ST-Structures

Christian Johansen

*Department of Informatics – University of Oslo, Norway

Abstract

The present paper defines ST-structures (and an extension of these, called STC-structures). The main purpose is to provide concrete relationships between highly expressive concurrency models coming from two different schools of thought: the higher dimensional automata, a state-based approach of Pratt and van Glabbeek; and the configuration structures and unrestricted event structures, an event-based approach of van Glabbeek and Plotkin. In this respect we make comparative studies of the expressive power of ST-structures relative to the above models. Moreover, standard notions from other concurrency models are defined for ST-structures, like steps and paths, bisimilarities, and action refinement, and related results are given. These investigations of ST-structures are intended to provide a better understanding of the state-event duality described by Pratt, and also of the (a)cyclic structures of higher dimensional automata.

Contents

1 Introduction 2
   1.1 Positioning ST-structures wrt. other concurrency models . . . . . 3
2 ST-structures 4
3 Expressiveness of ST-structures and correspondences 16
   3.1 Correspondence with configuration structures . . . . . . . . . . . . . 16
   3.2 Correspondence with the event structures of Plotkin and van Glabbeek 25
   3.3 Correspondence with higher dimensional automata . . . . . . 29
   3.4 Correspondence with Chu spaces . . . . . . . . . . . . . . . . . . . . 43
4 Action refinement for ST-structures 44
5 Expressiveness shortcomings through examples 50

✩This article is a full version of the extended abstract presented at the Nordic Workshop on Programming Theory (NWPT) 2013 in Tallinn.

∗The address of the author (née Cristian Prisacariu): P.O. Box 1080 Blindern, 0316 Oslo, Norway.

Email address: cristioifi.uio.no (Christian Johansen)
1. Introduction

The geometric model of concurrency, studied by Pratt and van Glabbeek [1, 2, 3], is of high expressive power, thus providing a general framework for studying the differences and common features of various other models of concurrency (as done in [2] and [4]). This model was named Higher Dimensional Automata (HDA) by Pratt [1]. An attractive aspect of HDA is the automata-like presentation [3], which emphasizes the state aspect of the modelled system (and transitions between states). This aspect is opposed to the event-based models of concurrency, like (prime, flow, (non-)stable) event structures [5, 6, 7] or configuration structures and unrestricted event structures [8, 9].

We see the notion of configuration (in its various guises [5, 9, 10]) as fundamental to event-based models. The configuration structures, introduced in [8], are a rather general model of concurrency based on sets of events (forming the configurations of the modelled system). A thorough study of the generality and expressiveness of configuration structures is carried out in [9] where relations with general forms of event structures are made (where the pure event structures are the more well behaved, instances of which are found in the literature). The configuration structures lend themselves easily to action refinement, as studied in [11], which makes them an ideal candidate for incremental development of concurrent systems where the system architect starts with an abstract model which is subsequently refined to more concrete instances.

We are interested in studying such models based on sets of events, but in relation to the state-based model of higher dimensional automata. This study of event-state duality is argued for by Pratt [12], and the model of Chu spaces has been developed in response [13, 14]. Here we take the challenge of Pratt, with insights from Chu spaces, and develop models based on sets of events, in the spirit of van Glabbeek and Plotkin [9]. We call this model ST-structures. We investigate the expressiveness and relationships of this new model with the ones we mentioned above (i.e., with unrestricted event structures and configuration structures of [8] and with the triadic event structures, Chu spaces, and HDA of Pratt). We also investigate how definitions that one finds for configuration structures are extended to this new setting. In particular, we define for ST-structures a hereditary history preserving bisimulation, which in [11] is the most expressive equivalence presented for configuration structures. We also investigate the notion of action refinement (and properties of this) for ST-structures.

We point out shortcomings in the expressiveness of the ST-structures using examples. We then present an extension with the notion of cancellation, advocated by Pratt [15]. In this extension, called STC-structures, we are able to investigate closer the HDA with cycles. This extended model opens the way to tackling the problem posed by Pratt in [2] about the expressive power of HDAs with cycles wrt. event-based models.

The notion of an ST-configuration has been used in [16] to define ST-bisimulation and in [3] in the context of HDA. But the model of ST-structures,
as we define here for capturing concurrency, does not appear elsewhere. We think that a main characteristic of higher dimensional automata is captured by ST-structures, opposed to the standard configuration structures; this is the power to look at the currently executing concurrent events (not only observe their termination). In other words, we can now talk about what happens during the concurrent execution of one or more events. This is opposed to standard models that talk only about what happens after the execution (which may have duration and complex structure, apparent only after subsequent refinements of an initial abstract model).

1.1. Positioning ST-structures wrt. other concurrency models

Besides having the potential to help understand better the state-event duality, the (a)cyclic structures of HDAs, or the “during” aspect of events, ST-structures could also help correlate better the concurrency models that we compare with in Section 3. We focus on those models which are most relevant, i.e.: configuration structures [8], unrestricted event structures [9], acyclic higher dimensional automata [1, 3], and Chu spaces [14]. Nevertheless, thorough studies of expressiveness and interrelations of the above models with other existing models done in works like [4, 9, 11, 16, 2, 15, 3] can help position ST-structures wrt. those other models as well.

The results from Section 3 are not meant to just show expressiveness of ST-structures, but more to study the relations between the properties, the translation maps, and what exactly differentiates ST-structures from the other models. These studies are also meant to understand better the particularities of the models from Section 3 wrt. to each other. Of special interest are the higher dimensional automata and their way of capturing events or cycles.

With the risk of being too cryptic for a less expert reader, we shortly present here some of these findings, and kindly ask the reader to return to these paragraphs after a first go through the rest of this paper.

ST-structures extend configuration structures with the notion of “during” as opposed to only talking about “what happens after” an event. Therefore, in ST-structures we can see when an event, or several concurrent ones, are currently being executed, as well as when the event has terminated its execution. More technically, configuration structures are those ST-structures where the “during” is irrelevant, i.e., where the events are always terminated. When computational steps are also considered, together with the property of connectedness, then configuration structures impose that concurrency squares are always “filled in”. This suggests that configuration structures are not so adequate for distinguishing between the interleaving and concurrency of two events (not labels), because the interleaving square cannot be empty.

---

1I am thankful for having been made aware of the invited talk of Rob van Glabbeek at CONCUR’99 [16] where it is mentioned (at the end of Sec.1) as future work the investigation, on the same lines as the work of [8], of “translations between arbitrary Petri nets and ST-structures, showing that also these models are equally expressive” ; nevertheless, their recent work [6] does not present such an investigation yet.
In response, we see when comparing ST-structures with the unrestricted event structures of [9] that the latter can capture the empty interleaving squares, and thus do right justice to the distinction that Pratt advocates for, between the notion of concurrency and interleaving, at the level of events even, not only at the level of the labels. Still, for unrestricted event structures we see that the concurrency steps will always be fully filled in. This is made obvious by the property of ST-structures called “adjacent-closure” (which, as the name suggests, is close related to the similar notion of adjacency for HDAs). But we show that there are quite natural examples that fall outside the adjacent-closure property of ST-structures, and thus justify the full ST-structures.

All these three event-based models, including ST-structures, are not cyclic and can capture cycles only up to unfoldings. Still, there are cycles from HDAs that elude ST-structures even under unfolding, thus suggesting that more expressiveness is needed. For this we define STC-structures which show promise in this direction.

In the event-based models configurations are capturing the state of the system. For configuration structures these correspond to only the corners of a HDA, i.e., cells of dimension 0, whereas the ST-configurations correspond to any higher dimensional cell in a HDA. Therefore, we may expect that when looking at acyclic HDAs to find the ST-structures. In particular, the property of adjacent-closure of ST-structures corresponds to HDAs being non-degenerate (i.e., having for a cell/cube all the faces nicely in place). But it turns out that neither of these can be embedded in the other. For HDAs it is easy to see because ST-structures only capture unfoldings. For the opposite, it turns out that the classic way of identifying events in HDAs (as parallel boundaries of a square) is not enough to express the way ST-structures can work with events.

It is known that configurations structures correspond to the Chu spaces over 2. The extension with “duration” that ST-structures represent in turn corresponds to the extension of Chu spaces to being defined over 3, since the structure 3 is the extension of 2, which is the simple “start < terminate”, to the “start < during < terminate”. It could be interesting to look at Chu spaces as isomorphic to ST-structures through the results of Section 3.

2. ST-structures

We define ST-structures, showing in Section 3 that they are a natural extension of configuration structures 2, and define related notions that stem from the latter. The classical notions of concurrency, causality, and conflict are not interdefinable as in the case of event structures or stable configuration structures; but are more loose, as is the case with HDAs. In Section 3 we relate ST-structures also to HDAs by identifying a corresponding class of ST-structures, i.e., with the particular property of adjacent-closure. We also define the class of stable ST-structures and relate this with their counterpart in stable configuration structures. We define the (hereditary) history preserving bisimulation in the context of ST-structures, which when stability is imposed on adjacent-closed ST-structures it corresponds to the same bisimulation for stable configuration
structures. In Section 4 we define action refinement for ST-structures and investigate properties of it, like being preserved under the above bisimulation, or that it preserves the properties of the refined ST-structures.

**Definition 2.1 (ST-configuration).** An ST-configuration over some set $E$ of events is a pair of finite sets $(S, T)$ (i.e., $S, T \subseteq E$) respecting the property:

\[(\text{start before terminate}) \quad T \subseteq S.\]

Intuitively $S$ contains the events that have started and $T$ the events that have terminated. Therefore, we see the events in $S \setminus T$ as being executed concurrently in the current ST-configuration. We call their number $|S \setminus T|$ the concurrency degree of this ST-configuration. An event can be started (i.e., running) but not terminated yet. This notion of seeing events while running but not terminated, is a key aspect captured by ST-structures, and it is also found in HDAs, but not in classical event-based models like configuration structures or event structures.

Define the dimension of an ST-configuration to be $|(S, T)| = |S| + |T|$. Unions and intersections of ST-configurations are considered pairwise.

The notion of ST-configuration is very close to that of [16, Def.2.4] only that here we do not presuppose the existence of a partial order, but we define it later, in the same style as done for configuration structures [9].

**Definition 2.2 (ST-structures).** An ST-configuration structure (also called ST-structure) is a tuple $ST = (E, ST, l)$ with $ST$ a set of ST-configurations over $E$ satisfying the constraint:

\[\text{if } (S, T) \in ST \text{ then } (S, S) \in ST, \tag{1}\]

and $l : E \to \Sigma$ a labelling function with $\Sigma$ the set of labels. We often omit the set of events $E$ from the notation when there is no danger of confusion.

The constraint (1) above is a closure, ensuring that we do not represent events that are started but never terminated. The set of all ST-structures is denoted $ST$.

**Definition 2.3 (stable ST-structures).** An ST-structure $(ST, l)$ is called:

1. rooted iff $(\emptyset, \emptyset) \in ST$;
2. connected iff for any non-empty $(S, T) \in ST$, either $\exists e \in S : (S \setminus e, T) \in ST$ or $\exists e \in T : (S, T \setminus e) \in ST$;
3. closed under bounded unions iff for any $(S, T), (S', T') \in ST$ if $(S, T) \cup (S', T') \subseteq (S'', T'')$ then $(S, T) \cup (S', T') \in ST$;
4. closed under bounded intersections iff for $(S, T), (S', T') \in ST$ if $(S, T) \cup (S', T') \subseteq (S'', T'')$ then $(S, T) \cap (S', T') \in ST$.

\[\text{2This constraint could be weakened, as done in Definition 5.3 of STC-structures, and discussed in the extended version [18], but the implications on the results of this paper are not clear.}\]
An ST-structure is called stable iff it is rooted, connected, and closed under bounded unions and intersections.

We define a computational interpretation for ST-structures by defining simple steps between ST-configurations. Other computational interpretations could be imagined, like the more complex steps defined for STC-structures in Definition 5.4. Our choice is similar to interpretations existing for other concurrency models, like configuration structures. We give results below, like Theorem 5.10 meant to show that this computational interpretation is more fine-grained than for other models that ST-structures are compared with. Intuitively, opposed to standard event-based models, the computational interpretation of ST-structures naturally captures the “during” aspect of the events, i.e., what happens while an event is executing (before it has finished). The model of HDAs do the same job but in the state-based setting. Moreover, action refinement and bisimulation are well behaved wrt. this interpretation. Besides, ST-structures exhibit a natural observable information (on the same lines as for HDAs) as ST-traces, which, cf. [3, Sec.7.3], constitute the best formalization of observable content for concurrent systems.

Definition 2.4 (ST steps). A step between two ST-configurations is defined as either:
s-step: \((S, T) \xrightarrow{a_s} (S', T')\) when \(T = T', S' = S \cup \{e\}\), \(e \notin S\), and \(l(e) = a\); or
t-step: \((S, T) \xrightarrow{a_t} (S', T')\) when \(S = S', T' = T \cup \{e\}\), \(e \notin T\), and \(l(e) = a\).

When the type is unimportant we denote a step by \(\xrightarrow{a}\) for \(\xrightarrow{a_s} \cup \xrightarrow{a_t}\).

Definition 2.5 (paths and traces). A path of an ST-structure, denoted \(\pi\), is a sequence of steps, where the end of one is the beginning of the next, i.e.,

\[ \pi \triangleq (S_0, T_0) \xrightarrow{a} (S_1, T_1) \xrightarrow{b} (S_2, T_2) \ldots \]

A path is rooted if it starts in \((\emptyset, \emptyset)\). The ST-trace of a rooted path \(\pi\), denoted \(st(\pi)\), is the sequence of labels of the steps of \(\pi\) where each label is annotated as \(a^0\) if it labels an s-step or as \(a^n\) if it labels a t-step, where \(n \in \mathbb{N}^+\) is determined by counting from the beginning the number of steps until the s-step that has added the event \(e\) to the \(S\) set, with \(e\) being the event that has been added to \(T\) in the current t-step.

For rooted and connected ST-structures the notion of ST-trace conforms with the one defined in [14, def.2.5] or [3, sec.7.3].

Proposition 2.6 (connectedness through paths).

\(^3\)We generally work with rooted paths.
1. For a rooted ST-structure $\text{ST}$ the following are equivalent:
   (a) $\text{ST}$ is connected;
   (b) For any $(S,T) \in \text{ST}$ there exists a rooted path ending in $(S,T)$.

2. For a rooted ST that is closed under bounded unions the following are equivalent:
   (a) $\text{ST}$ is connected;
   (b) For any two ST-configurations s.t. $(S,T) \subseteq (S',T')$, there exists a path starting in $(S,T)$ and ending in $(S',T')$.

Proof: To prove the implication $(1a) \Rightarrow (1b)$ use induction on the dimension of $(S,T)$ applying subsequently to smaller ST-configurations the connectedness property of Definition 2.3.2.

To prove the implication $(1b) \Rightarrow (1a)$ is easier by using the definition of a path which implies the connectedness Definition 2.3.2.

The proof of $(2a) \Rightarrow (2b)$ makes unions of the ST-configurations on the two rooted paths corresponding to $(S',T')$ respectively $(S,T)$. Since the paths evolve through simple steps (i.e., which remove one event at a time) and since $(S',T')$ includes $(S,T)$ we slowly reach configurations that include events not part of $(S,T)$. Union with these intermediate configurations will make up the configurations on the path we are looking for.

To prove the implication $(2b) \Rightarrow (2a)$ observe that $(2b)$ implies $(1b)$. $\square$

Proposition 2.7. For any ST-configuration $(S,T)$, all the rooted paths ending in $(S,T)$ have the same length.

Proof: Each single step adds one single new event to either the $S$ or the $T$ sets. Therefore, since the number of events in the goal ST-configuration $(S,T)$ is fixed, no matter the order of adding these events, there will be the same number of steps, or event addition operations, that can be performed from the root. $\square$

Definition 2.8 (concurrency and causality).

For a particular ST-configuration $(S,T) \in \text{ST}$ define the relations of concurrency and causality on the events in $S$ as:

concurrency for $e,e' \in S$ then $e \parallel e'$ iff exists $(S',T') \subseteq (S,T)$ s.t. $(S',T') \in \text{ST}$ and $\{e,e'\} \subseteq S' \setminus T'$;

causality for $e,e' \in S$ then $e < e'$ iff $e \neq e'$ and for any $(S',T') \subseteq (S,T)$ s.t. $(S',T') \in \text{ST}$, is the case that $e' \in S' \Rightarrow e \in T'$.

ST-structures represent concurrency in a way that is different than other event-based models in the sense that each ST-configuration gives information about the currently concurrent events, and this information is persistent throughout the whole execution. Two events are considered concurrent wrt. a particular ST-configuration if and only if at some point in the past (i.e., in some
sub-configuration) both events appeared as executing (i.e., in $S'$) and none was terminated yet (i.e., not in $T'$); they were both executing concurrently. In event structures or configuration structures in order to decide whether two events are concurrent one needs to look at many configurations or many events to decide this. For example, in event structures the concurrency is defined as not being dependent nor conflicting; which requires to inspect all configurations to decide. An ST-configuration does not give complete information about the concurrency relation in the whole system. In consequence one could view the information about concurrency that an ST-configuration provides as being sound but not complete.

The above two notions of concurrency and causality are defined for one particular ST-configuration; in consequence one could emphasize this by indexing the relation symbol by the particular ST-configuration (similar to what is done in [11, Sec.5.3]), but the aesthetics would not be so nice in our case.

This local definition of \textit{causality} for an ST-configuration is in the tradition of viewing causality as a partial order (in fact this definition makes a partial order on the ST-configuration when the ST-structure is rooted and connected).

In ST-structures we can look at when an event is started and when it terminates. In consequence, an intuitive understanding of causality is that an event $e$ \textit{is a cause of} $e'$ if and only if in all previous ST-configurations, $e'$ is never started without $e$ having terminated. In other words, whenever during the run of the system up to the current ST-configuration, the event $e'$ is supposed to be started (i.e., $e' \in S'$), any event $e$ on which it depends should have been terminated already (i.e., $e \in T'$).

The definition of event structures from [9] uses a \textit{dependency relation} that can capture the notion of \textit{disjunctive causality} (as opposed to the more common \textit{conjunctive causality} where one event depends on several events). Disjunctive causality is nicely exemplified by the “parallel switch of Winskel” (see Example 2.21 and Figure 2 on page 14) where an event $b$ is caused by either of the two events 0 or 1 having happened. ST-structures capture disjunctive causality when the above local definition is lifted to the whole ST-structure.

On arbitrary ST-structures the concurrency and causality are not interdefinable (in a standard way e.g. [11, Def.5.6] where concurrency is the complement of causality). Nevertheless, concurrency and causality are disjoint on every ST-configuration of an arbitrary ST-structure. For the more well behaved stable ST-structures the concurrency and causality are interdefinable. Even more, results similar to the ones in [11, Sec.5.3] can be stated and proven about stable ST-structures and their causality partial order.

\textbf{Proposition 2.9.} On arbitrary ST-structures

1. concurrency and causality are disjoint;
2. concurrency and causality are not interdefinable (in a standard way e.g. [11, Def.5.6] where concurrency is the negation of causality).

\textbf{Proof:} The counterexample for the second part of the proposition consists of
the empty square from Figure 1(right)

\[ \text{ST} = (\{a, b\}, \{(\emptyset, \emptyset), (a, \emptyset), (b, \emptyset), (a, a), (b, b), (ab, a), (ab, b), (ab, ab)\}) \]

with the upper right ST-configuration \((a, b), \{a, b\})\). For this configuration the two events \(a\) and \(b\) are not causal in any order, because of the existence of the two ST-configurations \((a, \emptyset)\) and \((b, \emptyset)\). But still, the two events \(a\) and \(b\) are not concurrent, because the ST-configuration \((ab, \emptyset)\) is missing. Moreover, the two events are not conflicting in the sense of the Definition 2.11 which also rules out interdefinability of concurrency as negation of causality and conflict (i.e., two events are concurrent when they are not conflicting nor causal in any order).

To show disjointness one can notice that if two events are concurrent then they cannot be causally depended in any order. The witnessing ST-configuration is exactly the configuration that witnesses the concurrency, i.e., the \((S, T)\) with \(\{e, e'\} \subseteq S \setminus T\). This configuration breaks the \(e < e'\) because \(e' \in S\) and \(e \notin T\); and analogous for \(e' < e\).

The counterexample from the proof of Proposition 2.9 shows how HDAs and ST-structures model a notion that is eluding the standard notions of causality, concurrency, or conflict. I would call this notion interleaving, as other authors do as well [2, 3], and consider it different than concurrency.

**Proposition 2.10.** Let \(\text{ST}\) be a stable ST-structure. For some \((S, T) \in \text{ST}\) and two events \(e, e' \in S\) we have:

\(e \parallel e'\) if not \((e < e'\) or \(e' < e)\).

**Proof:** Knowing that \(e \not< e'\) and \(e' \not< e\) we show the existence of some ST-configuration \((S', T') \subseteq (S, T)\) for which \(\{e, e'\} \subseteq S' \setminus T'\), hence that \(e \parallel e'\).

The two assumptions are equivalent to

- \(\exists(S_1, T_1) \subseteq (S, T) : e' \in S_1 \lor e \notin T_1\); and
- \(\exists(S_2, T_2) \subseteq (S, T) : e \in S_2 \lor e' \notin T_2\).
From the fact that $\mathsf{ST}$ is connected and closed under bounded unions, using Proposition 2.6 we know that $(S_1, T_1) \rightarrow^* (S, T)$ and $(S_2, T_2) \rightarrow^* (S, T)$. This means that from $(S_1, T_1)$ we can reach a configuration $(S'_1, T'_1) \subseteq (S, T)$ where $S'_1$ contains both $e, e'$ but still $e \notin T'_1$. The same for some $(S'_2, T'_2)$ s.t. $e, e' \in S'_2$ and $e' \notin T'_2$. This implies that $e, e' \in S'_1 \cap S'_2$, and that $e \notin T'_1 \cap T'_2$ and $e' \notin T'_1 \cap T'_2$. Because $\mathsf{ST}$ is closed under bounded intersections it means that we have found $(S'_1 \cap S'_2, T'_1 \cap T'_2)$ which is an ST-configuration of $\mathsf{ST}$ that is included in the original $(S, T)$ and which satisfies $\{e, e'\} \subseteq (S'_1 \cap S'_2) \setminus (T'_1 \cap T'_2)$.

The notion of conflicting events is not definable for a specific ST-configuration because it is a general notion definable only on the whole ST-structure. Essentially, conflicting events can never appear in the same configuration.

**Definition 2.11 (conflict).** For an ST-structure $\mathsf{ST}$ the notion of global conflict is defined as a predicate over sets of events $E' \subseteq E$:

$$\#E' \iff \exists(S, T) \in \mathsf{ST} \text{ with } E' \subseteq S.$$  

The standard notion of binary conflict is an instance of the above, where $E = \{e, e'\}$. Moreover, a particular ST-configuration cannot contain conflicting events.

**Proposition 2.12 (partial order causality).** The causality relation of Definition 2.8 when extended with equality is a partial order if the ST-structure $\mathsf{ST}$ from which the ST-configuration $(S, T)$ on which $<$ is defined, is rooted and connected.

**Proof:** Extend the causality relation with equality by defining

$$e \leq e' \overset{\Delta}{=} e < e' \lor e = e'.$$

Clearly $\leq$ is reflexive.

To prove that $\leq$ is transitive take three events $e_1 \leq e_2 \leq e_3$ and show $e_1 \leq e_3$. The proof is immediate when any two of the three events are equal. Thus work with the assumption $e_1 < e_2 < e_3$. By applying two times Definition 2.8 we have that $\forall(S', T') \subseteq (S, T)$ an ST-configuration of $\mathsf{ST}$ then $e_3 \in S' \Rightarrow e_2 \in T' \subseteq S' \Rightarrow e_1 \in T'$; thus having the desired result.

To prove antisymmetry assume $e_1 < e_2$ and $e_2 < e_1$. Applying two times the Definition 2.8 we get that for any $(S', T') \subseteq (S, T)$ that is an ST-configuration of $\mathsf{ST}$, we have $e_2 \in S' \Rightarrow e_2 \in T'$. But this contradicts the fact that $\mathsf{ST}$ is rooted and connected, which implies that there is a rooted path to $(S', T')$. Hence this path coming from the root through single steps must necessarily pass through an ST-configuration that has $e_2$ started but not terminated.

**Definition 2.13 (adjacent-closure).** We call an ST-structure $\mathsf{ST}$ adjacent closed if the following are respected:

1. if $(S, T), (S \cup e, T), (S \cup \{e, e'\}, T) \in \mathsf{ST}$, with $(e \neq e') \notin S$, then $(S \cup e', T) \in \mathsf{ST}$,
2. if $(S, T), (S \cup e, T), (S \cup e, T \cup e') \in ST$, with $e \not\in S \wedge e' \not\in T \wedge e \neq e'$, then $(S, T \cup e') \in ST$;

3. if $(S, T), (S \cup e, T), (S \cup e', T) \in ST : e \not\in S \wedge e' \not\in T \wedge e \neq e'$, then $(S \cup e, T \cup e') \in ST$;

4. if $(S, T), (S, T \cup e), (S, T \cup \{e, e'\}) \in ST$, with $e \neq e' \not\in T$, then $(S, T \cup e') \in ST$.

Anticipating the definition of higher dimensional automata (see [1, 2, 3] and the Definition 3.23 on page 30) one can see a correlation of the above definition of adjacent-closure on ST-structures and the cubical laws of higher dimensional automata. This correlation is even more visible in the definition of adjacency of Def.19 which is used to define homotopy over higher dimensional automata (see Definition 3.26 on page 32). Since homotopy classes essentially define histories, then the above adjacent-closure on ST-structures intuitively makes sure that the histories of ST-configurations are not missing anything.

Another way to see adjacent-closure is as requiring that all the sides and corners of a square exist. For an illustration, we can take the square of Figure 1(right) with the middle ST-configuration $(ab, \emptyset)$ also present and if we remove any of its sides or corners then we see that applying the adjacent-closure would add it back. See also the examples at the end of this section.

Definition 2.14 (closure under single events). An ST-structure ST is called closed under single events iff $\forall (S, T) \in ST, \forall e \in S \setminus T$ :

1. $(S, T \cup \{e\}) \in ST$ and
2. $(S \setminus \{e\}, T) \in ST$.

Proposition 2.15 (equivalent with adjacent-closure). A rooted and connected ST-configuration structure is closed under single events iff is adjacent-closed.

Proof : The left-to-right implication is simple. For the first condition in Def. 2.13 use the second restriction of this proposition. For the second condition we may use any of the two restrictions, as we know that $e' \in S \setminus T$. For the third and forth condition use the first restriction, knowing that $e' \in S \setminus T$.

The right-to-left implication is more involved.

We first use induction on the reachability path to show that:

for every ST-configuration $(S, T)$ with $|S \setminus T| \neq 0$ then all the immediately lower ST-configurations that can reach $(S, T)$ through an s-step exist in ST, i.e.,

$\forall e \in S \setminus T : (S \setminus \{e\}, T) \in ST$.

This would prove the second requirement for closure under single events.

Since the ST-structure that we work with is rooted and connected, then every ST-configuration is reachable from the root $(\emptyset, \emptyset)$ through a series of single steps, i.e., through a rooted path, cf. Proposition 2.6.
Because of Proposition 2.7 we can use induction on the reachability path, because there exists at least one such path, and any other path has the same distance.

Base step: is for reachability paths of distance 1. This means when \((S, T) = (\{e\}, \emptyset)\); trivial.

For the Induction case use the proof principle reductio ad absurdum and assume for some \(e \in S \setminus T\) the ST-configuration \((S \setminus \{e\}, T) \not\in \mathcal{ST}\). From connectedness we know that \((S, T)\) is reachable through either an s- or a t-step from an ST-configuration that has lower reachability distance.

Assume that \((S, T)\) is reachable through an s-step, thus \(\exists e' \in S \setminus T\ s.t \ (S \setminus \{e'\}, T) \in \mathcal{ST}\) and \(e \neq e'\). Since \(e \in (S \setminus \{e'\})\setminus T\) we may apply the induction hypothesis to \((S \setminus \{e'\}, T)\) to get that \((S \setminus \{e, e'\}, T)\). We can now apply the first adjacent-closure requirement of Definition 2.13 to get that \((S \setminus \{e\}, T)\), which is a contradiction.

Assume now that no s-steps are possible, and thus only a t-step is possible from some \((S, T \{f\})\) with \(f \not\in S \setminus T\), hence \(f \neq e\). By applying the induction hypothesis to \((S, T \{f\})\) we get that \((S \setminus \{e\}, T \{f\}) \in \mathcal{ST}\), since \(e \in (S \setminus T \{f\})\). We can now apply the second condition for adjacent-closure to get that \((S \setminus \{e\}, T) \in \mathcal{ST}\), which is a contradiction.

It remains to show that the first requirement of closure under single events is satisfied. Thus, for some arbitrary \((S, T)\) we use induction on the dimension of \(|S \setminus T|\) to show that

\[
\forall e \in S \setminus T : (S, T \cup e) \in \mathcal{ST}.
\]

We could also use induction on the reachability path (as before).

Base step is for \(|S \setminus T| = 1\), i.e., when \(S = T \cup e\) for some \(e \not\in T\). By the definition of ST-structures we have that for our \((T \cup e, T)\) there also exists \((T \cup e, T \cup e) \in \mathcal{ST}\).

For the inductive case, i.e., when \(\exists e, e' \in S \setminus T\) distinct, we know from the previous step of the proof that for all \(f \in S \setminus T\) we have \((S \setminus f, T) \in \mathcal{ST}\). Pick one of these which is different than \(e\), as at least one exists \(e' \neq e\). Since \(|(S \setminus e') \setminus T|\) is smaller than the initial \(|S \setminus T|\) we can apply the induction hypothesis to obtain that for \(e \in ((S \setminus e') \setminus T)\) we have \((S \setminus e', T \cup e) \in \mathcal{ST}\). We may now apply the third requirement in the definition of adjacent-closure to obtain that also \((S, T \cup e) \in \mathcal{ST}\).

One may assume to work with rooted and connected structures, not only because these are natural, but also because we can obtain them using the notion of reachability.

**Definition 2.16 (reachable part).** An ST-configuration \((S, T)\) is said to be reachable iff there exists a rooted path ending in \((S, T)\). The reachable part of some arbitrary ST-structure is formed of all and only the reachable configurations.
The reachable part of a structure is connected, cf. Prop. 2.6.1. Therefore, assuming connectedness is the same as assuming to work with the reachable part of a structure.

**Definition 2.17 (morphisms of ST-structures).** A morphism \( f : ST \to ST' \) between two ST-structures \( ST = (E, ST, l) \) and \( ST' = (E', ST', l') \) is defined as a partial function on the events, \( f : E \to E' \) which:

- preserves ST-configurations, if \( (S, T) \in ST \) then \( f(S, T) = (f(S), f(T)) \in ST' \),
- preserves the labelling when defined, i.e., \( l'(f(e)) = l(e) \) if \( f \) is defined for \( e \), and
- is locally injective and total, i.e., for any \( (S, T) \in ST \) the restriction \( f|_S \) is injective and total.

Note that if \( T \subseteq S \) then \( f(T) \subseteq f(S) \). Also note that without local totality the morphisms would not preserve steps, as the next proposition shows.

**Proposition 2.18.** The morphisms of Definition 2.17 preserve steps.

**Proof:** We prove that for a step \( (S, T) \xrightarrow{s} (S \cup e, T) \in ST \) then \( f(S, T) \) can make an s-step with the event \( f(e) \) into the corresponding ST-configuration \( f(S \cup e, T) \). Since \( (S \cup e, T) \in ST \) then \( f(S \cup e, T) \) is also a configuration in \( ST' \) and thus \( f \) is defined for \( e \). Since \( e \not\in S \) (by definition of an s-step) it means that \( e \) is different than any other event \( g \) from \( S \), and by the injective property of \( f \) it means that \( f(e) \not\in f(S) \). Moreover, the label is preserved. Therefore we have the s-step in \( ST' \) with the same label and the corresponding event, \( (f(S), f(T)) \xrightarrow{f(e)} (f(S) \cup f(e), f(T)) \).

We can define a category \( ST \) to have objects ST-structures and the morphisms from Definition 2.17 because composition of morphisms is well defined and for any ST-structure there exists a unique identity morphism which is the total function taking an event to itself.

**Definition 2.19 (isomorphic ST-structures).** A function \( f \) is an isomorphism of two ST-configurations \((S, T)f(S', T')\) iff \( f \) is a bijection between \( S \) and \( S' \) that agrees on the sets \( T \) and \( T' \) (i.e., \( f|_T = T' \)). Two ST-structures \( ST \) and \( ST' \) are isomorphic, denoted \( ST \cong ST' \), iff there exists a bijection \( f \) on their events that is also a morphism between the two ST-structures.

**Definition 2.20 (hh-bisimulation for ST-structures).**

For two ST-structures \( ST \) and \( ST' \), a relation \( R \subseteq ST \times ST' \times \mathcal{P}(ST \times ST') \) is called a history preserving bisimulation between \( ST \) and \( ST' \) iff \( (\emptyset, \emptyset, \emptyset) \in R \) and whenever \( ((S, T), (S', T')), f) \in R \) then

1. \( f \) is an isomorphism between \((S, T)\) and \((S', T')\); and
2. if \((S, T) \sim (S_a, T_a)\) in \(ST\) then there exist \((S'_a, T'_a)\) in \(ST'\) and \(f'\) extending \(f\) (i.e., \(f'|_{(S, T)} = f\)) with \((S'_a, T'_a) \sim (S'_a, T'_a)\) and \(((S_a, T_a), (S'_a, T'_a), f') \in R\); and

3. if \((S', T') \sim (S'_a, T'_a)\) in \(ST'\) then there exist \((S_a, T_a)\) in \(ST\) and \(f'\) extending \(f\) with \((S, T) \sim (S_a, T_a)\) and \(((S_a, T_a), (S'_a, T'_a), f') \in R\).

\(R\) is moreover called hereditary if the following back condition holds:

4. if \((S_a, T_a) \sim (S, T)\) in \(ST\) then there exists \((S'_a, T'_a)\) in \(ST'\) and \(f'\) with \(f|_{(S_a, T_a)} = f'\) and \((S'_a, T'_a) \sim (S'_a, T'_a)\) and \(((S_a, T_a), (S'_a, T'_a), f') \in R\).

5. if \((S'_a, T'_a) \sim (S', T')\) in \(ST'\) then there exists \((S_a, T_a)\) in \(ST\) and \(f'\) with \(f|_{(S'_a, T'_a)} = f'\) and \((S_a, T_a) \sim (S, T)\) and \(((S_a, T_a), (S'_a, T'_a), f') \in R\).

A history preserving bisimulation between two \(ST\)-structures is denoted \(ST \sim ST'\), and a hereditary one is denoted \(ST \simh ST'\). We usually abbreviate to \(hh\)-bisimulation.

Because of symmetry of the requirements for history preserving bisimulation (i.e., the points 2 and 3 above), the two conditions for hereditary are redundant together, and we could well use only one of them. In our proofs we will consider only condition 4.

**Example 2.21.** The parallel switch of Winskel\[7, Ex.1.1.7\] consists of an event \(b\) (lighting a light bulb) that depends on either of the two parallel switches being closed (i.e., the two events 0 and 1). This example emphasizes disjoint causality, where event \(b\) depends on either 0 or 1, and hence the fact that there is no unique causal history, as opposed to stable structures. The \(ST\)-structure for this example, in Figure 2(left and middle), is adjacent-closed and closed under unions, but not closed under intersections, i.e., the \(ST\)-configurations \((01b, \emptyset) = (01b, 1) \cap (01b, 0) \subseteq (01b, 01b)\) but \((01b, \emptyset) \not\in STW\).
Figure 3: Example of a stable ST-structure that is not adjacent-closed.

The resolved conflict of [8, Ex.2], pictured in Figure 2(right), represents the fact that the initial conflict of the two actions $a$ and $b$ is resolved as soon as the action $c$ has finished (i.e., $a$ and $b$ may run concurrently as soon as $c$ has finished). The corresponding ST-structure of Figure 2(right) is adjacent-closed and closed under intersections but not closed under unions: $(bc, \emptyset) \cup (ac, \emptyset) = (abc, \emptyset) \not\in ST_{RC}$. Both examples can be pictured as three sides of a HDA cube (middle and right) whereas on (left) is an ST-structure. In several cases we use the more clean HDA presentation for ST-structures because of the results below (see the Definition 3.23 of a HDA). The standard example of a square with the empty inside, as pictured in Fig. 1(right) on page 9, is adjacent-closed but not closed under unions nor under intersections.

**Example 2.22.** For an example of an ST-structure that is not adjacent-closed but is stable consider the filled square of $a \parallel b$ but where the triangle above the diagonal is removed as in Fig. 3. Intuitively, this models a system where both $a$ and $b$ may run concurrently but $a$ is always faster than $b$ (hence starts and also terminates first); in other words $b$ cannot start before $a$ has started and cannot finish before $a$ has finished. The event $a$ may be a resource allocation mechanism that $b$ may need for running, and thus $a$ must be running when $b$ can start. But $b$ may run concurrently with $a$, e.g., while $a$ finishes all the resource allocation work (like logging or lock setting). Nevertheless, $b$ must wait for all this resource allocation work to properly finish (having all logging in place, etc.) before itself can finish (and maybe do some more logging and lock releasing).

These examples lead to the results in the next section where properties of ST-structures are correlated with properties of related known concurrency models, i.e.: stable configuration structures, unrestricted event structures, and the popular class of cyclic and non-degenerate HDAs. This last model is expressive enough to faithfully represent the various examples used in papers like [16, 11, 19, 20] which studied bisimulations for true concurrency. Nevertheless, in Section 5 we give examples that challenge the expressiveness of this class of HDAs and ST-structures as well. These justify an extension of ST-structures
introduced in Section 5 and which we will study more thoroughly in a future paper under the name of \textit{STC-structures}.

3. Expressiveness of ST-structures and correspondences

We give relations between ST-structures and three relevant concurrency models, i.e.:

1. with configurations structures \[8\] because ST-structures are a natural extension of these;
2. with the (unrestricted) event structures of Plotkin and van Glabbeek \[9\], because these are rather expressive event-based models that could be thought to challenge the alleged expressiveness that ST-structures are assumed to bring;
3. with higher dimensional automata because they are our motivation for studying ST-structures as an event-based model counterpart of HDAs.

The related works \[4, 9\] make comparisons between these existing models, i.e., \[4\] studies event-based models including Winskel-style event structures, Petri nets, and propositional theories; whereas \[4\] studies the relationships of HDAs to some of the standard true concurrency models, including Winskel-style event structures. Studies about relationships between true concurrency models can also be found in \[11\] where refinement and equivalence notions are thoroughly investigated, in \[16\] where splitting is investigated closely, in \[2, 15\] where Chu spaces are closely being correlated with, or in \[8\] where HDAs are related to models like Petri nets among other.

Logics for such models and their bisimulation distinguishing powers have been investigated in \[19, 20\] for event structures and configurations structures and in \[21, 22\] for HDAs.

3.1. Correspondence with configuration structures

We investigate the relationship of ST-structures with the configuration structures of \[8, 9\] and show that ST-structures are a natural extension of the later. This result holds also when their respective computational aspects are considered, i.e., the concurrent step interpretations are related.

**Definition 3.1 (cf. [11, Def.5.1][9, Def.1.1]).**

A configuration structure \(C = (E, C)\), is formed of a set \(E\) of events and a set of configurations which are subsets of events \(C \subseteq 2^E\). A labelled configuration structure also has a labelling function of its events, \(l : E \rightarrow \Sigma\).

**Definition 3.2 (morphisms for \(C\)).** A morphism between two labelled configuration structures \(C = (E, C, l)\) and \(C' = (E', C', l')\) is a partial map \(f : E \rightarrow E'\) between their events that:

- preserves the configurations; i.e., if \(X \in C\) then \(f(X) \in C'\),

---

16
• preserves the labelling when defined, i.e., \( l'(f(e)) = l(e) \) if \( f \) is defined for \( e \), and

• is locally injective, i.e., for any \( X \in C \) the restriction \( f|_X \) is injective.

Two configurations structures are called isomorphic, denoted \( C \cong C' \), iff there exists a morphism \( f \) that is bijective on the events.

The set of labelled configurations together with the morphisms form a category, which we will denote by \( C \).

**Definition 3.3 (C to ST).** Define a mapping \( ST : C \to ST \) that associates to every configuration structure \( C \) an ST-structure \( ST(C) \) as follows. Associate to each configuration \( X \in C \) an ST-configuration \( ST(X) = (X, X) \in ST(C) \). No other ST-configurations are part of \( ST(C) \). The labelling function is just copied.

**Proposition 3.4.** The mapping \( ST \) can be extended to a functor between the categories \( C \) and \( ST \) by defining its application on the morphisms as \( ST(f) = f \).

**Proof:** Trivial. (see technical report [18]) \( \square \)

**Proposition 3.5 (ST is embedding).**

1. The map \( ST \) from Definition 3.3 preserves isomorphic configuration structures and does not identify non-isomorphic configuration structures.

2. There are ST-structures that are not the image of any configuration structure.

**Proof:** For embedding it is easy to see that for any two configuration structures the function \( ST_{C_1, C_2} : Hom_C(C_1, C_2) \to Hom_{ST}(ST(C_1), ST(C_2)) \) that associates to each morphism \( f : C_1 \to C_2 \) the morphism \( ST(f) \), is bijective.

To show that \( ST \) preserves isomorphic configuration structures consider the bijective morphism \( f : E_1 \to E_2 \) that witnesses the isomorphism of \( C_1 \) and \( C_2 \). This same function between the events of \( ST(C_1) \) and \( ST(C_2) \) is also a bijection and a morphism between the two ST-structures.

To show that non-isomorphic configuration structures are not identified by \( ST \) for some arbitrary \( C_1 \not\cong C_2 \) assume that there exists a bijective morphisms \( f \) witnessing the isomorphism of their translations \( ST(C_1) \cong ST(C_2) \). It is easy to show that this same function between the events of \( C_1 \) and \( C_2 \) makes these isomorphic as it preserves configurations, the labelling and is locally injective.

For the part (2) of the proposition just take any ST-structure that also has ST-configurations of concurrency degree non-zero, i.e., some \( (S, T) \) with \( |S \setminus T| > 0 \); these do not correspond to the image through \( ST \) of any configuration structure. \( \square \)
Definition 3.6 (ST to C). Define a mapping $C : ST \rightarrow C$ that associates to every ST-structure $ST$ a configuration structure by keeping only those ST-configurations that have $S = T$; i.e., $C(ST) = \{ T | (S, T) \in ST \land S = T \}$, which preserves the labelling.

Proposition 3.7. If an ST-structure $ST$ is rooted, connected, or closed under bounded unions, or intersections, then the corresponding $C(ST)$ is respectively rooted, connected, closed under bounded unions, or intersections.

Proof: A configuration structure is rooted if it contains the configuration $\emptyset$. The definitions of bounded union and intersection for configuration structures are the natural simplification of the respective definitions for ST-structures from Definition 2.3. Proving the rootedness and closure properties is immediate. A configuration structure is connected (cf. [11, Def.5.5]) if for every configuration $X$, there exists an event $e \in X$ s.t. $X \setminus \{e\}$ is also a configuration in the structure. For connectedness note that any connected ST-structure is also rooted (and the same holds for configuration structures). Therefore, for any configuration $X \in C(ST)$ there exists the ST-configuration $(X, X) \in ST$ from which it was obtained. Since ST is connected it means that there is a sequence of ST-configurations, each one event smaller than the previous, which reach the root $(\emptyset, \emptyset)$. This means that on this sequence there must eventually be an ST-configuration $(X \setminus e, Y)$ with $Y \subseteq X \setminus e$. By the constraint of the ST-structures it means that also the ST-configuration $(X \setminus e, X \setminus e) \in ST$ and therefore also the configuration $(X \setminus e) \in C(ST)$.

But there is not a one to one correspondence between ST-structures and configuration structures because there can be several ST-structures that have the same configuration structure. Take the example from Figure 1 of one HDA square that is filled in and one that is not, seen as ST-structures as the right picture shows; both have the same set of corners and hence the same configuration structure. But the two ST-structures are not isomorphic and also not hh-bisimilar (in the sense of Definition 2.20).

Proposition 3.8 (C is forgetful).

1. The map $C$ from Definition 3.6 preserves isomorphic ST-structures.
2. The map $C$ may identify non-isomorphic ST-structures (in fact non-hh-bisimilar).

Proof: Part (1) is easy, similar to Proposition 3.5.

For part (2) take the empty square and the filled-in concurrency square examples. These two are translated in the same configuration structure; i.e., their corners only. But as ST-structures these two examples are not isomorphic and neither hh-bisimilar.

It is easy to see that the map $C$ can be lifted to a functor between $ST$ and $C$ the same as we did in Proposition 3.4.
Next we show that the asynchronous concurrent step interpretation of configuration structures is captured by ST-structures (cf. Def. 2.1), an asynchronous step is defined between two configurations $X \rightarrow^C Y$ iff $X \subseteq Y$ and $\forall Z: X \subseteq Z \subseteq Y \Rightarrow Z \in C$).

**Lemma 3.9.**

1. Morphisms of $C$ preserve asynchronous concurrent steps.
2. Morphisms of $ST$ preserve (s-/t-)steps.

**Proof:** For part (1) we take an arbitrary $f : C_1 \rightarrow C_2$ and an arbitrary step $X \rightarrow^C Y \in C_1$ and show that $f(X) \rightarrow^C f(Y)$. From the definition of asynchronous steps ($X \subseteq Y$) and the local injectivity of $f$ we get $f(X) \subseteq f(Y)$. Moreover, the injectivity on the larger set $Y$ makes $f$ bijective between $Y$ and $f(Y)$ which means that any subset of $f(Y)$ is the image of some subset of $Y$. The asynchronous step says that all subsets $X \subseteq Z \subseteq Y$ are configurations $Z \in C_1$ and since $f$ preserves configurations it means that $f(Z) \in C_2$. These are all possible subsets $f(X) \subseteq Z' \subseteq f(Y)$, therefore we have the expected step $f(X) \rightarrow^C f(Y)$.

The proof of part (2) is easy since the steps in ST-structures involve single events. The proof again uses the local injectivity of morphisms. $\blacksquare$

**Theorem 3.10.** Define a mapping $ST_2 : C \rightarrow ST$ by extending the one in Definition 3.3 s.t. for each asynchronous step $X \rightarrow^C Y \in C$ add also an ST-configuration $ST_2(X \rightarrow^C Y) = (Y, X) \in ST_2(C)$. This map $ST_2$ preserves the asynchronous concurrent steps of the configuration structure, i.e., for each asynchronous step $X \rightarrow^C Y \in C$ there is a chain of single steps in the ST-structure $ST_2(C)$ that passes through $(Y, X)$ (thus signifying the concurrent execution of all events in $Y \setminus X$).

**Proof:** Take $C$ to be some configuration structure and $ST_2(C)$ the corresponding ST-structure that we construct for it. The construction extends the simple encoding from before which associated with each configuration $X \in C$ an ST-configuration $ST_2(X) = (X, X) \in ST_2(C)$. The function $ST_2(\cdot)$ is applied to the configurations of $C$ and does not introduce new events. Thus the labelling of the structures is just copied.

We show that for any $X \rightarrow^C Y \in C$ we have $ST_2(X) \rightarrow^* (Y, X) \rightarrow^* ST_2(Y)$ in $ST_2(C)$. We do this using induction on the number of concurrent events in the concurrent step between the configurations.

The base case is for $|Y \setminus X| = 1$ (we ignore the reflexive steps that are assumed for each configuration in $C$). Essentially, in terms of HDAs, the $ST_2$ adds also the transition between the two states of the HDA. In $ST_2(C)$ we have one s-step from $(X, X)$ to $(Y, X)$ and one t-step from $(Y, X)$ to $(Y, Y)$, where $Y = X \cup \{e\}$ as $\{e\} = Y \setminus X$.

Take $|Y \setminus X| = n \geq 2$, thus $\exists e \neq e' \in Y \setminus X$. We use the property of asynchronous steps in configuration structures from Def. 2.1 which says that
if $X \rightarrow_C Y$ then $\forall Z : X \subseteq Z \subseteq Y \Rightarrow Z \in \mathcal{C}$. This also implies that there are asynchronous steps from $X \rightarrow_C Z$ and $Z \rightarrow_C Y$, and both have fewer number of concurrent events. We can apply the inductive hypothesis in the following two instances: (1) $|Y \setminus e \setminus X| = n - 1$; and (2) $|Y \setminus (X \cup e')| = n - 1$. From (1) we get the chain of single steps $(X, X) \rightarrow^*(Y \setminus e, X) \rightarrow^*(Y \setminus e, Y \setminus e)$. Since $(Y, X) \in \text{ST}_2(\mathcal{C})$ was added by $\text{ST}_2(X \rightarrow_C Y)$ we have $(Y \setminus e, X) \rightarrow (Y, X)$. By the induction hypothesis on (2) we have $(X \cup e', X \cup e') \rightarrow^*(Y, X \cup e') \rightarrow^*(Y, Y)$. Because $e' \in Y$ and $e' \notin X$ we have also the transition $(Y, X) \rightarrow (Y, X \cup e')$. Thus we have the conclusion that there exists the chain of single steps $(X, X) \rightarrow^*(Y \setminus e, X) \rightarrow (Y, X) \rightarrow (Y, X \cup e') \rightarrow^*(Y, Y)$ that passes through $(Y, X)$.

Intuitively, thinking in terms of acyclic HDA$s$, for each transition $X \rightarrow_Y Y \in \mathcal{C}$ we build the HDA cube of dimension $|Y \setminus X|$ with all the faces filled in.

\begin{proof}

Since we work with rooted configurations structures, the $\text{ST}_2$ function clearly preserves rootedness.

From a connected configuration structure for any sequence of transitions $X \rightarrow Y$ we find a sequence of single steps in the associated ST-structure. This is easy to see from Theorem 3.10. Each individual transition has a corresponding sequence of single steps in the ST-structure.

From the proof of Theorem 3.10 we see that an ST-configuration $(Y, X)$ with $X \neq Y$ is introduced only when there is a concurrent transition between the configurations $X$ and $Y$. With this observation it is easy to prove the four adjacency restrictions. Take as example the first restriction (leaving the others as exercise) and infer from $(S, T)$ that there is the transition $T \rightarrow S$ and thus from \cite{[6], Def.2.1} it means that $\forall X : T \subseteq X \subseteq S$ hence $X \in \mathcal{C}$ is also a configuration. Also we have $T \rightarrow S \cup e$ and $T \rightarrow S \cup \{e, e'\}$. To prove that $(S \cup e', T) \in \text{ST}$ it is enough to show that there is a transition $T \rightarrow S \cup e'$. This is easy from \cite{[6], Def.2.1} and the fact that $S \cup e' \subseteq S \cup \{e, e'\}$.

It is easy to check that the parallel switch of Winskel \cite{[7]} (not closed under bounded intersections) and the resolved conflict example of \cite{[9], Ex.2} (not closed under bounded unions) are expressible as configuration structures.

\end{proof}

**Corollary 3.11.** An ST-structure $\text{ST}_2(\mathcal{C})$ generated as in Theorem 3.10 is adjacent-closed (though not necessarily closed under bounded unions nor bounded intersections).

**Proof:** Since we work with rooted configurations structures, the $\text{ST}_2$ function clearly preserves rootedness.

From a connected configuration structure for any sequence of transitions $X \rightarrow Y$ we find a sequence of single steps in the associated ST-structure. This is easy to see from Theorem 3.10. Each individual transition has a corresponding sequence of single steps in the ST-structure.

From the proof of Theorem 3.10 we see that an ST-configuration $(Y, X)$ with $X \neq Y$ is introduced only when there is a concurrent transition between the configurations $X$ and $Y$. With this observation it is easy to prove the four adjacency restrictions. Take as example the first restriction (leaving the others as exercise) and infer from $(S, T)$ that there is the transition $T \rightarrow S$ and thus from \cite{[6], Def.2.1} it means that $\forall X : T \subseteq X \subseteq S$ hence $X \in \mathcal{C}$ is also a configuration. Also we have $T \rightarrow S \cup e$ and $T \rightarrow S \cup \{e, e'\}$. To prove that $(S \cup e', T) \in \text{ST}$ it is enough to show that there is a transition $T \rightarrow S \cup e'$. This is easy from \cite{[6], Def.2.1} and the fact that $S \cup e' \subseteq S \cup \{e, e'\}$.

It is easy to check that the parallel switch of Winskel \cite{[7]} (not closed under bounded intersections) and the resolved conflict example of \cite{[9], Ex.2} (not closed under bounded unions) are expressible as configuration structures.

\begin{corollary}

In an ST-structure $\text{ST}_2(\mathcal{C})$ generated as in Theorem 3.10 the ST-configurations with $S = T$ correspond exactly to the configurations of $\mathcal{C}$. That is to say that $\mathcal{C}(\text{ST}_2(\mathcal{C})) \cong \mathcal{C}$, where the first $\mathcal{C}$ is from Definition 3.8.

\end{corollary}

**Proposition 3.13.** The new map $\text{ST}_2$ from Theorem 3.10 can be lifted to a functor by defining its application on morphisms to be $\text{ST}_2(f) = f$. This is the right adjoint to the functor $\mathcal{C}$.
Proof: Translating configuration structures into ST-structures does not change the set of events nor the labelling function, therefore it is easy to see that $\text{ST}_2(f)$ preserves the labelling.

To show that $\text{ST}_2(f)$ preserves the ST-configurations consider some $(S, T) \in \text{ST}_2(C_1)$ and take two cases cf. the definition of $\text{ST}_2$ from Theorem 3.10.

Case for $S = T$. From Theorem 3.10 we deduce that $(S, S)$ comes from the fact that $S \in C_1$ is a configuration. Since $f$ preserves configurations it means that $f(S) \in C_2$ and thus we have the desired result that $(f(S), f(S)) \in \text{ST}_2(C_2)$, i.e., $f(S, T) \in \text{ST}_2(C_2)$.

Case for $S \neq T$. From Theorem 3.10 we deduce that $(S, T)$ comes from a transition $T \rightarrow_C S \in C_1$. This means that $S, T \in C_1$ are configurations preserved by $f$, hence $f(T), f(S) \in C_2$. Since by Lemma 3.9 $f$ preserves also asynchronous steps we have the step $f(T) \rightarrow_C f(S) \in C_2$ which implies that this is translated into the ST-configuration $(f(S), f(T)) \in \text{ST}_2(C_2)$, i.e., $f(S, T) \in \text{ST}_2(C_2)$.

It is easy to see that $\text{ST}_2(f)$ is locally injective.

To show that $\text{ST}_2$ is right adjoint to $C$ we exhibit the co-unit $\epsilon_C : C \circ \text{ST}_2 \rightarrow I_C$ to be the isomorphism from Corollary 3.12.

We have to show that for any object $C$ of $\mathbb{C}$ and any morphism $g : C(\text{ST}) \rightarrow C$ in $\mathbb{C}$ there exists a unique morphism $g^\# : \text{ST} \rightarrow \text{ST}_2(C)$ for which the diagram on the right commutes. The map $C$ preserves events then the events of ST are the same as those of $C(\text{ST})$: the same holds for $\text{ST}_2$ meaning that the events of $C$ are the same as the events of $\text{ST}_2(C)$. Therefore, we can take $g^\#$ to be $g$, and the functor returns $C(g^\#) = g^\# = g$.

It is easy to see that the diagram commutes: for any $e \in E_{C(\text{ST})}$ we have that $g(e) = \epsilon_C \circ g(e)$ because the isomorphism $\epsilon_C$ from Corollary 3.12 is the identity.

To show uniqueness of $g^\#$ assume the existence of another $f : \text{ST} \rightarrow \text{ST}_2(C)$ for which the diagram commutes and which is different than $g^\#$ for some event $e \in E_{C(\text{ST})}$, i.e., $f(e) \neq g^\#(e)$. This means that $f(e) \neq g(e)$ and that $C(f)(e) \neq g(e)$. But then the composition with the co-unit would again result in $\epsilon_C \circ C(f)(e) \neq g(e)$, i.e., a contradiction.

\[\square\]

Corollary 3.14 (filled-in). The ST-structure obtained in Theorem 3.10 is “filled in”, in the sense that any cube is filled in. By a “cube” is meant an initial ST-configuration $(S, S)$, a final $(S \cup X, S \cup X)$, where $X$ is a nonempty set of events, together with all the ST-configurations $(Y, Y)$ from the subsets $S \subseteq Y \subseteq S \cup X$. To be “filled in” means that the intermediate ST-configuration $(S \cup X, S)$ exists.

Proof: We call “corners” the ST-configurations where $S$ and $T$ are equal. The subsets $S \subseteq Y \subseteq S \cup X$ form the corners of the cube. The definition of
a “cube” between some \((S, S)\) and \((S \cup X, S \cup X)\) implies that all the corners of the cube come from configurations \(S \subseteq S \cup Y \subseteq S \cup X\). This means that in the configuration structure there exists the asynchronous step \(S \rightarrow_c S \cup X\).

Therefore, by the definition of ST\(_2\) from Theorem 3.10, this asynchronous step is translated into the ST-configuration \((S \cup X, S)\).

\[ \square \]

**Proposition 3.15.** For stable and adjacent-closed ST-structures and stable configuration structures there is a one-to-one correspondence (the adjacency is necessary) given by the map \(C\) from Definition 3.6 and the following map \(\text{ST}_3 : C \rightarrow \text{ST}\) defined as:

1. associates to each configuration \(X \in C\) an ST-configuration \(\text{ST}_3(X) = (X, X) \in \text{ST}_3(C)\);
2. for each pair of configurations \(T\) and \(T \cup \{e\}\) add also the intermediate ST-configuration \((T \cup \{e\}, T)\);
3. then close the resulting ST-structure under bounded unions and intersections.

We give the proof of this proposition after proving a few helper results. Essentially, \(\text{ST}_3\) extends the simple \(\text{ST}\) from Definition 3.3 in the style of \(\text{ST}_2\) from Theorem 3.10 but where only single steps are considered and applying the closures at the end.

Since the input \(\text{ST}\) is stable, by Proposition 3.7, the resulting configuration structure \(C(\text{ST})\) is also stable. For configuration structures the property of being stable is defined as for ST-structures, i.e., as being rooted, connected and closed under bounded unions and intersections.

**Lemma 3.16.** The ST-structure \(\text{ST}_3(C)\) is stable if \(C\) is stable.

**Proof:** The structure \(\text{ST}_3(C)\) is rooted, and also closed under bounded unions and intersections, by its definition. We need to show it is connected.

Since the generated ST-structure is rooted, then by Proposition 2.6 we can show instead that all ST-configurations are reachable through a rooted path.

From the definition of \(\text{ST}_3\) and the connectedness of the input structure \(C\), we see that all ST-configurations of concurrency degree 0 or 1 are reachable. It remains to show that any ST-configuration of concurrency degree more than 1 is reachable; these are those ST-configurations coming from the closures.

We use induction on the dimension of the ST-configurations that enter the union (or intersection). We can safely assume that the closure applies the union first to the smallest ST-configurations. Since the closure starts from ST-configurations of concurrency degree 0 and 1, which are reachable, then we work with the assumption that for two reachable ST-configurations \((S, T)\) and \((S', T')\) which enter the requirements of the closure under bounded unions, then their union \((S \cup S', T \cup T')\) is also reachable. This would imply that any ST-configuration produced through the closure under bounded unions is reachable.
Take \((S, T)\) to be reachable from some \((S' \setminus e, T')\), i.e., through an \(s\)-step (the argument for a \(t\)-step is analogous). By the closure it means that also the union \(((S' \setminus e) \cup S', T \cup T')\) is an ST-configuration of \(\text{ST}_3(C)\) having dimension one smaller, coming from two ST-configurations reachable through shorter paths. Therefore this has been obtained through an earlier closure operation, implying the existence of a rooted path reaching this smaller \(((S' \setminus e) \cup S', T \cup T')\). But from this we can make an \(s\)-step, with the event \(e\) to reach the initial union ST-configuration \((S \cup S', T \cup T')\), thus finding the desired rooted path.

For the closure under bounded intersections a similar inductive reasoning on the length of the path goes through.

Lemma 3.17. \(\text{ST}_3(C)\) respects the constraint from Definition 2.2 of being an ST-structure.

Proof: It is not difficult to check that the claim holds for ST-configurations of concurrency degree 1.

For any two \((S, T), (S', T')\) satisfying the constraint, i.e., \(\exists (S, S'), (S', S') \in \text{ST}_3(C)\), we show that their union ST-configuration also respects the constraint. Respecting the conditions for closure under bounded unions means that there exists \((S'', T'')\) s.t. \((S \cup S', T \cup T') \subseteq (S'', T'')\) and \((S'', T'')\) also satisfies the constraint of Definition 2.2, i.e., \((S'', S'') \in \text{ST}_3(C)\). But in this case we see that the two ST-configurations \((S, S'), (S', S')\) respect too the conditions of closure under bounded unions, which implies that their union is an ST-configuration also \((S \cup S', S \cup S') \in \text{ST}_3(C)\), which is our desired result.

Lemma 3.18. The map \(\text{ST}_3(C)\) does not introduce corners \((X, X) \in \text{ST}_3(C)\) which do not have a corresponding \(X \in C\).

Proof: Any new corners can come only from unions or intersections. Assume two \((S, T), (S', T')\) that respect the conditions for closure under bounded unions and their union is a new corner \((S, T) \cup (S', T') = (S'', S'')\). But by the previous lemma there exist also the ST-configurations \((S, S)\) and \((S', S')\) which are both smaller than (i.e., included in) \((S'', S'')\). To these the inductive hypothesis says that are not new, but come from \(C\), i.e., \(S, S' \in C\). Since \(C\) is also closed under bounded unions it means that \(S \cup S' \in C\), our desired result.

Proof of Proposition 3.15 To show the one-to-one correspondence we show two results.

Claim: For a stable configuration structure \(C\) the application of the two association functions from above results in an isomorphic configuration structure:

\[
C \cong C(\text{ST}_3(C)).
\]

This result is easy to establish because, intuitively, the first map \(\text{ST}_3\) adds information which is then forgotten by the application of \(C\). It is easy to see that
$\text{ST}_3$ does not introduce new events; and the same for $C$. Therefore exhibiting the isomorphism is done by the identity function between the events of $C$ and $C(\text{ST}_3(C))$. We need to show that it preserves configurations, which means to show that for any configuration $X \in C$ then the same configuration is found in the right structure, i.e., $Id(X) = X \in C(\text{ST}_3(C))$. Any configuration is translated into a corner $(X, X) \in \text{ST}_3(C)$. By the previous claim, no other corners exist. Then each corner is translated into an appropriate configuration $X \in C(\text{ST}_3(C))$.

**Claim:** For a stable and adjacent-closed ST-structure $\text{ST}$ the application of the two association functions above results in an isomorphic ST-structure:

$$\text{ST} \cong \text{ST}_3(C(\text{ST})).$$

Note that only requiring $\text{ST}$ to be stable is not enough for this result. A counterexample is given by the stable ST-structure from Fig. 6 which is not adjacent closed and for which the above isomorphism is not the case.

The proof has two parts: first we show that any ST-configuration $(S, T) \in \text{ST}$ has an isomorphic version in $\text{ST}_3(C(\text{ST}))$; second is to show that the function applications $\text{ST}_3(C(\cdot))$ does not introduce new ST-configurations.

For the first part, if $S = T$ then it is easy to see that $(S, T) \in \text{ST}_3(C(\text{ST}))$.

When $S \neq T$ then let $E = S \setminus T$. Because the input $\text{ST}$ is stable (hence rooted and connected) and adjacent-closed it means it is closed under single events, cf. Proposition 2.15. Therefore, $(S \setminus e, T) \in \text{ST}$ for all $e \in E$. With a simple inductive argument using the above closure under single events one can easily show that $\forall X : T \subseteq X \subseteq S$ we have $(X, T) \in \text{ST}$. Therefore, together with the requirement on ST-structures that $(X, X)$ exists for any $(X, T)$, it means we have all configurations $X \in C(\text{ST})$, for $T \subseteq X \subseteq S$. By the definition of the association function $\text{ST}_3(\cdot)$ for all pairs of configurations $T \cup e$ and $T$, for all $e \in E$, the function adds an ST-configuration $(T \cup e, T) \in \text{ST}_3(C(\text{ST}))$. When closing under bounded unions all these ST-configurations we obtain the desired $(T \cup E, T) \in \text{ST}_3(C(\text{ST}))$.

For the second part, assume some $(S, T) \in \text{ST}_3(C(\text{ST}))$ then we show that $(S, T) \in \text{ST}$. If $S = T$ then this ST-configuration must come from a configuration $S \in C(\text{ST})$ (cf. the previous claim about no new corners), which in turn only comes from an ST-configuration $(S, S) \in \text{ST}$.

Assume $(S, T)$ comes from the existence of two configurations $T$ and $T \cup e$ in $C(\text{ST})$; i.e., $(S, T) = (T \cup e, T)$. But this means that in ST there exist the ST-configurations $(T, T)$ and $(T \cup e, T \cup e)$. From the fact that ST is stable it means that $(T \cup e, T \cup e)$ is reachable from $(\emptyset, \emptyset)$ through a path of single event steps. Assume we do not remove the event $e$ from the second set of the pair immediately (for otherwise we already have our desired result) and thus there is a series of single steps that remove single events different than $e$ gradually, first removing from the second set. But eventually we must reach a point when we remove $e$ from the second set and not from the first set yet. This means we reach an ST-configuration $(S', T')$ with $S' \subseteq T \cup e$ containing $e$, and $T' \subseteq T$. We
can apply the property of closed under bounded unions for \((T,T)\) and \((S',T')\) to obtain \((T \cup e,T)\).

Assume that \((S, T)\) comes from closure under bounded union of two smaller \((S', T')\) and \((S'', T'')\). By induction these are in \(\mathbf{ST}\) which is closed under bounded unions, hence it also contains \((S, T)\). The basis of the induction is essential here. We check it for \(\mathbf{ST}\)-configurations of concurrency degree 1.

Take two \((T' \cup e, T')\), \((T'' \cup f, T'')\) \(\in \mathbf{ST}\_3(C(\mathbf{ST}))\) and show that their union is \((T' \cup e \cup T'' \cup f, T' \cup T'') \in \mathbf{ST}\). By the previous argument we know that \((T' \cup e, T'), (T'' \cup f, T'') \in \mathbf{ST}\) which because \(\mathbf{ST}\) is closed under bounded unions delivers the expected result.

The results in this section also apply to pure event structures because these are shown in [9, Th.2 and Prop.2.2] to be equivalent to configuration structures under their respective computational interpretations, i.e., asynchronous steps are preserved through translations.

3.2. Correspondence with the event structures of Plotkin and van Glabbeek

We relate the \(\mathbf{ST}\)-structures with the (unrestricted) event structures of [9, Def.1.3] and the asynchronous transition relation associated to them in [9, Def.2.3]. An event structure (which we call unrestricted since their definition in [9, Def.1.3] is different from standard event structures and also the restriction of being pure is not imposed) is \(E = (E, \vdash)\), a set of events with an enabling relation defined between sets of events \(\vdash \subseteq 2^E \times 2^E\). An event structure can be associated with its set of configurations, cf. [9, Def.1.4], \(L(E) = \{X \subseteq E \mid \forall Y \subseteq X, \exists Z \subseteq X : Z \vdash Y\}\). Asynchronous transitions between these configurations are then defined in [9, Def.2.3] as \(X \rightarrow_E Y\) if \(X \subseteq Y\) and \(\forall Z \subseteq Y, \exists W \subseteq X : W \vdash Z\).

Denote by \(E\) the set of all such unrestricted event structures.

**Theorem 3.19 (\(E\) to \(\mathbf{ST}\)).** An unrestricted event structure can be encoded into an \(\mathbf{ST}\)-structure s.t. any asynchronous concurrent step transition (cf. [9, Def.2.3]) \(X \rightarrow_E Y\) is matched by an appropriate path that passes through the \(\mathbf{ST}\)-configuration \((Y, X)\). The encoding is done with the mapping \(\mathbf{ST} : E \rightarrow \mathbf{ST}\) defined similarly to the one for configuration structures of Theorem 3.10, considering the set \(L(E)\) of left-closed configurations [9, Def.1.4] of the event structure; i.e., \(\mathbf{ST}(X) = (X, X)\) for \(X \in L(E)\) and for any transition \(X \rightarrow_E Y\) add also the \(\mathbf{ST}\)-configuration \(\mathbf{ST}(X \rightarrow_E Y) = (Y, X)\).

The property that the theorem requires on the generated \(\mathbf{ST}\)-structure captures the concurrency that the event structure transition embodies.

**Proof:** The proof uses induction on the dimension of the asynchronous transitions, i.e., on \(|Y \setminus X|\), noting the fact that \(X \subseteq Y\). The proof is similar to what we did in Theorem 3.10 and is facilitated by Corollary 3.20.

The basis for \(|Y \setminus X| = 1\) is easy, for \(Y = X \cup \{e\}\).

The induction case for \(|Y \setminus X| \geq 2\) means we can consider two different events \(e \neq e' \in Y \setminus X\). After we prove that \(Y \setminus \{e\}\) and \(X \cup \{e'\}\) are also part of \(L(E)\) we can use Corollary 3.20 two times, with \(X \subseteq X \subseteq Y \setminus \{e\} \subseteq Y\) and
with \( X \subseteq X \cup \{ e' \} \subseteq Y \subseteq Y \), to get asynchronous transitions of shorter length respectively \( X \to_X Y \setminus \{ e \} \) and \( X \cup \{ e' \} \to_X Y \), which we can use inductively.

To show that \( (Y \setminus \{ e \}) \in L(E) \) we must show that \( \forall Z \subseteq (Y \setminus \{ e \}) : \exists W \subseteq (Y \setminus \{ e \}) : W \vdash Z \). This can be shown from the existence of the transition \( X \to_X Y \) which says that \( \forall Z \subseteq Y \), hence for our \( Z \subseteq (Y \setminus \{ e \}) \) also, there exists \( W \subseteq X \) with the property \( W \vdash Z \). But since \( X \subseteq (Y \setminus \{ e \}) \) we have found the \( W \) that we needed.

A similar argument is carried to prove that \( X \cup \{ e' \} \in L(E) \).

Now having the transition \( X \to_X Y \setminus \{ e \} \) of lower dimension we can apply the inductive hypothesis to obtain that there is a sequence of single steps in the ST-structure \( ST(E) \) between the ST-configurations \( (X, X) \to^* (Y \setminus e, X) \to^* \) \( (Y \setminus e, Y \setminus e) \). The existence of \( (Y, X) \) is guaranteed by the construction, thus having also a single step \( (Y \setminus e, X) \to (Y, X) \). Using induction with the other asynchronous transition we get that \( (Y, X \cup \{ e' \}) \to^* (Y, Y) \). Thus, we get the sequence of single steps we were looking for because \( (Y, X) \to (Y, X \cup \{ e' \}) \). \( \square \)

**Corollary 3.20 (from \([3, \text{Def.2.3}]\)).** For some transition \( X \to_X Y \) all the intermediate smaller transitions exist (i.e., going between any two subsets \( X \subseteq S \subseteq S' \subseteq Y \)).

**Proof:** By intermediary smaller transitions we mean the transitions that go between some two subsets \( X \subseteq S \subseteq S' \subseteq Y \). Thus, knowing that \( X \to_X Y \) we prove that \( S \to_X S' \). From \([3, \text{Def.2.3}]\) of the step transition relation \( \to_X \) on unrestricted event structures we have that: \( X \subseteq Y \) and \( \forall Z \subseteq Y : \exists W \subseteq X : W \vdash Z \). We prove that \( \forall Z' \subseteq S' : \exists W' \subseteq S : W' \vdash Z' \). We have that all \( Z' \subseteq S' \subseteq Y \) and therefore \( \exists W \subseteq X : W \vdash Z' \), and since \( X \subseteq S \) we found our \( W' \subseteq S \) to be \( W \). \( \square \)

Intuitively, one can view this last corollary as the opposite of the “filled in” property that was observed in Corollary 3.14 for the ST-structures produced from a configuration structure. The ST-structures associated to the unrestricted event structures are not “filled in”; i.e., the opposite direction of this last corollary does not hold.

**Proposition 3.21.** The ST-structures generated from unrestricted event structures as in Theorem 3.19 are adjacent-closed (and rooted and connected if the event structure is rooted and connected).

**Proof:** We show first rootedness and connectedness.

Assume the event structure \( E = (E, \vdash) \) is rooted, meaning that \( \emptyset \vdash \emptyset \), which is equivalent to \( \emptyset \in L(E) \), by the definition of left-closed configurations \( L(E) \) from \([3, \text{Def.1.4}]\). The translation function \( ST \) from Theorem 3.19 adds the ST-configuration \( (\emptyset, \emptyset) \in ST(E) \), therefore making it also rooted.

Assume now that \( E \) is connected, i.e., all \( X \in L(E) \) are reachable from the root \( \emptyset \) through a sequence of asynchronous steps \( X_1 \to_E \ldots \to_E X_n \) with \( X_1 = \emptyset \) and \( X_n = X \). But Theorem 3.19 says that for each of these steps there
exists a path \( \pi_i \) that goes from \((X_i, X_i)\) to \((X_{i+1}, X_{i+1})\), for \(1 \leq i < n\). These paths can be concatenated (in the right order) to obtain a path from \((0, 0)\) to \((X, X)\) thus making any ST-configuration from \(\text{ST}(E)\) that is of the form \((S, S)\) reachable.

Note now that the definition of \(\text{ST}\) from Theorem 3.19 adds one ST-configuration \((X, X)\) for each configuration \(X \in L(E)\) and one ST-configuration \((Y, X)\) for each asynchronous step from \(E\). It adds no other ST-configurations than these. This implies that any ST-configuration in \(\text{ST}(E)\) either comes from a configuration in \(L(E)\) or it comes from an asynchronous step in \(E\). It is, therefore, an easy consequence of Theorem 3.19 that for any ST-configuration \((Y, X)\) there is a path from \((X, X)\) to \((Y, Y)\) that passes through \((Y, X)\) (i.e., reaches it). Since \((X, X)\) is reachable from the root, then also \((Y, X)\) is reachable from \((0, 0)\). Thus we have connectedness for the rooted \(\text{ST}(E)\).

We check each adjacency condition from Definition 2.13.

1. Assuming \((S, T), (S \cup e, T), (S \cup \{e, e'\}, T)\) as ST-configurations it means that these should come from the transitions \(T \rightarrow_E S, T \rightarrow_E S \cup e, T \rightarrow_E S \cup \{e, e'\}\), where \(e \neq e'\) \(\notin S\). To prove the conclusion that \((S \cup e', T)\) is also an ST-configuration we prove that there is a transition \(T \rightarrow_E S \cup e'\).

By the definition we have that \(\forall X \subseteq S \cup \{e, e'\} : Y \subseteq T : Y \vdash X\); therefore it also holds that \(\forall X \subseteq S \cup e' : Y \subseteq T : Y \vdash X\), which is the desired transition.

2. Assuming \((S, T), (S \cup e, T), (S \cup e', T \cup e')\) as ST-configurations it means that these should come from the transitions \(T \rightarrow_E S, T \rightarrow_E S \cup e, T \cup e' \rightarrow_E S \cup e, T \cup e' \rightarrow_E S \cup e\), where \(e' \in S \setminus T\) and \(e \notin S\). To prove the conclusion that \((S, T \cup e')\) is also an ST-configuration we prove that there is a transition \(T \cup e' \rightarrow_E S\).

By the definition we have that \(\forall X \subseteq S : Y \subseteq T : Y \vdash X\); therefore it also holds that \(\forall X \subseteq S : \exists Y \subseteq T \cup e' : Y \vdash X\), which is the desired transition.

3. Assuming \((S, T), (S \cup e, T), (S, T \cup e')\) as ST-configurations it means that these should come from the transitions \(T \rightarrow_E S, T \rightarrow_E S \cup e, T \cup e' \rightarrow_E S \cup e, T \cup e' \rightarrow_E S\), where \(e' \in S \setminus T\) and \(e \notin S\). To prove the conclusion that \((S \cup e, T \cup e')\) is also an ST-configuration we prove that there is a transition \(T \cup e' \rightarrow_E S\).

We may use Corollary 3.20 to obtain the desired transition.

4. Assuming \((S, T), (S, T \cup e), (S, T \cup \{e, e'\})\) as ST-configurations it means that these should come from the transitions \(T \rightarrow_E S, T \cup e \rightarrow_E S, T \cup \{e, e'\} \rightarrow_E S\), where \(e' \in S\). To prove the conclusion that \((S, T \cup e')\) is also an ST-configuration we prove that there is a transition \(T \cup e' \rightarrow_E S\) using Corollary 3.20.

\(\square\)

**Proposition 3.22 (\(\text{ST} \text{ to } E\)).** Any rooted, connected, and adjacent-closed ST-structure can be translated into an unrestricted event structure s.t. the transitions of the event structure capture the concurrency embodied by the ST-structure. Define a map \(E : \text{ST} \rightarrow E\) which for \((E, \text{ST}, l)\) returns the event structure \((E, \vdash)\) with the labelling function copied and \(\vdash\) defined as:

1. \(\emptyset \vdash \emptyset\)

27
2. for each \((S \cup X, S) \in ST\) s.t. \(|X| > 0\) add 
\(S \vdash X' \cup Y\) for every \(Y \subseteq S\) and every \(X' \subseteq X\).

**Proof:** The translation of the ST-structure ensures that all the concurrency is captured in the resulting event structure, in the sense that if the ST-structure expresses that a set of events can be done concurrently, then there is a transition with that set of events in the generated event structure.

The ST-structure expresses that some set of events \(X\) are done in parallel whenever we have an ST-configuration where \(X \subseteq S \setminus T\) (cf. Definition 2.8).

The adjacent-closure ensures that Corollary 3.20 holds. Adjacent-closure also ensures that all the faces (including corners, i.e., those ST-configurations where \(S = T\)) of a cube (i.e., of concurrency degree \(|X| > 0\)) exist as ST-configurations.

The map \(E : ST \rightarrow E\) can be seen as taking all the corner ST-configurations into left-closed configurations of the event structure, similar to what we did in Definition 3.6 when embedding ST-structures into configuration structures, using the rest of the ST-configurations to build the enabling relation of the event structure s.t. the resulting transitions correspond exactly to those in the ST-structure (i.e., no new transitions are introduced).

Having \(\emptyset \vdash \emptyset\) ensures that the empty set is an admissible left-closed configuration of the event structure. Therefore the root of the ST-structure is translated into the root configuration of the event structure.

**Claim:** For any \((S, S) \in ST\) we have \(S\) as a left-closed configuration of \(E(ST)\).

A simple inductive argument based in the root configuration and using the dimension of the ST-configuration (and thus also of the length of the reachable rooted path) assumes a sequence of ST-configurations like \((S, S) \rightarrow (S \cup e, S) \rightarrow (S \cup e, S \cup e)\). All corner configurations are part of this pattern, because of the adjacent-closure and connectedness. For the middle configuration the map adds to \(\vdash\) the following: \(S \vdash e \cup Y\) for all \(Y \subseteq S\). The inductive hypothesis that \(S\) is left-closed in \(E(ST)\) means that \(\forall Y \subseteq S, \exists Z \subseteq S : Z \vdash Y\). Therefore, to show that \(S \cup e\) is a left-closed configuration of \(E(ST)\) we are left with verifying that for any \(X \subseteq (S \cup e)\) and \(X \not\subseteq S\) then \(\exists Z : Z \vdash X\). This is given by the above enabling pairs, where \(Z\) is always \(S\) for our case.

**Claim:** For any \((S \cup X, S) \in ST\) we have a step \(S \rightarrow_E S \cup X\) in \(E(ST)\).

Because of adjacent-closure the cube \((S \cup X, S) \in ST\) has the original corner \((S, S)\) which by connectivity is reachable through a rooted path. By an inductive argument on the length of the rooted paths we can assume that \(S\) is reachable through a step from some smaller \(S'\) (which can be also \(\emptyset\)), i.e., we have \(S' \rightarrow_E S \in E(ST)\). Therefore, to prove that \(S \rightarrow_E S \cup X\) we need to prove that \(\forall Z \subseteq S \cup X, \exists W \subseteq S : W \vdash Z\). From the fact that \(S\) is reachable it means that we have \(\forall Y \subseteq S, \exists W \subseteq S' \subseteq S : W \vdash Y\). To get the desired result we need to consider only those sets formed by adding some part of \(X\) to any of these
subsets \( Y \) of \( S \). But this is given by the map, i.e.: for the ST-configuration \((S \cup X, S)\) it adds \( S \vdash X' \cup Y \) for all \( Y \subseteq S \) and for all \( X' \subseteq X \).

We are creating redundancy in \( \vdash \), as the transition relation would require less pairs in the relation \( \vdash \). But this redundancy is artificial for our purpose of capturing the transition relations and the configurations of the two structures (i.e., unrestricted event structures and adjacent-closed ST-structures).

One can now check that no transitions are introduced in the event structure that are not present in the original ST-structure, nor new left-closed configurations.

**Claim:** For any \( X \) a left-closed configuration of \( E(ST) \) we have \((X, X) \in ST \).

This means that \( \forall Y \subseteq X. \exists Z \subseteq X : Z \vdash Y \). In particular, for \( X \) exists \( Z \subseteq X \) s.t. \( Z \vdash X \). By the definition of \( E(ST) \) this last enabling must come from some \((Z \cup A, Z) \in ST \) with \( X \) divided into \( X = B \cup C \) with \( C \subseteq Z \) and \( B \subseteq A \). This also says that \( A \cap Z = \emptyset \), hence \( B \cap Z = \emptyset \) and \( B \cap C = \emptyset \). From \( B \cap Z = \emptyset \), \( X = B \cup C \), and \( Z \subseteq X \) we have that \( Z \subseteq C \), and hence \( Z = C \). This means that the above ST-configuration is actually \((C \cup A, C) \in ST \) for which \( X = B \cup C \subseteq C \cup A \). Since ST is closed under single events and connected it means that we can remove the events in \( A \) to reach smaller ST-configurations; in particular we remove only the events in \( A \setminus B \) thus obtaining \((X, C) \in ST \). But by the property of ST we then have that also \((X, X) \in ST \).

**Claim:** For any \( X \rightarrow_E Y \) in \( E(ST) \) we have \((Y, X) \in ST \).

This means that \( X \subseteq Y \) and \( \forall Z \subseteq Y. \exists W \subseteq X : W \vdash Z \). In particular, for \( Y \) exists \( W \subseteq X \) s.t. \( W \vdash Y \). By the definition of \( E(ST) \) this last enabling must come from some \((W \cup A, W) \in ST \) with \( Y \) divided into \( Y = B \cup C \) with \( C \subseteq W \) and \( B \subseteq A \). This also says that \( A \cap W = \emptyset \), hence \( B \cap W = \emptyset \) and \( B \cap C = \emptyset \). From \( B \cap W = \emptyset \), \( Y = B \cup C \), and \( W \subseteq X \subseteq Y \) we have that \( W \subseteq C \), and hence \( W = C \). This means that the above ST-configuration is actually \((C \cup A, C) \in ST \) for which \( Y = B \cup C \subseteq C \cup A \) and hence \( C = W \subseteq X \subseteq Y \subseteq C \cup A \). This means that \( X \setminus C \subseteq A \). Since ST is closed under single events and connected it means that we can add the events from \( A \) to \( C \) to reach ST-configurations of smaller concurrency degree; in particular we add only the events in \( X \setminus C \) thus obtaining \((C \cup A, X) \in ST \). We can also remove elements from \( A \); in particular, removing the elements from \( A \setminus B \) we obtain now \((Y, X) \in ST \), i.e., the desired result. \( \square \)

### 3.3. Correspondence with higher dimensional automata

We recall the definition of higher dimensional automata (HDA) following the terminology of \([3, 13]\), defining also additional notions including the restriction to acyclic and non-degenerate HDAs. A logical study of HDAs can be found in \([21]\).

For an intuitive understanding of the HDA model consider the standard example \([13, 33]\) pictured in Figure 4 (middle-left). It represents a HDA that
models two concurrent events which are labelled by $a$ and $b$ (we can also have the same label $a$ for both events, giving rise to the notion of autoconcurrency). The HDA has four states, $q_0^4$ to $q_4^4$, and four transitions between them. This would be the classical picture for the interleaving concurrency, but in the case of HDA there is also a square $q_2$. Traversing through the interior of the square means that both events are executing. When traversing on the lower transition it means that event one is executing but event two has not started yet, whereas, when traversing through the upper transition it means that event one is executing and event two has finished already. In the states there is no event executing; in particular, in state $q_4^4$ both events have finished, whereas in state $q_1^0$ no event has started yet.

Similarly, HDAs allow to represent three concurrent events through a cube, or more events through hypercubes. Causality of events is modelled by sticking such hypercubes one after the other. For our example, if we omit the interior of the square (i.e., the grey $q_2$ is removed) we are left with a description of a system where there is the choice between two sequences of the same two events, i.e., $a; b + b; a$. This last interleaving choice example can be seen as obtained by sticking together four cubes of dimension 1 by identifying their endpoints; whereas the true concurrency example is just one single cube of dimension 2.

**Definition 3.23 (higher dimensional automata [3, Def.1]).**

A cubical set $H = (Q, \pi, \overline{\pi})$ is formed of a family of sets $Q = \bigcup_{n=0}^{\infty} Q_n$ with all sets $Q_n$ disjoint, and for each $n$, a family of maps $s_i, t_i : Q_n \to Q_{n-1}$, with $1 \leq i \leq n$, which respect the following cubical laws:

1. $\alpha_i \circ \beta_j = \beta_{j-1} \circ \alpha_i$, $1 \leq i < j \leq n$ and $\alpha, \beta \in \{s, t\}$.

In $H$, the $\pi$ and $\overline{\pi}$ denote the collection of all the maps from all the families (i.e., for all $n$). A higher dimensional automaton $(Q, \pi, \overline{\pi}, l, I, F)$ over an alphabet $\Sigma$ is a cubical set together with a labelling function $l : Q_1 \to \Sigma$ which respects $l(s_i(q)) = l(t_i(q))$ for all $q \in Q_2$ and $i \in \{1, 2\}$; and with $I \in Q_0$ initial and $F \subseteq Q_0$ final cells.
We call the elements of $Q_0, Q_1, Q_2, Q_3$ respectively states, transitions, squares, and cubes, whereas the general elements of $Q_n$ are called cells (also known as n-cells, n-dimensional cubes, or hypercubes). For a transition $q \in Q_1$ the $s_1(q)$ and $t_1(q)$ represent respectively its source and its target cells (which are states from $Q_0$ in this case). Similarly for a general n-cell $q \in Q_n$ there are $n$ source cells and $n$ target cells all of dimension $n - 1$.

Intuitively, an n-dimensional cell $q$ represents a configuration of a concurrent system in which $n$ events are performed at the same time, i.e., concurrently. A source cell $s_i(q)$ represents the configuration of the system before the starting of the $i^{th}$ event, whereas a target cell $t_i(q)$ represents the configuration of the system immediately after the termination of the $i^{th}$ event. We call all these source and target cells the faces of $q$. A cell of $Q_1$ represents a configuration of the system in which a single event is being performed. The cubical laws account for the geometry (concurrency) of the HDAs, with four kinds of cubical laws depending on the instantiation of $\alpha$ and $\beta$; Figure 3(left) presents one such instantiation.

**Definition 3.24 (isomorphism of HDAs).** A morphism between two HDAs, $f : H \rightarrow H'$ is a dimension preserving map between their cells $f : Q \rightarrow Q'$, such that:

1. the initial cell is preserved: $f(I) = I'$,
2. the labelling is preserved: $l'(f(q_1)) = l(q_1)$ for all $q_1 \in Q_1$,
3. the mappings are preserved, for any $q_n \in Q_n$ and $1 \leq i \leq n$:
   - $s_i(f(q_n)) = f(s_i(q_n))$ and
   - $t_i(f(q_n)) = f(t_i(q_n))$.

When a morphism is bijective we call it isomorphism. Two HDAs are isomorphic, denoted $H \cong H'$, whenever there exists an isomorphism between them.

The above definition of isomorphism conforms with that in [3, Def.2] whereas the definition of morphism conforms with that of [4, Sec.1.1].

We can define the category $\mathbb{HDA}$ to have objects HDAs and morphisms defined as in Definition 3.24. A thorough study of the category of HDAs was carried in [23].

**Definition 3.25 (paths in HDAs).** A single step in a HDA is either

1. $q_{n-1} \xrightarrow{s_1} q_n$ with $s_1(q_n) = q_{n-1}$ or
2. $q_n \xrightarrow{t_1} q_{n-1}$ with $t_1(q_n) = q_{n-1}$,

where $q_n \in Q_n$ and $q_{n-1} \in Q_{n-1}$ and $1 \leq i \leq n$.

A path $\pi \triangleq q^0 \xrightarrow{\alpha^1} q^1 \xrightarrow{\alpha^2} q^2 \xrightarrow{\alpha^3} \ldots$ is a sequence of single steps $q^i \xrightarrow{\alpha^{i+1}} q^{i+1}$, with $\alpha^j \in \{s, t\}$. We say that $q \in \pi$ iff $q = q^1$ appears in one of the steps in $\pi$. The first cell in a path is denoted $st(\pi)$ and the ending cell in a finite path is $en(\pi)$.
Note that the marking of the steps by \( s/t \) can be deduced from the fact that the step goes from a lower cell to a higher cell for \( s \)-steps (and the opposite for \( t \)-steps). It is though useful in many of the proofs to have easily visible the exact map (i.e., the index also) that the step uses, instead of explicitly assuming it every time. Also the label of a step can be deduced, following the definitions of [3, Sec.7.2], to be the label of the event that was started or terminated (i.e., the corresponding \( Q_1 \) cell; see later Definition 3.39). When the label is important we write it under the arrow as: \( \alpha_i \rightarrow \alpha_j \), with \( \alpha \in \{s, t\} \).

**Definition 3.26 (histories for HDA [3, Sec.7.4]).**

In a HDA two paths are adjacent, denoted \( \pi \leftrightarrow \pi' \), if one can be obtained from the other by replacing, for \( q, q' \in Q \) and \( i < j \),

1. a segment \( s_i \rightarrow q \rightarrow s_j \rightarrow q' \rightarrow s_j \), or
2. a segment \( t_j \rightarrow q \rightarrow t_i \rightarrow q' \rightarrow t_i \), or
3. a segment \( s_i \rightarrow q \rightarrow t_j \rightarrow q' \rightarrow s_i \), or
4. a segment \( t_j \rightarrow q \rightarrow t_i \rightarrow q' \rightarrow s_j \).

Two finite paths are \( l \)-adjacent \( \pi \leftrightarrow \pi' \) when the segment replacement happens at position \( l+1 \); i.e., \( q \) is the \( l+1 \) cell in the path. Homotopy is the reflexive and transitive closure of adjacency. Two homotopic paths are denoted \( \pi \leftrightarrow \pi' \) and share their respective start and end cells. The homotopy class of a rooted path is denoted \( \langle \pi \rangle \). A homotopy class with end cell \( q \) is said to be a history of \( q \). One cell may have several histories, as is the case with the interleaving square HDA from Figure 4. Whenever a cell has a unique history we use the notation \( \langle q \rangle \), instead of \( \langle \pi \rangle \) with \( \text{en}(\pi) = q \).

Above, homotopy is defined for all paths (opposed to the definition in [4, Sec.1.6]) and thus also a cell of higher dimension, like the inside of a square, has a history, not only the state cells of dimension 0 that form the corners of the square.

Inspired by the definition of history unfolding for process graphs from [24, Sec.3] we define the same notion for HDAs, which can also be correlated with the penultimate definition from [25]. Alternative definitions of unfolding for HDAs can be found in [26, 27].

**Definition 3.27 (history unfolding for HDAs).**

The history unfolding of a higher dimensional automaton \( H \) is a HDA denoted \( U(H) \), respecting the cubical laws, as Proposition 3.29 shows, and given by:

- \( Q_n^{U(H)} \) is the set of histories that end up in cells on level \( Q_n \) of \( H \);
- has the labelling copied from \( H \), i.e.: \( l^{U(H)}(\langle \pi \rangle) = l^H(\text{en}(\pi)) \);
- initial cell the empty rooted history;
• the $s/t$ maps are built from the corresponding maps between the end cells of the histories, i.e.:

\[ s_i(\pi) = \pi' \iff s_i(q) = q' \land \pi' \xrightarrow{\pi} \pi \land \text{en}(\pi') = q' \land \text{en}(\pi) = q; \]

\[ t_i(\pi) = \pi' \iff t_i(q) = q' \land \pi' \xrightarrow{\pi} \pi \land \text{en}(\pi') = q' \land \text{en}(\pi) = q. \]

**Remark 3.28.** Correlating with the penultimate definition from [25] is not difficult. In particular note that we work with homotopy classes, which are sets of paths that are homotopic, as in [24]. Because of this, we can identify a homotopy class with any of its paths (usually called the representative of the class), and thus pick any such path that suits us. This is particularly put to use in the last bullet-point of Definition 3.27. From the many paths of $\pi$ we pick those that end with an $s_i$ step; and $\pi'$ is one of these with the last step removed. The definition then returns the homotopy class of this $\pi'$.

**Proposition 3.29.** The history unfolding $U(H)$ respects the cubical laws.

**Proof:** For the history of some path $\pi$ we want to prove, for $i < j$:

\[ t_i(s_j(\pi)) = s_j(t_i(\pi)). \]

The left-hand-side implies the following: $t_i(s_j(\text{en}(\pi))) = \text{en}(\pi''_L)$ with $\pi'_L \xrightarrow{t_i} \pi''_L$ and $\pi'_L \xrightarrow{s_j} \pi$.

The right-hand-side implies: $s_j-1(t_i(\text{en}(\pi))) = \text{en}(\pi''_R)$ with $\pi \xrightarrow{t_i} \pi''_R$ and $\pi''_R \xrightarrow{s_j-1} \pi''_R$.

We want to prove that the two homotopy classes $\pi''_L$ and $\pi''_R$ are the same, for which we prove that the two $\pi''_L$ and $\pi''_R$ are part of the same homotopy class.

The cubical laws in $H$ imply that $\text{en}(\pi''_L) = \text{en}(\pi''_R)$.

For $\pi'_L \xrightarrow{s_j} \pi \xrightarrow{t_i} \pi''_R \in [\pi''_R]$ we can apply the last adjacency swap from Definition 3.26 to obtain that $\pi'_L \xrightarrow{t_i} \pi''_L \xrightarrow{s_j-1} \pi''_R \in [\pi''_R]$. But we also have that $\pi''_R \xrightarrow{s_j-1} \pi''_L \in [\pi''_L]$.

Similar arguments can be made for the other three types of cubical laws. \(\square\)

**Lemma 3.30.** For two homotopic paths $\pi \xrightarrow{\text{hom}} \pi'$, i.e., with $\text{en}(\pi) = \text{en}(\pi')$, that have the same last $s$-step, i.e., $\pi_1 \xrightarrow{s_i} \pi$ and $\pi'_1 \xrightarrow{s_i} \pi'$, then the shorter paths are also homotopic.

**Proof:** The proof of this result has been previously done in the topological presentation of HDAs using results from [28, 26, 29, 30]. We redo this result in the setting of this paper.
Assume $\pi \xrightarrow{\text{adj}} \pi'$ are adjacent. The adjacency swap cannot involve $s_i(en(\pi))$ because this is not allowed by the statement of the lemma. Since the adjacency happens lower along the paths, then removing the last step still gives adjacent paths (of the same l-adjacency).

Thus assume the paths enter a transitive series of l-adjacency swapping. If these swapping so not involve the last step, then, as before, after removing this step the same series of l-adjacency swaps can be applied to make the shorter paths homotopic.

The remaining question is whether the series of swaps can involve the last step, i.e., the cell $s_i(en(\pi)) = en(\pi_1) = en(\pi'_1)$. If this would be the case then we must first have a l-adjacency swap changing this cell, and later at the same point $l$ another l-adjacency swap that changes back to the same cell and s-map (so that the statement of the lemma is satisfied). One can check (albeit rather tedious) that whenever such a series exists then there exists also another series that does not involve the position $l$; thus reverting back to the previous paragraph.

\[\square\]

**Definition 3.31 (hh-bisimulation).** Two higher dimensional automata $H_A$ and $H_B$ (with $I_A$ and $I_B$ the initial cells) are hereditary history-preserving bisimulation equivalent (hh-bisimilar), denoted $H_A \xrightarrow{\text{hh}} H_B$, if there exists a binary relation $R$ between their paths starting at $I_A$ respectively $I_B$ that respects the following:

1. if $\pi_A R \pi_B$ and $\pi_A \xrightarrow{a} \pi'_A$ then $\exists \pi'_B$ with $\pi_B \xrightarrow{a} \pi'_B$ and $\pi'_A R \pi'_B$;
2. if $\pi_A R \pi_B$ and $\pi_B \xrightarrow{a} \pi'_B$ then $\exists \pi'_A$ with $\pi_A \xrightarrow{a} \pi'_A$ and $\pi'_A R \pi'_B$;
3. if $\pi_A R \pi_B$ and $\pi_A \xleftarrow{a} \pi'_A$ then $\exists \pi'_B$ with $\pi_B \xleftarrow{a} \pi'_B$ and $\pi'_A R \pi'_B$;
4. if $\pi_A R \pi_B$ and $\pi_B \xleftarrow{a} \pi'_B$ then $\exists \pi'_A$ with $\pi_A \xleftarrow{a} \pi'_A$ and $\pi'_A R \pi'_B$;
5. if $\pi_A R \pi_B$ and $\pi'_A \xrightarrow{a} \pi_A$ then $\exists \pi'_B$ with $\pi_B \xrightarrow{a} \pi_B$ and $\pi'_B R \pi'_B$;
6. if $\pi_A R \pi_B$ and $\pi'_B \xrightarrow{a} \pi_B$ then $\exists \pi'_A$ with $\pi_A \xrightarrow{a} \pi_A$ and $\pi'_A R \pi'_B$.

A corollary from [3] strengthens the above conditions 3 and 4 to unique existence.

**Corollary 3.32 (cf. [3, sec.7.5, prop.2]).** For a path $\pi$ and a point $l > 1$ there exists a unique path $\pi'$ that is l-adjacent with $\pi$.

Many of the results in this paper work with acyclic and non-degenerate HDAs in the following sense. Such HDAs are often considered in the literature on concurrent systems [31, 32, 33, 34] and are more general than most of the true concurrency models [15, 3].

**Definition 3.33 (acyclic and non-degenerate HDAs).**
A HDA is called acyclic if no path visits a cell twice. A HDA is called non-degenerate if for any cell $q$ all its faces exist and are different, in the sense of
$\forall i \neq j : \alpha_i(q) \neq \beta_j(q) \land \alpha, \beta \in \{s, t\}$, and no two transitions with the same label share both their end states.

The restriction on HDAs that we call here “non-degenerate” is close to that of Cattani and Sassone [35, Def.2.2], that of van Glabbeek [3, p.10], and the one defined in [20]. The second constraint of non-degeneracy is close to the notion of strongly labelled of [4, Def.1.13]. Note that the non-degeneracy still allows for two opposite s and t-maps to be equal, i.e., it is allowed $s_i(q) = t_j(q)$. But when the HDA is also required to be acyclic then this is also ruled out since it would create a cycle. In this paper we usually work with non-degenerate HDAs; and moreover we silently assume all the s/t-maps to be total.

Definition 3.34 (ST to HDA). We define a mapping $H : \Sigma \rightarrow HDA$ from ST-structures into HDAs which for an $\Sigma = (E, ST, l)$ with the events linearly ordered as a list $E$ (i.e., each event being indexed by a natural number, as in a sequence) returns the HDA $H(\Sigma)$ which

- has cells $Q = \{q^{(S, T)} \in Q_n \mid (S, T) \in ST \land |S \setminus T| = n\}$;
- for any two cells $q^{(S, T)}$ and $q^{(S \setminus e, T)}$ add the map entry $s_i(q^{(S, T)}) = q^{(S \setminus e, T)}$ where $i$ is the index of the event $e$ in the listing $E^{\downarrow}_{\Sigma}(S \setminus T)$;
- for any two cells $q^{(S, T)}$ and $q^{(S \cup e, T)}$ add the map entry $t_i(q^{(S, T)}) = q^{(S \cup e, T)}$ where $i$ is the index of the event $e$ in the listing $E^{\uparrow}_{\Sigma}(S \setminus T)$;
- has labelling $l(q^{(T \cup e, T)}) = l(e)$ for any $q^{(T \cup e, T)} \in Q_1$.

More precisely, by $E^{\downarrow}_{\Sigma}(S \setminus T)$ we represent the listing of the events in $S \setminus T$, i.e., a list of dimension $|S \setminus T|$ obtained from the original listing $E$ by removing all other events. This new listing has the events of $S \setminus T$ in the same original order but with new indexes attached (ranging from 1 to $|S \setminus T|$).

Theorem 3.35. For a rooted, connected, and adjacent-closed ST-structure $\Sigma$ the mapping $H$ associates a $H(\Sigma)$ which is a higher dimensional automaton respecting all cubical laws and is acyclic and non-degenerate.

Proof: We first show that $H(\Sigma)$ is a HDA in the sense of Definition 3.23. For any cell $q^{(S, T)}$ all immediately lower cells $q^{(S \setminus e, T)}$ and $q^{(S \cup e, T)}$, with $e \in S \setminus T$ exist because the ST-structure is rooted, connected, and adjacent-closed, and by Proposition 2.15 is closed under single events, therefore all ST-configurations $(S \setminus e, T)$ and $(S \cup e, T)$ exist and thus have the above associated cells. Consider for now that each immediately lower cell $q^{(S \setminus e, T)}$ is linked through $s_e((S, T)) = (S \setminus e, T)$. Note that the s-maps are not indexed as in the definition, but are indexed by an event. We will replace these event indexes by numbers. Link these cells also to $q^{(S \cup T, e)}$ through $t_e(S, T) = (S, T \cup e)$. 

35
To get the cubical laws right we must use a discipline in replacing the event indexes for the s and t maps by numbers. This is what the Definition 3.34 does (inspired by [3]). The listing of the events that the ST-structure comes with provides a bijective indexing map \( i(\cdot) \) from \( E \) to \( \mathbb{N} \). For a specific ST-configuration \((S, T)\) this indexing map becomes a map from \( S \setminus T \) to \( I = \{1, \ldots, n\} \), with \( n = |S \setminus T| \), that respects the original ordering of the events from the listing of \( E \). Call this indexing \( i \downarrow_{(S \setminus T)} \). For the cell \( q^{(S,T)} \) replace \( s_e \) by \( s_{i_e(S,T)}(e) \) (where \( i_e \) is \( i \downarrow_{(S \setminus T)} \)). For each immediately lower cell, like \( q^{(S,e,T)} \), which is linked as \( s_{i_e(S,T)}(e) = (S, T) \), their corresponding indexing maps look like \( i \downarrow_{(S \setminus e)\setminus T} \). The relationship between these two maps \( i \downarrow_{(S \setminus e)\setminus T} \) and \( i \downarrow_{(S \setminus e)\setminus T} \) is easy to see; for simplicity of notation denote the two maps respectively by \( i \) and \( i_e \). The indexing map \( i_e(\cdot) \) is defined on \( (S \setminus e) \setminus T \) as \( i_e(f) = i(f) \) if \( i(f) < i(e) \), and \( i_e(f) = i(f) - 1 \) if \( i(f) > i(e) \). The same holds for \( q^{(S,T,e)} \).

One can check that the cubical laws hold. This is easier done by keeping in mind an intuitive association between each cubical law and the corresponding adjacent-closure constraint. As an example: \( t_{i_{e(S,T)}}(f)(q^{(S,T)}(q^{(S,T)},(S, T)})) = s_{i_{e(S,T)}}(f)(q^{(S,T)}(q^{(S,T)}(e))) \) for two events \( e, f \in S \setminus T \) under the assumption that \( i(f) < i(e) \).

The labelling of the HDA is obtained from the labelling of the ST-structure. Each \( q^{(S,T)} \in Q_1 \) is labelled with \( l(e) \) where \( \{e\} = S \setminus T \) is the single event that is concurrent in \((S, T)\).

**Claim:** The HDA is acyclic and non-degenerate.

To prove non-degeneracy one can notice that for showing that any cell has all its faces distinct it is enough to recall how the faces of some cell \( q^{(S,T)} \) have been built. One s-face is a cell \( q^{(S,e,T)} \) that is obtained from an ST-configuration that can immediately reach \((S, T)\) through an s-step in ST, and which adds the event \( e \); hence the labelling of the corresponding s-map by \( s_e \). Since we added one such map and face for each distinct event from \( S \setminus T \), then all the resulting cells are distinct. The same for the t-maps.

A note is in order. In the definition and the argumentation above, two generated cells \( q^{(S,T)} \) and \( q^{(S',T')} \) are considered equal (respectively different) iff \((S, T) = (S', T')\) (respectively \((S, T) \neq (S', T')\)).

We now finish proving non-degeneracy. For two transitions, i.e., cells of dimension one, hence obtained as \( q^{(S,T)} \) and \( q^{(S',T')} \) with \( S \setminus T = \{e\} \) (respectively \( S' \setminus T' = \{f\} \)) where \( l(e) = l(f) \) assume they have the same source. This implies that \( T = T' \) and thus the two transitions are \( q^{(Te,T)} \) and \( q^{(Tf,T)} \). Since these two transitions are assumed different then it implies that \( e \neq f \). The target of the first transition thus becomes \( q^{(Te,T)} \) and of the second \( q^{(Tf,T)} \) which are different.

To prove that the obtained HDA is acyclic note first that each step in the ST-structure is matched precisely by a corresponding single step of the same type in the HDA. Moreover, one step in the ST-structure increases strictly the dimension of the ST-configuration (since it adds one new event to one of the two sets). Because each cell in the resulting HDA is labelled by an ST-configuration to which it corresponds, we can define a weight for each cell to be the dimension
of the ST-configuration that it is labelled with. With this we can define a weight for each finite path to be the weight of the cell it ends in.

Each path in the $HDA$ is matched by one path in the ST-structure. Since each extension of a path reaches an ST-configuration of strictly larger dimension, it means that each extension of a path in the $HDA$ will have strictly larger weight. To have a cycle, the $HDA$ must have one path that visits the same cell twice; say $q^{(S,T)}$. This means that an initial segment of this path that ends in $q^{(S,T)}$, which has weight $|(S,T)|$, is extended to another path that ends in the same cell, hence having the same weight. But this is a contradiction, since any extension strictly increases the weight. \hfill $\square$

The next lemma ensures that it is immaterial which listing of the events is picked in the definition of the mapping $H$.

**Lemma 3.36.** For some ST and two listings $E_1, E_2$ of the events $E$, the HDAs resulting from the application of $H$ with each listing are isomorphic up to reindexing of the maps.

**Proof:** Take the two generated HDAs to be respectively $H_1$ and $H_2$. Since $E_1^1$ and $E_2^2$ are two listings of the same set then we get a permutation $p$ of their indexes, in the sense that if $e$ is on position $i$ in $E_1^1$ then the same event is on position $p(i)$ in $E_2^2$.

The two generated HDAs are isomorphic through the identity morphism $F(q^{(S,T)}) = q^{(S,T)}$. The only thing to check is that it preserves the mappings up to the reindexing of the maps according to the above permutation; i.e., instead of showing that $s_i(F(q^{(S,T)})) = F(s_i(q^{(S,T)}))$ we show that $s_{p^{-1}(S,T)(i)}(F(q^{(S,T)})) = F(s_i(q^{(S,T)}))$.

In $H_1$ we have that $s_i(q^{(S,T)}) = q^{(S\setminus e,T)}$ with $e$ having index $i$ in the listing $E_1^1 \downarrow_{(S,T)}$, thus making the whole right-hand side of the equality $q^{(S\setminus e,T)}$. On the left side, $F(q^{(S,T)})$ returns the same $q^{(S,T)}$ in $H_2$; and $s_{p^{-1}(S,T)(i)}(q^{(S,T)}) = q^{(S\setminus g,T)}$ where $g$ is the event on index $p \downarrow_{(S,T)}(i)$ in the listing $E_2^2 \downarrow_{(S,T)}$. By the notation $p \downarrow_{(S,T)}(i)$ we mean the restriction of the permutation $p$ to $S \setminus T$ in the following sense. We have $p \downarrow_{(S,T)}(i) = l$ if $E_2^2 \downarrow_{(S,T)}[l] = E_2^2[p(k)]$ with $E_1^1[k] = E_1^1 \downarrow_{(S,T)}[k]$. It is easy to see that $g = e$ and hence the desired result. \hfill $\square$

**Example 3.37 (Strong asymmetric conflict).** This example, taken from [2, Ex.3] (called strong in [12, p.22]), shows the gain in expressive power of the ST-structures. Asymmetric conflict cannot be captured in the pure event structures of [4, Def.1.5], hence not by the configuration structures. Asymmetric conflict can be captured by the unrestricted event structures of [2], and thus, also by the adjacent-closed ST-structure of Fig. 19 (middle).
The example has no concurrency and involves two events, imposing the only restriction that once event $s$ happens, event $b$ cannot happen any more. The notion of asymmetric conflict for even structures has been studied in [36].

Within HDAs it is more cumbersome to represent this example because HDAs are not good at identifying the particular events. HDAs abstract from the concrete events and concentrate only on the labels. One way of identifying events is by equivalence classes of transitions, where two transitions are equivalent when they are parallel in the boundary of a filled square (i.e., what we assumed until now in our HDAs examples).

Applying this technique to the HDA in Fig. 5(middle) would not result in the corresponding 2-events ST-structure (which is what we want), but would result in the 3-events ST-structure of Fig. 5(right), and these two ST-structures are not isomorphic. On the other hand, the two representations of HDA from Fig. 5(middle and right) are isomorphic. Nevertheless, if we are interested in representing systems only up to $hh$-bisimulation, then both HDAs and ST-structures are as good, because the different representations of ST-structures (with 2 or 3 events) would be equated by the $hh$-bisimulation.

**Proposition 3.38.**

1. The mapping $H$ from Definition 3.34 preserves isomorphism; i.e.,

   for $ST \cong ST'$ then $H(ST) \cong H(ST')$.

2. The mapping $H$ may collapse non-isomorphic ST-structures into isomorphic HDAs.

**Proof:** For the second part of the proposition consider the two ST-structures from Figure 5 which are not isomorphic as the left one is defined on two events whereas the right one is on three events. But the HDAs that the mapping $H$ associates are isomorphic.

For the first part of the proposition consider two isomorphic ST-structures $ST \cong ST'$ with $f : E \to E'$ their respective isomorphism. To show that $H(ST) \cong H(ST')$ we build an isomorphism between the two generated HDAs as $F : Q \to Q$. 

---

Figure 5: Strong asymmetric conflict $s + b; s$ as unrestricted event structure (left), and ST-structure (middle). Related isomorphic HDAs give rise to non-isomorphic ST-structures (middle and right).
Definition 3.39 (HDA to ST).

Consider a non-degenerate HDA $H = (Q, π, T, I, I)$. Define a relation $\sim \subseteq Q_1 \times Q_1$ on transitions as

$$q_1 \sim q'_1 \ \text{iff} \ \exists q_2 \in Q_2 : \alpha_i(q_2) = q_1 \land \beta_i(q_2) = q'_1$$

for some $i \leq 2$ and $\alpha, \beta \in \{s, t\}$. Consider the reflexive and transitive closure of the above relation, and denote it the same. This is now an equivalence relation on $Q_1$. Consider an equivalence class $[q_1]$ to be all $q'_1$ equivalent with $q_1$. Such an equivalence class is called an event.

Define a map $\text{ST} : \text{HDA} \to \text{ST}$ which builds an ST-structure $\text{ST}(H)$ by associating to each rooted path $\pi \in H$ an ST-configuration as follows.

1. for the minimal rooted path which ends in $I$ associate $(\emptyset, \emptyset)$;
2. for any path $\pi$ which ends in a transition $en(\pi) = q_1 \in Q_1$ then

$Q'$ given by $F(q^{(S,T)}) = q^{(f(S), f(T))}$. We prove that $F$ is a dimension preserving isomorphism of the two $H(\text{ST})$ and $H(\text{ST}')$ as in Definition 3.38.

Because of Lemma 3.38 if for the translation of $\text{ST}$ we pick some listing of $E$, then for the translation of $\text{ST}'$ we pick the listing of $f(E)$ such that the order of the events is preserved; i.e., if $e_i < e_j$ in the listing of $E$ then also $f(e_i) < f(e_j)$ in the listing of $f(E)$. Since $f$ is a bijection this means that if $e_i$ is on position $i$ in the listing of $E$ then we find $f(e_i)$ on the same position $i$ in the listing of $f(E)$.

For $q^{(S,T)}$ which was generated from the $(S, T) \in \text{ST}$, the isomorphism of $\text{ST}$-structures ensures that $(f(S), f(T)) \in \text{ST}'$, which means that the mapping will associate the cell $q^{(f(S), f(T))} \in H(\text{ST}')$. This makes $F$ well defined. Moreover, because the isomorphism $f$ preserves the concurrency degree of the $\text{ST}$-configurations, i.e., $|S \setminus T| = |f(S) \setminus f(T)|$, then $F$ preserves the dimension of the cells. It is easy to see that $F$ preserves the initial cell. $F$ also preserves the labelling since $l'(F(q^{(T, T')})) = l'(f(q^{(T, T')})) \ (1)$ comes from the Definition 3.38 of the $H$ map and (2) comes from the Definition 2.14 of isomorphism for $\text{ST}$-structures; and since $l(q^{(T, T')}) \ (1) \equiv I(e)$, we obtain the requirement (2) of Definition 3.24.

It remains to show that $F$ preserves the mappings, i.e., $s_i(F(q^{(S,T)})) = F(s_i(q^{(S,T)}))$. (The case for $t$-maps is analogous.) By Definition 3.34 we have $s_i(q^{(S,T)}) = q^{(S, e, T)}$ for $e$ with index $i$ in the listing of the events $E \downarrow S \setminus T$, and thus $F(s_i(q^{(S,T)})) = q^{(f(S), e, f(T))}$. Since $F(q^{(S,T)}) = q^{(f(S), f(T))}$, by Definition 3.34 $s_i(q^{(f(S), f(T))}) = q^{(f(S), \alpha, \beta, f(T))}$ for $g$ the event with index $i$ in the listing of the events $f(E) \downarrow f(S) \setminus f(T)$. If we choose the listing of the events of $\text{ST}'$ such that the isomorphisms $f$ preserves their order, as we explained before, then the event $g$ is exactly $f(e)$. Therefore, we have the equality we are looking for; $s_i(F(q^{(S,T)})) = q^{(f(S), f(e), f(T))}$.

A corollary of Proposition 3.38 is that $H$ is not an embedding from $\text{ST}$ to $\text{HDA}$ since it loses information, i.e., the events.
(a) add the ST-configuration $\text{ST}(\pi) = \text{ST}(\pi_s) \cup ([q_1], \emptyset)$, where $\pi_s$ is a shorter path reaching through an $s$-map the homotopy class of $\pi$, i.e., $\pi_s \rightarrow q_1 \in [\pi]$;

(b) add the ST-configuration $\text{ST}(\pi \rightarrow q_0) = \text{ST}(\pi) \cup (\emptyset, [q_1])$;

3. for any path $\pi$ which ends in a higher cell $\text{en}(\pi) = q_n \in Q_n$, with $n \geq 2$, then add the ST-configuration $\text{ST}(\pi) = \text{ST}(\pi^i) \cup \text{ST}(\pi^j)$, with $\pi^i \neq \pi^j$, $\pi^i \rightarrow q_n \in [\pi]$, and $\pi^j \rightarrow q_n \in [\pi]$.

Note that in the case (3) above the paths $\pi^i, \pi^j$ always exist because we work with non-degenerate HDAs. The same goes for the path $\pi_s$ used in (2a).

**Proposition 3.40.** For an acyclic and non-degenerate HDA, the resulting ST-structure $\text{ST}(H)$ is rooted, connected, and adjacent-closed.

**Proof:** Rootedness is easy because it corresponds to the minimal rooted path of the HDA, i.e., the initial cell. Note that Definition 3.39 works with rooted paths which implies that only the reachable part of the HDA is considered, which is connected.

In a non-degenerate HDA for any cell $q_n \in Q_n$, with $n \leq 1$, all the faces $s_i(q_n)$ exist and are distinct; and the same for $t$-maps. Therefore, in a non-degenerate HDA every $Q_1$ cell has exactly one $s$ and one $t$ map. Hence, for any $\pi$, with $\text{en}(\pi) = q_1$, there is exactly one continuation by a $t$-step, and this reaches a $q_0$ state cell; thus motivating in part 2a of the definition the consideration of the paths $\pi \rightarrow q_0$. In fact any state cell $q_0 \in Q_0$ can be reached through such a path.

The same path $\pi$ can be reached either through an $s$-step from a $Q_0$ cell or through a $t$-step from a $Q_2$ cell. In the second case homotopy ensures that there also exists a path homotopic with $\pi$ that reaches $\text{en}(\pi)$ through an $s$-step. This ensures that step 2a in the construction of ST is well defined.

Connectedness is proven using induction on the length of the paths that are used to generate the ST-configurations. The induction hypothesis says that ST-configurations generated from shorter paths are already reachable from the root of the ST-structure we are building.

It is not difficult to see that the ST-configuration introduced in 2a is connected to the ST-configuration of the immediately shorter path through adding one new event to the $S$ set. This event is new because the HDA is acyclic, and thus the path $\pi$ never goes through a cell twice; in particular it has never been through the $q_1$ that is used in the definition. Even more, the path cannot go through any other cell equivalent with $q_1$ because it would break the directed nature of the paths. Similarly, the ST-configuration introduced in 2b is connected to the ST-configuration of the immediately shorter path through a t-step, i.e., terminating the event $[q_1]$.

All remaining ST-configurations that need to be considered are built in step 3 from paths ending in cells of dimension higher than 1. To show connectedness for these paths we show that the associated ST-configuration differs from each
immediately lower paths reaching it through an s-step by only one new event in the $S$ set. The basis for this has already been proven for paths ending in cells from $Q_1$.

For some path $\pi$ and two $\pi^i$ and $\pi^j$ with $s_i(en(\pi))$ and $s_j(en(\pi))$, a cubical law is applicable. Assume wlog. $i < j$ and consider $s_i(s_j(en(\pi))) = s_{j-1}(s_i(en(\pi))) = q''$. Take that path $\pi''$ with $en(\pi'') = q''$ which when extended as above is still part of the homotopy class of $\pi$, i.e., $\pi'' \xrightarrow{s_j} \pi \xrightarrow{s_i} \pi \in \hat{\pi}$. From the induction hypothesis, $ST(\pi')$ differs from $ST(\pi'')$ in one new event, i.e., $ST(\pi^i) = ST(\pi'') \cup ([q_i], \emptyset)$; whereas $ST(\pi^j)$ differs from $ST(\pi'')$ in some other event $[q_j]$. Assuming these two events are different we have our desired result because $ST(\pi) = ST(\pi'') \cup ([q_i], \emptyset) \cup ([q_j], \emptyset)$ differs from each immediately lower $ST(\pi^i)$ or $ST(\pi^j)$ through one event.

Any path $\pi$, when extended $\pi \xrightarrow{s_i} \pi^i$ and $\pi \xrightarrow{s_j} \pi^j$ by two s-maps that enter a cubical law, involves two different events, i.e., the two new end cells $en(\pi^i)$ and $en(\pi^j)$ are not coming from parallel sides of the cube to which the cubical law pertains. This implies through a recursive reasoning that the newly added events are different. The reasoning ends in cells of dimension 1, when $\pi$ ends in a state, and the cube is of dimension 2.

To show adjacent-closure we use Proposition 2.15 because the ST-structure is rooted and connected. Therefore it is enough to show closure under single events.

For ST-configurations introduced by the phase 2 the closure is clear from the definition. We thus look at ST-configurations introduced in phase 3 as $(S, T) = ST(\pi) = ST(\pi^i) \cup ST(\pi^j)$ with $en(\pi) = q_n \in Q_n$ and $n > 1$. From the non-degeneracy of $H$ we know that $\pi$ can be extended with all $n$ t-maps. We take $t_i$ which extends $\pi \xrightarrow{t_i} \pi'$ and show that $ST(\pi') = ST(\pi) \cup (\emptyset, [q_i])$ where $[q_i]$ is the one event by which $ST(\pi)$ differs from the shorter $ST(\pi')$.

We know from before that $ST(\pi) = ST(\pi_{n-2}) \cup ([q_i], \emptyset) \cup ([q_j], \emptyset)$ with $\pi_{n-2}$ being the shorter and two dimensions lower path entering the following cubical law for $\pi$: $s_i(s_j(en(\pi))) = en(\pi_{n-2}) = s_{j-1}(s_i(en(\pi)))$. Besides this we have another cubical law: $t_i(s_j(en(\pi))) = en(\pi_{n-2}^{t_i}) = s_{j-1}(t_i(en(\pi)))$. Put together these two cubical laws say that the end cells of $\pi^i$ and $\pi \xrightarrow{t_i} \pi^{t_i}$ are parallel faces of the cube $en(\pi)$. A recursive reasoning thus tells that the event added when extending $\pi_{n-2}$ with $s_{j-1}$ is the same as the event added when extending $\pi_{n-2}^{t_i}$ with $s_{j-1}$, which we denoted $[q_i]$.

Since $\pi^j$ is shorter, the induction hypothesis says that is is closed under single events and thus when extended with $\pi^{t_i} \xrightarrow{t_i} \pi^{t_i}_{n-2}$ we reach the ST-configuration $ST(\pi_{n-2}) \cup ([q_j], \emptyset) \cup (\emptyset, [q_j])$. But we know that $ST(\pi^{t_i}) = ST(\pi_{n-2}^{t_i}) \cup ([q_i], \emptyset)$ which is the same as $ST(\pi^j) = ST(\pi_{n-2}^j) \cup ([q_j], \emptyset) \cup (\emptyset, [q_j]) \cup ([q_i], \emptyset)$. Therefore we have found for $ST(\pi)$ this ST-configuration $ST(\pi^{t_i})$ which is reachable through the event $[q_i]$. Since every t-map that extends $\pi$ adds one different event, we have our closure under single events. \qed
Proposition 3.41.

1. The mapping $\text{ST}$ from Definition 3.39 preserves isomorphism of reachable parts; i.e.,
   
   \[ H \cong H' \text{ then } \text{ST}(H) \cong \text{ST}(H'). \]

2. The mapping $\text{ST}$ may collapse non-isomorphic HDAs into isomorphic ST-structures.

Proof: For the second part of the proposition consider the two HDAs from Figure 6 without the two dotted transitions, which are translated into the same ST-structure by ST. Even when the dotted transitions are added, they are mapped to the same ST-structure.

For the first part we build an isomorphism $I_{\text{ST}}$ between $\text{ST}(H)$ and $\text{ST}(H')$ starting from the isomorphism $I_H$ between $H \cong H'$ as follows. Take

\[ [q]I_{\text{ST}}[q'] \iff \exists q \in [q], q' \in [q'] : qI_Hq'. \]

Isomorphisms of HDAs preserve equivalence classes (i.e., if $qI_Hq'$ then $\forall q_1 \in [q], q_1' \in [q'] : q_1I_Hq_1'$ ) and preserve the structure of paths in the sense that for any path $\pi$ the isomorphic path $\pi'$ has the same s/t maps. The isomorphism $I_H$ applied to paths is defined as its application to all the cells on the path.

We want to show that $I_{\text{ST}}$ preserves ST-configurations in the sense that any $(S, T) \in \text{ST}(H)$ then $I_{\text{ST}}(S, T) \in \text{ST}(H')$. From the construction of $\text{ST}(H)$ we know that it exists a path $\pi$ s.t. $\text{ST}(\pi) = (S, T)$. This has an isomorphic path $I_H(\pi) = \pi' \in H'$. Since $\pi'$ has the same s/t sequence the ST is applied the same on $\pi'$ as on $\pi$ and thus adds the events corresponding to $I_H(q')$, with $q'$ the corresponding events from $\pi$. Therefore it adds $[I_H(q')]$ and the resulting ST-configuration $\text{ST}(\pi')$ looks like $(S, T)$ but for any $[q']$ we have instead $[I_H(q')]$. By the definition of $I_{\text{ST}}$ this implies that for each $[q']$ we have the corresponding $I_{\text{ST}}([I_H(q')])$ in $\text{ST}(\pi')$ which gives our desired result that $I_{\text{ST}}(\text{ST}(\pi)) \in \text{ST}(H')$.

Remark 3.42 (no adjoint). One can prove that the two maps $H$ and $\text{ST}$ can be lifted to functors between the categories $\text{ST}$ and $\text{HDA}$. Nevertheless, we cannot...
find a unit to make $H$ the left adjoint of $ST$ because of the example of the $ST$-structure of Figure 5(middle). For this $ST$-structure there is no way to associate a morphism to its translation through $ST \circ H$, which is the $ST$-structure from Figure 5(right). There is also not possible to get the adjunction the other way, because of the example of Figure 7(right) showing unfolding of the triangle $HDA$. For this $HDA$ there is no way to associate a morphism to its translation through $H \circ ST$, which is unfolded.

3.4. Correspondence with Chu spaces

We show the correspondence between $ST$-structures and the Chu spaces over $3$ of Pratt [2]. The latter can be represented in terms of the 3-2 logic over $E$, the set of events. Instead of two values for each event, i.e., $0$ for not started and $1$ for finished (or before and after), the 3-valued case introduces the value $\frac{1}{2}$ to stand for during, or in transition. These values are ordered as $0 < \frac{1}{2} < 1$, which extends to the whole $3^E$. $ST$-structures capture the “during” aspect in the event-based setting, extending the configuration structures with this notion.

Note that configuration structures [8, 9] correspond to Chu spaces over $2$. A Chu space over $K$ is a triple $\text{Chu} = (A, r, X)$ with $A$ and $X$ sets and $r : A \times X \to K$ is an arbitrary function called the matrix of the Chu space, and $K$ is in our examples a set with a partial order on it. Chu spaces can be viewed in various equivalent ways. For our setting we can take the view of $A$ as the set of events and the $X$ as the set of configurations. The set $K$ is representing the possible values the events may take: when $K = \{0, 1\}$ is the classical case of an event being either not started or terminated, where an order of $0 < 1$ would be used to define the steps in the system. In general, the order on $K$ will be used to define the meaningful steps in the Chu space.

The Chu space can be viewed as a matrix with entries from $K$ and rows representing the events of $A$ and columns representing the configurations of $X$. As an example, an entry $r((e, x)) = 0$ says that the event $e$ is not started yet in the configuration $x \in X$. In consequence, a Chu space can also be viewed as the structure $(A, X)$ where $X \subseteq K^A$. This very much resembles the configuration structures when $K$ is $2$. 

Figure 7: Unfoldings of $HDA$s through $ST$. 
Proposition 3.43 (ST-structures and Chu spaces over 3). ST-structures and Chu spaces over 3 (cf. [2, Sec.3]) are isomorphic.

Proof: It is not difficult to see how for events in ST-structures we can associate a value as required by the Chu spaces. Moreover, this association is bijective.

We provide an association between Chu spaces over 3 and ST-structures. For an ST-structure \( ST \) construct \( (E, X)^{ST} \) the associated Chu space over 3 with \( E \) the set of events from \( ST \) and \( X \subseteq 3^E \) states of the system formed of valuations of the events into the set \( 3 = \{0, \frac{1}{2}, 1\} \) as follows. For one ST-configuration \( (S, T) \) build a state \( x^{(S,T)} \in X \) by assigning to each \( e \in E \):

- \( e \rightarrow 0 \) iff \( e \notin S \land e \notin T \);
- \( e \rightarrow \frac{1}{2} \) iff \( e \in S \land e \notin T \);
- \( e \rightarrow 1 \) iff \( e \in S \land e \in T \).

The possibility \( e \notin S \land e \in T \) is dismissed by the requirement \( T \subseteq S \) of ST-structures. This association can be applied the other way to obtain the ST-structure corresponding to the Chu space. \( \square \)

4. Action refinement for ST-structures

We define the notion of action refinement [11] for ST-structures using a refinement function \( \text{ref} : \Sigma \rightarrow ST \). The \( \text{ref}(a) \) is a non-empty ST-structure which is to replace each and all events that are labelled with \( a \). In this way, what before was abstracted away into a single event, now using action refinement, can be given more (concurrent) structure. The definition of \( \text{ref} \) is over all action labels of \( \Sigma \), but normally only a (small) subset of actions is refined, whereas the rest should remain the same. For all such actions just refine with a singleton ST-structure labelled the same (i.e., \( \text{ref}(b) = (\{()\}, \{()\}, (f, f), l(f) = b) \) with \( f \) a new event).

Definition 4.1 (action refinement). Consider \( ST = (ST, l) \) over \( \Sigma \) an ST-structure to be refined by a refinement function \( \text{ref} : \Sigma \rightarrow ST \). We call the pair of sets \( (S, T) \) a refinement of an ST-configuration \( (S_{ST}, T_{ST}) \in ST \) by \( \text{ref} \) iff:

\[
(\tilde{S}, \tilde{T}) = \bigcup_{e \in S \setminus T} \{(e) \times S_e, (e) \times T_e\} \cup \bigcup_{f \in T} \{(f) \times S_f, (f) \times S_f\}
\]

where each \( (S_e, T_e) \) is a non-empty and non-maximal ST-configuration from \( \text{ref}(l(e)) \) and each \( (S_f, T_f) \) is a maximal ST-configuration from \( \text{ref}(l(f)) \). The refinement of \( ST \) is defined as \( \text{ref}(ST) = (ST_{ref}, l_{ref}) \) with

- \( ST_{ref} = \{(\tilde{S}, \tilde{T}) \mid (S, T) \text{ is a refinement of some } (S, T) \in ST \text{ by } \text{ref}\} \);
- \( l_{ref}((e, e')) = l_{\text{ref}(l(e))}(e') \).

44
Note that because of the closure restriction in the definition of ST-structures, any maximal ST-configuration, wrt. set inclusion, must have both $S$ and $T$ equal.

**Proposition 4.2 (refinement is well defined).**

For two isomorphic ST-structures $ST \cong ST'$ and two isomorphic refinement functions, i.e. $\forall a \in \Sigma : ref(a) \cong ref'(a)$, we have:

1. $ref(ST)$ is also an ST-structure;
2. $ref(ST) \cong ref'(ST)$; and
3. $ref(ST) \cong ref(ST')$.

**Proof:** We first have to show that every set $(\tilde{S}, \tilde{T}) \in ST_{ref}$ is a well defined ST-configuration. This is easy to see because we use unions in the definition of $(\tilde{S}, \tilde{T})$ and in the right-most unions the resulting $S$ and $T$ sets are the same, whereas in the left unions all events in the $T$ sets are also in the $S$ sets because they are built from ST-configurations $(S_e, T_e)$, i.e., $\{e\} \times T_e \subseteq \{e\} \times S_e$. Union of sets preserves set inclusion.

We need to show now that for any set $(\tilde{S}, \tilde{T}) \in ST_{ref}$ there is the case that $(\tilde{S}, \tilde{S}) \in ST_{ref}$. Assume that $(\tilde{S}, \tilde{T})$ is a refinement of some $(S, T)$ and assume there are some events $e \in S \setminus T$ and that we have used some ST-configuration $(S_e, T_e)$ from $ref(l(e))$. (Otherwise, when $S = T$ it implies, by construction, that also $\tilde{S} = \tilde{T}$.) Because the refinement function uses only ST-structures then it means that in $ref(l(e))$ there is also an ST-configuration $(S_e, S_e)$. If $(S_e, S_e)$ is maximal, then the refinement of $(S, T)$ that uses it will result in the desired $(\tilde{S}, \tilde{S})$. Otherwise, when $(S_e, S_e)$ is not maximal, by the Definition 4.1 it means we eventually build another refinement of $(S, T)$ using this $(S_e, S_e)$. Therefore, we would have both the $\tilde{S}$ and $\tilde{T}$ sets the same. These arguments must be carried for all events $e \in S \setminus T$ to obtain the desired $(\tilde{S}, \tilde{S})$.

The final step is to show that the new labelling function $l_{ref}$ is correctly build in the sense that for each new event it assigns some label, in fact the correct label coming from the refining ST-structures given by $ref$. The definition of $l_{ref}$ is for every new event $(e, e')$ where $e$ is from the old ST-structure and $e'$ is from the new refining structures. The new label is the same as the corresponding label in the ST-structure where the $e'$ comes from.

The refinement operation is well defined also wrt. isomorphic structures.

- For a ST and two isomorphic refinement functions $ref$ and $ref'$, i.e. $\forall a \in \Sigma : ref(a) \cong ref'(a)$, then
  
  $ref(ST) \cong ref'(ST)$.

- For two isomorphic ST-structures $ST \cong ST'$ and some refinement function $ref$ we have that
  
  $ref(ST) \cong ref(ST')$.

The proof uses the fact that isomorphic ST-structures agree on the labelling functions, and uses similar arguments as above. $\square$
Proposition 4.3 (preserving properties).

The refinement operation preserves the properties of the refined structure, i.e., for some \( ST \) and \( ref \):

- If \( ST \) is rooted then \( ref(ST) \) is rooted.
- If \( ST \) is connected and all \( ref(a) \) are connected then \( ref(ST) \) is connected.
- If \( ST \) and all \( ref(a) \) are adjacent-closed then \( ref(ST) \) is adjacent-closed.
- If \( ST \) and all \( ref(a) \) are closed under bounded unions (or intersections) then \( ref(ST) \) is closed under bounded unions (resp. intersections).

Proof: The property of being rooted is easy.

For connectedness consider some non-empty \((S, \tilde{T}) \in ref(ST)\) which is a refinement of some (non-empty) \((S, T) \in ST\). The fact that is non-empty it implies that \( \exists (e, e') \in \tilde{S} \) an event, which by the definition of refinement it means that \( e \) comes from \( S \). Therefore \((S, T) \) is also non-empty. By the connectedness of the original \( ST \) it means that there exists some \( f \in S \) s.t. either (1) \((S \setminus f, T) \in ST\) or (2) \((S, T \setminus f) \in ST\).

Assume that (1) \((S \setminus f, T) \in ST\) and take a refinement of it where for all events different than \( f \) we take the same ST-configurations \((S, T) \in ref(l(e))\) as we did in the refinement for \((S, T)\). This means that \((S, \tilde{T})\) has all the events of \((S \setminus f, \tilde{T})\) and on top it has some more events coming from the refinement of \( f \). If there is only one such event \((f', \emptyset) \in ref(l(f))\), then the proof is finished since we found this event which if removed from \( S \) we obtain a new ST-configuration which is also in \( ref(ST) \). Therefore, consider the case when there are more events \((f, f')\) as coming from some chosen ST-configuration \((S, T) \in ref(l(f))\) is also connected it means that we can find some event \( f_k \) s.t. either \((S \setminus f_k, T) \in ref(l(f))\) or \((S, T \setminus f_k) \in ref(l(f))\). No matter which is the case we have the following: one refinement of \((S, T)\) is \((S, \tilde{T}) \in ref(ST)\) using \((S, T) \in ref(l(f))\) and another refinement of the same \((S, T)\) uses all the same ST-configurations except for \((S, T)\), in place of which another ST-configuration is used which has exactly one less event. This concludes this case, as we found the single event \((f, f')\) which can be removed to obtain another refinement.

Assume (2) \((S, T \setminus f) \in ST\) and take the refinement of \((S, T \setminus f)\) the same as that for \((S, T)\). This is possible because before, in \((S, T)\), \( f \) was part of \( T \) and thus it was refined using some maximal configuration \((S_f, S) \in ref(l(f))\). But since now \( f \in S \setminus (T \setminus f) \) we can refine with any configuration from \( ref(l(f))\), and hence also with the maximal one \((S_f, S) \in ref(l(f))\). Because \( ref(l(f)) \) is connected then there exists some event \( f_k \) in the maximal configuration \((S_f, S_f) \) s.t. \((S_f, S_f \setminus f_k) \in ref(l(f))\). Use this configuration to refine \( f \) instead of the one \((S_f, S_f) \) used to obtain \((S, \tilde{T})\). In this way we obtain a refinement configuration in \( ref(ST) \) that differs from \((S, \tilde{T})\) by one single event \((f, f')\).
For bounded unions and intersections the proof should be similar to that in [11, Prop.5.6] and should use argument specific to ST-structures as we used above.

We concentrate on the new property of adjacent closure. One can prove this directly using Definition 2.13 of adjacent-closure and take (more tedious) cases. We will take the alternative route through Proposition 2.15 and use the closure under single events. This results in fewer cases to consider.

Therefore, we want to show that for some arbitrary \((S, T) \in \text{ref(ST)}\), being a refinement of some \((\tilde{S}, \tilde{T}) \in \text{ST}\) we have

1. \(\forall (e, e') \in \tilde{S} \setminus \tilde{T} : (\tilde{S}, \tilde{T} \cup (e, e')) \in \text{ref(ST)}\) and
2. \(\forall (e, e') \in \tilde{S} \setminus \tilde{T} : (\tilde{S} \setminus (e, e'), \tilde{T}) \in \text{ref(ST)}\).

Since \((e, e') \in \tilde{S} \setminus \tilde{T}\) it means that \(e \in S \setminus T\) and \(e' \in (S_e, T_e)\), where \((S_e, T_e) \in \text{ref(l(e))}\) is the chosen ST-configuration. Because \(\text{ref(l(e))}\) is closed under single events, the first requirement then says that also \((S_e, T_e \cup e') \in \text{ref(l(e))}\). Take now another refinement of \((S, T)\) that is the same as before only that in place of \((S_e, T_e) \in \text{ref(l(e))}\) uses \((S_e, T_e \cup e')\). Clearly this new refinement has all the events of the old refinement with the exception that now the event \((e, e')\) is also contained in \(\tilde{T}\). This proves the first requirement.

To show the second requirement consider the existence of \((S_e \setminus e', T_e) \in \text{ref(l(e))}\) and the same argument applies as before. The only difference is that the new \((S_e \setminus e', T_e)\) may in fact be \((\emptyset, \emptyset)\). In this case we use the fact that the original ST is closed under single events and thus, for the \(e \in S \setminus T\) we can find \((S \setminus e, T) \in \text{ST}\). Take the refinement of this which will have all the event of the old one, except the one \((e, e')\) which is not in \(\tilde{S}\) anymore. \(\square\)

Single steps in the new structure relate to steps in the old structure, before the refinement, and the refining structures given by \(\text{ref}\). A single s-step in the refined structure comes either from a single s-step in the old structure, and thus coupled with an initial step in \(\text{ref}\) (i.e., one like \((\emptyset, \emptyset) \xrightarrow{s} (e, \emptyset))\) or only from an s-step in the \(\text{ref}\) (with the ST-configuration unchanged).

**Proposition 4.4.** The hh-bisimulation is preserved under action refinement; i.e.,

\[
\text{for ST} \overset{hh}{\sim} \text{ST'} \text{ then } \text{ref(ST)} \overset{hh}{\sim} \text{ref(ST')},
\]

**Proof:** We consider given a hereditary history preserving bisimulation (hh-bisimulation) \(R\) that relates the two initial ST-configurations of \(\text{ST}\) and \(\text{ST'}\). We construct a relation \(\tilde{R}\) between the refinements \(\text{ref(ST)}\) and \(\text{ref(ST')}\) which will also equate their initial empty configurations, and we show that it respects the restrictions of Definition 2.20 of being a hh-bisimulation.

We will also show that the proof works also when we consider \(R\) to be only history preserving bisimulation. Moreover, we point out how the proof can be changed to show a result where we do not refine with the same refinement function, but with refinement functions that are also hh-bisimilar (or only h-bisimilar in the other case).
Define $\tilde{R}$ as:

$((\tilde{S}, \tilde{T}), (S', T')) \in \tilde{R}$ iff $\exists ((S, T), (S', T'), f) \in R$ s.t.

1. $(\tilde{S}, \tilde{T})$ is a refinement of $(S, T)$;
2. $(\tilde{S}', \tilde{T}')$ is a refinement of $(S', T')$ that uses the same choices as done for $(\tilde{S}, \tilde{T})$;
3. $f : \tilde{S} \rightarrow \tilde{S}'$ is defined as $\tilde{f}(e, e') = (f(e), e')$.

When we want to prove the result for hh-bisimilar refinement functions $\text{ref} \overset{hh}{\sim} \text{ref}'$ then we need to complicate the definition of $\tilde{R}$ by adding one more requirement:

4. $\forall e \in S$, with $(S_e, T_e) \in \text{ref}(l(e))$ the refining configuration that makes $(\tilde{S}, \tilde{T})$, $\exists f_e : S_e \rightarrow S'_{f(e)}$ s.t. $((S_e, T_e), (S'_{f(e)}, T'_{f(e)}), f_e) \in R_{l(e)}$, with $R_{l(e)}$ being the hh-bisimulation relating the refinement choices for the label $l(e)$ for $(\tilde{S}', \tilde{T}')$;

and we also need to change the requirement 3 above to satisfy $\tilde{f}(e, e') = (f(e), f_e(e'))$.

It is not difficult to see from the definition above that because $(\emptyset, \emptyset, \emptyset) \in R$ then also $(\emptyset, \emptyset, \emptyset) \in \tilde{R}$. It remains to prove the restrictions of Definition 2.20.

1. We prove that $\tilde{f}$ is an isomorphism of $(\tilde{S}, \tilde{T})$ and $(\tilde{S}', \tilde{T}')$. Since $f$ is an isomorphism between $(S, T)$ and $(S', T')$ then, by Definition 2.19, $f$ is an isomorphism of $S$ and $S'$ that agrees on the $T$ and that preserves the labelling. In consequence, when the refinement $(\tilde{S}, \tilde{T})$ involves some event $e \in S$ then the isomorphic image $f(e)$ has the same label, hence it is refined with the same ST-structure $\text{ref}(l(f(e))) = \text{ref}(l(e))$. Here is where the constraint 2. in the definition of $\tilde{R}$ says to make the same choice of $(S_e, T_e)$. By the definition of $f$, this is also an isomorphism of $\tilde{S}$ and $\tilde{S}'$ since we use the same events $e'$. It is easy to see that $\tilde{f}$ agrees on $\tilde{T}$. The $\tilde{f}$ also preserves the labelling function of the new refinements because the new events get the label of the second component $e'$ which is related to the label of either $e$ or $f(e)$, which are the same.

When proving the proposition for two bisimilar refining functions then the argument above works because of the extra requirement 4. This gives an equivalent configuration to pick when obtaining $(\tilde{S}', \tilde{T}')$, i.e., pick $(S'_{f(e)}, T'_{f(e)})$. Then in the definition of $\tilde{f}$ we use not the same $e'$ but an isomorphism $f_e$, therefore the $\tilde{f}$ is also an isomorphism.

2. We prove the second requirement of Definition 2.20 and assume there is a step $(S, T) \overset{a}{\rightarrow} (S_a, T_a)$ in the refinement $\text{ref}(ST)$, for $((\tilde{S}, \tilde{T}), (S', T'), \tilde{f}) \in \tilde{R}$. This is equivalent to saying that we have $((S, T), (S', T'), f) \in R$ satisfying the three requirements from before, i.e., that $(\tilde{S}, \tilde{T})$ is a refinement of $(S, T)$, $(\tilde{S}', \tilde{T}')$ is a refinement of $(S', T')$ with the same choices, and that $\tilde{f}(e, e') = (f(e), e')$. We take cases depending on what kind of step and how did this step get formed.

48
(a) When we work with an s-step which is formed from an s-step in the refinement and the same (S, T) in the original ST. The s-step comes from a configuration (S, T) ↪ (S ∪ g, T) corresponding to refining some e ∈ S. More precisely, if (S, T) = ((e) × S, (e) × T) ∪ \( \bigcup_{e′ \neq S \cap T} (\{e′\} × S, (e′) × T) \cup \bigcup_{f \in T} (\{f\} × S, f × S) \) and knowing that the above s-step is in \( \text{ref}(l(e)) \) and therefore also in \( \text{ref}(l(f(e))) \), since the isomorphism f preserves labelling, then the s-step we are assuming, i.e., (S, T) ↪ (S, T) is obtained by having \((S, T)\) also a refinement of \((S, T)\) with the same choices as for \((S, T)\) with one difference: \((S, T) = ((e) × (S ∪ g), (e) × T) ∪ \bigcup_{e′ \neq S \cap T} (\{e′\} × (S ∪ g), (e′) × T) \cup \bigcup_{f \in T} (\{f\} × (S ∪ g), f × (S ∪ g)) \). Which means that one new event is added to \(\tilde{S}\), and that is \((e, g)\). Knowing that \((S′, T′)\) is a refinement of \((S, T)\) which is in relation R with \((S, T)\) and f, then we take another refinement of \((S', T')\), the same as \((S', T')\) in all respects except that for the event \(f(e)\) we take the configuration \((S_e ∪ g, T_e)\) (which we know from before that it exists, because the above step exists in \( \text{ref}(l(f(e))) \)). Denote this new refinement as \((S'_e, T'_e)\) which has the difference in the ST-configuration \((f(e) × (S_e ∪ g), f(e) × T_e)\), i.e., the single step that we are looking for adds the new event \((f(e), g)\). Clearly there is a single s-step \((S'_e, T'_e) \rightarrow (S'_a, T'_a)\). Moreover, the new configurations are in the relation we built \(((S_a, T_a), (S'_a, T'_a), f_a) \in R \) where \(f_a\) extends \(\tilde{f}\) with \(f_a(e, g) = (f(e), e, g)\). It is easy to show this last statement, using the definition for \(R\) above. Just take the same \(((S, T), (S'_e, T'_e), f) \in R\), and thus have that \((S_a, T_a)\) is a refinement of \((S, T)\) by construction, and \((S'_a, T'_a)\) a refinement of \((S', T')\) using the same choices (in particular choosing \((S_e ∪ g, T_e)\) for refining \(e\)). It is easy to see that \(f_a\) respects the condition 3 since is extends \(\tilde{f}\) which does.

We are interested how the proof changes when working with two refinement functions. Because the refining configurations are hbi-similar it means that instead of the same configuration as before, we find a bisimilar one which comes as an s-step extension of the old one; i.e., we find \((S_e ∪ f_e(g), T_e)\). In consequence the isomorphism is extended with \(f_a(e, g) = (f(e), f_e(g))\).

The rest of the three cases are similar and we skip their details.

(b) When we work with an s-step which is formed from an s-step in the original ST, and a minimal s-step in the refinement. More precisely, this step comes from the step \((S, T) \rightarrow (S ∪ e, T)\) and some initial step \((\emptyset, \emptyset) \rightarrow (g, \emptyset)\) in \( \text{ref}(l(e)) \) in the following way. Take \((S_a, T_a)\) to be the refinement of \((S ∪ e, T)\) which is exactly like \((S, T)\) on the sub-configuration \((S, T)\) and for the new event \(e\) it uses the above ST-configuration \((g, \emptyset)\), which is non-empty. This
new ST-configuration refinement has extra to the event \((e,g)\). Because \(((S,T),(S',T'),f) \in R\) then a matching step exists \((S',T') \xrightarrow{f} (S' \cup f'(e), T')\), where \(f'\) extends \(f\) and is also an isomorphism, hence preserving the label of \(e\). In consequence, we can find the refinement \((\tilde{S}_a, \tilde{T}_a)\) of \((S' \cup f'(e), T')\) to be the same as \((\tilde{S}', \tilde{T}')\) and for the new event \(f'(e)\) choose the same non-empty minimal ST-configuration \((g, \emptyset)\). One can show that \(((\tilde{S}_a, \tilde{T}_a), (\tilde{S}_a', \tilde{T}_a'), \tilde{f}') \in \tilde{R}\) and \(\tilde{f}'\) extends \(\tilde{f}\) from the unrefined case. All the restrictions from the definition of \(R\) are satisfied and the extension of the isomorphism is the case because \(\tilde{f}'((e,g)) = (f'(e), g)\).

(c) When we work with a t-step which is formed from a t-step in the refinement and the same \((S,T)\) in the original ST.

(d) When we work with a t-step which is formed from a t-step in the original ST, and a maximal t-step in the refinement.

3. The third requirement of Definition 2.20 is symmetric to the one above.

By this point we have proved that the history preserving bisimulation alone is preserved under action refinement, because in the proof we did not make reference to the backwards step requirements on the original \(R\). Moreover, when working with two bisimilar refinement functions we again did not make reference to the backwards steps.

4. Proving the forth requirement of Definition 2.20, i.e., for backwards steps.

Similar arguments as before are used only that we remove events, instead of adding. The same cases need to be considered depending on what kind of steps we are working with.

\[\square\]

From the proof of the above proposition one gets also that the history preserving bisimulation is preserved under refinement. The proof can also be extended to show that these bisimulations are congruences for action refinement.

5. Expressiveness shortcomings through examples

Example 5.1 (speed game of angelic vs. demonic choice).

Consider a game (modelled in Figure 8(middle)) of two players, where the “demon” player and the “angel” player each are having a task to do, i.e., the events labelled by \(d\) respectively \(a\). This game is about speed and choice between two further actions \(c_1\) and \(c_2\). There are two different points of choice between the same two events: the early choice, at the initial cell, made by the demon, in advance (we do not know which); the second choice point is after both \(a\) and \(d\) have been done.

The lower side represents the classic “demonic choice” between two actions, i.e., before doing \(d\), whereas the upper part represents the classic “angelic choice”. The example puts these two together and differentiates between them depending on the speed of the events labelled \(a\) and \(d\). In HDAs and ST-structures
“speed” (also understood as structure) of events is an important aspect, which interleaving models are missing.

Which choice is to succeed is determined by speed: if the demon finishes his task first, he then can influence the angel into doing the action that he chose in the beginning; but if the angel finishes first, the choice of the demon is not important any more and the angel gets to choose. Both the demon and the angel are starting at the same time and their tasks are going in parallel.

The viewpoint of the demon sees either the front or back sides (i.e., two consecutive squares) depending on his choice. But the model has to consider both, because of uncertainty.

This example cannot be considered in the interleaving world, nor when action splitting is adopted. This example also poses problems for the notion of “cancellation” studied by Pratt [15] because here we would need transient forms of cancellation, not permanent as in his studies.

This example is nicely depicted as a non-degenerate and acyclic HDA in Figure 8(middle). Moreover, this HDA is also its own history-unfolding. This HDA puts together the angelic and demonic choice patterns in a single system. This example is not representable as ST-structures. But it is representable as an STC-structure, as defined further down in Definition 5.3. The computational steps in this STC-structure involve cancellation of both angelic and demonic kind. It is not enough to use simple cancellation steps because when the event finishes then the canceled events must be removed so to give the appropriate angelic choice. Events must be enabled again in such a step.

Example 5.2 (termination). Instead of considering any maximal configuration to be terminal we want to have a more general termination predicate over ST-configurations, as is done in [11, Def.4.1&Def.5.1] for event structures and configuration structures. But there a configuration can be terminal only if it is maximal, which is a natural requirement that we want to stick with. Take now the ST-structure for asymmetric conflict from Fig. 5(middle), on page 51, where
both ST-configurations \((s, s)\) and \((bs, bs)\) should be final, but only \((bs, bs)\) is maximal. The same issue appears for the ST-structures that we discuss below for the two cyclic HDAs from Fig. 9 (middle-right and right). But note though that it is very natural to assign final cells to these two HDAs: they are \(q_4^0\) respectively \(q_3^0\).

The STC-structures, which we define below, make it natural for maximal configurations to be terminal. Take the STC-structure for asymmetric conflict from Fig. 9 (left) where both \((s, s, b)\) and \((bs, bs, \emptyset)\) are maximal.

We define an extension of ST-structures by including the notion of cancellation (similar to Pratt’s [15]), and call this extension STC-structures. These seem richer than ST-structures, acyclic HDAs, or the unrestricted event structures of [9]. STC-structures overcome several shortcomings of ST-structures: the inability to associate a natural termination predicate, as Example 5.2 illustrates; the inability to properly capture the behaviour of HDAs with cycles, cf. Example 5.6; and the inability to distinguish the classical angelic vs. demonic choices, cf. Example 5.5. Some kind of cycles can be captured by ST-structures alone, but some more convoluted cyclic HDAs require notions of cancellation as well, illustrated in Example 5.6. Note also that STC-structures model well the two examples of HDAs from Figure 6 when the dotted transitions are added. But without these transitions, i.e., only over the three events, the STC-structures also equate the two example HDAs.

**Definition 5.3 (STC-structures).**

An STC-configuration over \(E\) is a set triple \((S, T, C)\), with \(S \subseteq E\) finite and \(C \subseteq E\) possibly infinite, respecting the following restrictions:

\[
\text{(start before terminate)} \quad T \subseteq S; \quad \text{(cancellation)} \quad S \cap C = \emptyset.
\]

An STC-structure is a tuple \(\text{STC} = (E, \text{STC}, l)\), with \(\text{STC}\) a set of STC-configurations over \(E\), the labelling function \(l\) defined as for ST-structures, and
Definition 5.4 (cancellation steps).

A more thorough study of STC-structures is the subject of a future paper. Here we only use them in the examples to show how the inabilities of ST-structures illustrated by these examples could be overcome by STC-structures. The intuitive understanding of STC-structures should be enough for the examples. One can see the steps from ST-structures extended in such a way that the set $C$ of final events, where an event cancels all remaining events of a repetition, like in Example 5.6. This would motivate the cancellation steps where more events may be canceled at once. But the Example 5.1 suggests that such cancellation steps may not be enough, and the preferred would be those that also enable events.

Note also the extra condition in the $\overline{n_{cs}}$ step which ensures that currently canceled events cannot be started. This condition is implicit (deducible) in the other kinds of steps, but for this particular kind of step it may be that the same event that is started belongs to the canceled events and is removed (enabled) in the current step; this would result in a correct STC-configuration. We disallow such steps.

Example 5.5 (angelic vs. demonic choice). The simple example, used by Pratt [13, sec.3.3], of angelic vs. demonic choice at the level of the events (not
actions) cannot be captured in the general event structures of \([9]\). This example involves three events, with \(c_1\) and \(c_2\) conflicting (i.e., a choice between them is made) and are causally depending on \(d\). Branching semantics normally distinguishes two systems depending on when the choice is made. Their STC-structures are given in Figure 10 where the (left) makes a late choice and the (right) an early choice. But when removing the third component \(C\) of all the STC-configurations, the two resulting ST-structures are the same. No ST-structure over three events can make this distinction between the two kinds of choices.

Example 5.6 (shutdown-backup). Initially one may model a Linux-like system abstractly using two events labelled \(s\) for shutdown and \(b\) for backup. At later, more concrete stages these actions may be refined in processes with more structure, eg., as part of a shutdown various actions are performed, like closing web or database services. The \(b\) and \(s\) are considered as being done concurrently, but at this abstract level of modelling the only clear constraint that we have is that \(s\) must wait for \(b\) to finish, before itself may finish. This does not mean that we first perform the \(b\) and after it is finished we start the \(s\), as in the interleaving case of Fig. 9(left). This also does not mean that we cannot have \(s\) alone, without the backup ever being started.

This example is modelled in Fig. 9(middle-left) as the three sides of the square for the asymmetric conflict but with the inside filled in, to model the fact that the two actions can happen concurrently. This example cannot be captured in the event structures of \([9]\), nor configuration structures, nor adjacent-closed ST-structures, nor non-degenerate HDAs. This example is not adjacent-closed. As a HDA this is degenerate because one of the \(t\) maps of the inside cell is missing. Moreover, this example is not closed under unions nor intersections, but it is rooted and connected.

This example is naturally extended to one involving cycles in HDAs. We now say that the system performs backup on a constant basis, in a loop. But the shutdown may be issued at any time point. Therefore, we model the shutdown as happening concurrently with all the backup events. As soon as a shutdown is started, the currently running backup (if any) is allowed to finish, but no other backups may start before the system shuts down. (Naturally, no more backup can be performed after a shutdown).

This is not a simple parallel composition \(s || b^*\), which is modelled by the cyclic HDA of Fig. 9(right), but it is modelled as the cyclic HDA from Fig. 9(middle-
right) which is like the parallel square but with the two lower corners \( q_1^3 = q_0^3 \) equated. Both these HDAs are non-degenerate. The (right) one can be encoded as an ST-structure over the set of events \( \{s\} \cup \{b_i \mid i \in \mathbb{N}^+\} \) by thinking of unfolding the cylinder HDA into infinitely many copies of the parallel square attached one after the other. An unfolding for the (middle-right) example is not easy to see. But it is naturally encoded as an STC-structure over the same set of events, but where the \( C \) component of the STC-configurations takes care of the cancellation of infinitely many copies of the backup events. The complete description is given in the extended version [18], but intuitively, whenever the \( s \) is executed it cancels all the remaining backup events, i.e., those that do not appear already in the \( S \) or \( T \) sets; until then \( b_i \) events can happen in sequence.

Essential is that when removing the \( C \) component from the above STC-structure we obtain an ST-structure isomorphic to the one for the \( s \| b^* \). Therefore, the two cyclic HDAs cannot be distinguished using ST-structures, even before thinking of termination.

6. Conclusion

The work reported here was started in [22] where the notion of ST-structures was first defined. Nevertheless, the work in [22] is mostly concerned with investigations into the higher dimensional modal logic with past modalities and its relations to HDAs and their bisimulations; whereas ST-structures get only little attention. In contrast, the present paper concentrates solely on ST(C)-configuration structures, investigating their expressiveness and relationship with existing concurrency formalisms, including HDAs [1, 2], configurations structures [8, 11], and general (or unrestricted as we call them) event structures [9]. We gave definitions of various notions for ST(C)-structures like steps and paths, bisimulations, or action refinement, and discussed their relationships with similar notions for existing models of concurrency.

Having a good understanding of ST-structures (and their extension STC-structures) would help tackle the problem posed by Pratt in [2] of getting a better understanding of the cyclic structure of HDAs wrt. event based models. This in turn would give a better understanding of the state-event duality in concurrency models described in [12, 15].

Interesting further investigations will ask how the correlations of Sections 3 can be expressed with category theory, following the works of Winskel and Nielsen [38] and of Cattani and Sassone [35], also trying to see the connections with the category of Chu spaces of Gupta [13], the categorical work on cubical sets of Goubault and Mimram [4], and the category of HDAs of Fahrenberg [23].

The results in Section 3 reveal connections and distinctions between the existing concurrency models of HDAs, configuration structures, pure event structures, and unrestricted event structures (and eventually general Petri nets). Some of these results are useful because they show existing knowledge in the new light given by the ST-structures. In particular, Corollary 3.14 shows the difference between configuration structures and unrestricted event structures to
be the fact that configuration structures are filled-in in essence, whereas unrestricted event structures are not. This also explains the counter example of the filled in concurrency square and the empty version, where the latter cannot be captured by pure event structures, but only by the unrestricted case. But both pure and unrestricted event structures are adjacent-closed. Because of this, when dropping the adjacent-closure constraint the ST-structures become more expressive, and the example from Fig. 2(right) models a natural concurrent system that falls into this category. Moreover, the correspondence between acyclic and non-degenerate HDAs and ST-structures that are adjacent-closed comes to say that non-degeneracy and adjacent-closure are close connected. Moreover, coupled with the above it comes to say that these HDAs can capture the unrestricted event structures, as well as suggesting a tighter correlation between these two by allowing acyclic and non-degenerate HDAs to be encoded into unrestricted event structures.

There are thought various examples that break either the acyclic constraint (as the ones in the last part of the paper) or that break the non-degeneracy constraint (as the one from Fig. 4(right-most) or from Figure 9(ii)). These examples find natural representations as ST-structures or as STC-structures. The geometric interpretation of these examples is still very natural, only that the geometric objects fall outside the definition of HDAs as we gave here in either the sense that we do not work with cubes any more but with triangles, or the geometric objects are open as boundaries are missing. It would be useful to investigate more these degenerate or cyclic geometric structures in the same line as started here, by looking at the HDA state-based model which is close to the standard finite state machines used in computer science, and looking at the event-based models of ST(C)-structures and the related configuration and event structures.

Another point that ST-structures and the results in Section 3 make (especially Corollary 3.20 and the results relating to unrestricted event structures) is that ST-structures make the transitions more fine-grained. This says that if for unrestricted event structures a concurrent transition implies that all the possible interleavings exist, for ST-structures a concurrent transition just says that the respective events are running at the same time, i.e., they overlap at least on some part of their execution (this is represented by the fact that those events are in the started stage but not terminated yet). On top of this concurrency aspect more constraints can be put on which events start first and which end before which. Such fine-graining cannot be achieved with the other concurrency models that we compare with in this paper.

Acknowledgements: The work on ST-structures started from a fruitful conversation with Luca Aceto. Olaf Owe and reviewers from FoSSaCS'13 and CONCUR'13 helped improve this draft.
References


57
car, M. Kretínský, A. Kucera (Eds.), 13th International Conference on
Concurrency Theory (CONCUR’02), Vol. 2421 of Lecture Notes in Com-

University (1994).

[14] V. R. Pratt, Chu spaces and their interpretation as concurrent objects,
in: J. van Leeuwen (Ed.), Computer Science Today: Recent Trends and
Developments, Vol. 1000 of Lecture Notes in Computer Science, Springer,

[15] V. R. Pratt, Transition and cancellation in concurrency and branch-
doi:10.1017/S0960129503004031

[16] R. J. van Glabbeek, F. W. Vaandrager, The Difference between Split-

[17] R. J. van Glabbeek, Petri nets, configuration structures and higher
dimensional automata, in: J. C. M. Baeten, S. Mauw (Eds.), 10th
International Conference on Concurrency Theory (CONCUR’99), Vol.
doi:10.1007/3-540-48320-9_3


F. Laroussinie (Eds.), 21st International Conference on Concurrency The-
ory (CONCUR’10), Vol. 6269 of Lecture Notes in Computer Science, Springer,
2010, pp. 147–161. doi:10.1007/978-3-642-15375-4_11

[20] I. Phillips, I. Ulidowski, A logic with reverse modalities for history-
preserving bisimulations, in: B. Luttik, F. Valencia (Eds.), 18th Interna-
tional Workshop on Expressiveness in Concurrency (EXPRESS’11), Vol. 64
of Electronic Proceedings in Theoretical Computer Science, Open Publish-
ing Association, 2011, pp. 104–118. doi:10.4204/EPTCS.64.8

[21] C. Prisacariu, Modal logic over higher dimensional automata, in: P. Gastin,
F. Laroussinie (Eds.), 21st International Conference on Concurrency The-
ory (CONCUR’10), Vol. 6269 of Lecture Notes in Computer Science, Springer,
2010, pp. 494–508. doi:10.1007/978-3-642-15375-4_34

[22] C. Prisacariu, The glory of the past and geometrical concurrency, in:
A. Voronkov (Ed.), The Alan Turing Centenary Conference (Turing-100),


