}}(s^{\mathcal{A}},a,i)\cdot\delta_{E^{\mathcal{C}}(i)=s'} & \text{if } s'\in S^{\mathcal{C}}+[n] \end{cases}$$

$$P^{(\mathcal{A}\wr\mathcal{C})\oplus(\mathcal{B}\wr\mathcal{D})}(s^{\mathcal{C}},a,s') = \begin{cases} P^{\mathcal{C}}(s^{\mathcal{C}},a,s') & \text{if } s'\in S^{\mathcal{C}}+[n] \\ 0 & \text{otherwise} \end{cases}$$

$$P^{(\mathcal{A}\wr\mathcal{C})\oplus(\mathcal{B}\wr\mathcal{D})}(s^{\mathcal{B}\wr\mathcal{D}},a,s') = \begin{cases} P^{\mathcal{B}\wr\mathcal{D}}(s^{\mathcal{B}\wr\mathcal{D}},a,s') & \text{if } s'\in S^{\mathcal{B}\wr\mathcal{D}}+[n] \\ 0 & \text{otherwise} \end{cases}$$

$$P^{(\mathcal{A}\wr\mathcal{C})\oplus(\mathcal{B}\wr\mathcal{D})}(s^{\mathcal{B}},a,s') = \begin{cases} P^{\mathcal{B}}(s^{\mathcal{B}},a,s') & \text{if } s'\in S^{\mathcal{B}} \\ \sum_{i\in[k]} P^{\mathcal{B}}(s^{\mathcal{B}},a,i)\cdot\delta_{E^{\mathcal{D}}(i)=s'} & \text{if } s'\in S^{\mathcal{D}}+[n] \end{cases}$$

$$P^{(\mathcal{A}\wr\mathcal{C})\oplus(\mathcal{B}\wr\mathcal{D})}(s^{\mathcal{D}},a,s') = \begin{cases} P^{\mathcal{D}}(s^{\mathcal{D}},a,s') & \text{if } s'\in S^{\mathcal{D}}+[n] \\ 0 & \text{otherwise} \end{cases}$$

From the above equations we can determine that:

$$P^{(\mathcal{A}\oplus\mathcal{B})\fatsemi(\mathcal{C}\oplus\mathcal{D})}(s^{\mathcal{A}}, a, s') = P^{(\mathcal{A}\fatsemi\mathcal{C})\oplus(\mathcal{B}\fatsemi\mathcal{D})}(s^{\mathcal{A}}, a, s')$$
$$P^{(\mathcal{A}\oplus\mathcal{B})\fatsemi(\mathcal{C}\oplus\mathcal{D})}(s^{\mathcal{B}}, a, s') = P^{(\mathcal{A}\fatsemi\mathcal{C})\oplus(\mathcal{B}\fatsemi\mathcal{D})}(s^{\mathcal{B}}, a, s')$$
$$P^{(\mathcal{A}\oplus\mathcal{B})\fatsemi(\mathcal{C}\oplus\mathcal{D})}(s^{\mathcal{C}}, a, s') = P^{(\mathcal{A}\fatsemi\mathcal{C})\oplus(\mathcal{B}\fatsemi\mathcal{D})}(s^{\mathcal{C}}, a, s')$$
$$P^{(\mathcal{A}\oplus\mathcal{B})\fatsemi(\mathcal{C}\oplus\mathcal{D})}(s^{\mathcal{D}}, a, s') = P^{(\mathcal{A}\fatsemi\mathcal{C})\oplus(\mathcal{B}\fatsemi\mathcal{D})}(s^{\mathcal{D}}, a, s')$$

This proves 3.

Since all three conditions are met, we can conclude that the bifunctoriality relation holds. □

A.2 Proofs of Convertibility of Extensions

A.2.1 Repeat

Proof. We will prove by induction that for any $n \geq 1$, it holds that:

$$\mathcal{A}^n = \mathcal{R}(\mathcal{A}, n)$$

Base Case
In the base case, $n = 1$. We get:

$\mathcal{A}^1 = \mathcal{R}(\mathcal{A}, 1)$
$\mathcal{A} = \mathcal{A}$

Induction Hypothesis
We assume that for some $k > 1$ it holds that:

$$\mathcal{A}^k = \mathcal{R}(\mathcal{A}, k)$$

Induction Step
We need to show that $\mathcal{A}^{k+1} = \mathcal{R}(\mathcal{A}, k+1)$
$\mathcal{A}^{k+1} = \mathcal{R}(\mathcal{A}, k+1)$

$$\begin{aligned}
\mathcal{A}^k \fatsemi \mathcal{A} &= \mathcal{R}(\mathcal{A}, k) \fatsemi \mathcal{A} && \text{(by the definition of } \mathcal{R}) \\
&= \mathcal{A}^k \fatsemi \mathcal{A} && \text{(by the IH)}
\end{aligned}$$

The above proves by induction that for any $n \geq 1$, it holds that $\mathcal{A}^n = \mathcal{R}(\mathcal{A}, n)$. □

A.2.2 Switch

We need to prove: $[m_1]\mathcal{A}[m_2] = S_1 \fatsemi \mathcal{A} \fatsemi S_2$

For the above to hold we need to show that:

1. $[m_1]S^{\mathcal{A}}[m_2] = S^{S_1 \fatsemi \mathcal{A} \fatsemi S_2}$
2. $E^{\mathcal{A}}(i)[m_1] = E^{\mathcal{A} \fatsemi S_1}(i)$ and $E^{\mathcal{A}}(i)[m_2] = E^{\mathcal{A} \fatsemi S_2}(i)$
3. $[m_1]P^{\mathcal{A}}[m_2] = P^{S_1 \fatsemi \mathcal{A} \fatsemi S_2}$

$$\begin{aligned}
[m_1]S^{\mathcal{A}}[m_2] &= S^{\mathcal{A}} && \text{(mapping does not add/remove states)} \\
S^{S_1 \fatsemi \mathcal{A} \fatsemi S_2} &= S^{S_1} \uplus S^{\mathcal{A}} \uplus S^{S_2} && \text{(by the definition of } \fatsemi) \\
&= \emptyset \uplus S^{\mathcal{A}} \uplus \emptyset && \text{(by the definition of switch)} \\
&= S^{\mathcal{A}}
\end{aligned}$$

This proves 1.

$$
\begin{aligned}
E^{\mathcal{A}}(i)[m_1] &= m_1(E^{\mathcal{A}}(i)) && \text{(mapping is applied after applying E on each i)}\\
E^{\mathcal{A}\fatsemi S_1}(i) &= E^{S_1}(E^{\mathcal{A}}(i)) && \text{(by the definition of } \fatsemi)\\
&= m_1(E^{\mathcal{A}}(i)) && \text{(by the definition of switch)}
\end{aligned}
$$

$$
\begin{aligned}
E^{\mathcal{A}}(i)[m_2] &= m_2(E^{\mathcal{A}}(i))\\
E^{\mathcal{A}\fatsemi S_2} &= E^{S_2}(E^{\mathcal{A}}(i))\\
&= m_2(E^{\mathcal{A}}(i))
\end{aligned}
$$

We have shown that $E^{\mathcal{A}}(i)[m_1] = E^{\mathcal{A}\fatsemi S_1}(i)$ and $E^{\mathcal{A}}(i)[m_2] = E^{\mathcal{A}\fatsemi S_2}$, which proves 2.

$$
[m_1]P^{\mathcal{A}}[m_2](s^{\mathcal{A}}, a, s') = \begin{cases} P^{\mathcal{A}}(s^{\mathcal{A}}, a, s') & \text{if } s' \in S^{\mathcal{A}} \\ 0 & \text{otherwise} \end{cases}
$$

$$
\begin{aligned}
P^{S_1\fatsemi\mathcal{A}\fatsemi S_2}(s^{S_1}, a, s') &= \ ! && \text{(by the definition of switch)}\\
P^{S_1\fatsemi\mathcal{A}\fatsemi S_2}(s^{\mathcal{A}\fatsemi S_2}, a, s') &= \begin{cases} P^{\mathcal{A}\fatsemi S_2}(s^{\mathcal{A}\fatsemi S_2}, a, s') & \text{if } s' \in S^{\mathcal{A}\fatsemi S_2} + [n] \\ 0 & \text{otherwise} \end{cases}\\
P^{S_1\fatsemi\mathcal{A}\fatsemi S_2}(s^{\mathcal{A}}, a, s') &= \begin{cases} P^{\mathcal{A}}(s^{\mathcal{A}}, a, s') & \text{if } s' \in S^{\mathcal{A}} \\ \sum_{i\in[k]} P^{\mathcal{A}}(s^{\mathcal{A}}, a, i)\cdot\delta_{E^{S_2}(i)=s'} & \text{if } s' \in S^{S_2} + [n] \end{cases}\\
&= \begin{cases} P^{\mathcal{A}}(s^{\mathcal{A}}, a, s') & \text{if } s' \in S^{\mathcal{A}} \\ 0 & \text{otherwise} \end{cases}\\
&\text{(because delta is always zero since } S_2 \text{ has no states)}
\end{aligned}
$$

$[m_1]P^{\mathcal{A}}[m_2](s^{\mathcal{A}}, a, s') = P^{S_1\fatsemi\mathcal{A}\fatsemi S_2}(s^{\mathcal{A}}, a, s')$, which proves 3.

We have shown that all three conditions hold, so we can conclude that $[m_1]\mathcal{A}[m_2] = S_1 \fatsemi \mathcal{A} \fatsemi S_2$.