ST-Configuration Structures

Cristian Prisacariu

Dept. of Informatics, Univ. of Oslo, P.O. Box 1080 Blindern, N-0316 Oslo, Norway
cristi@ifi.uio.no

1 Introduction

The present paper defines ST-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 (HDA), a *state-based* approach of Pratt and van Glabbeek; and the configuration structures and (in)pure 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 can be defined for ST-structures, like steps and paths, bisimulation, and action refinement, and related results can be found in the extended version. 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.

ST-configuration structures are a natural extension of configuration structures to the setting of higher dimensional automata. Configuration structures [6] are used in [5] as the most expressive model of concurrency which has a natural way of defining refinement and the partial order bisimulations, like history preserving bisimulation. The notion of an ST-configuration has been used in [7] to define ST-bisimulation and in [4] in the context of *HDA*. But the model of *ST-configuration structures*, as we define here for capturing concurrency, does not appear elsewhere. ST-structures have the power to look at the currently executing concurrent events. This captures a main characteristic of higher dimensional automata, which cannot be captured by standard models like configuration structures.

2 ST-configuration structures

Definition 2.1 (ST-configuration)**.** *An ST-configuration is a pair of sets (S, T) with the following property:*

$$\textit{(start before terminate)}\quad T \subseteq S.$$

Intuitively S contains the events that have started and T the event that have terminated.

Definition 2.2 (ST-configuration structures)**.** *An* ST-configuration structure *(also called* ST-structure*) is a tuple $\mathsf{ST} = (ST, l)$ with ST a set of ST-configurations and l a labelling function of the events, $l : \bigcup_{S \in ST} S \to \Sigma$, with ST satisfying the constraint: if $(S, T) \in ST$ then $(S, S) \in ST$.*

The constraint above is a closure so that we do not represent events that are started but never terminated.

ST-structures have a natural *computational interpretation* (on the same lines as configuration structures) as *steps* between ST-configurations, and *paths*. Our results show that this computational interpretation is more fine-grained than for other models we compare 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). Action refinement and bisimulation are well behaved wrt. this interpretation. The model of *HDAs* do the same job but in the state-based setting. Besides, ST-structures exhibit a natural *observable information* (on the same lines as for *HDAs*) as *ST-traces*, which, cf. [4, Sec.7.3], constitute the best formalization of observable content.

Definition 2.3 (ST steps)**.** *A step between two ST-configurations is defined as either:*
s-step $(S,T) \xrightarrow[s]{a} (S',T')$ *when* $T = T'$, $S \subset S'$, $S' \setminus S = \{e\}$ *and* $l(e) = a$; *or*
t-step $(S,T) \xrightarrow[t]{a} (S',T')$ *when* $S = S'$, $T \subset T'$, $T' \setminus T = \{e\}$ *and* $l(e) = a$.
When the type is unimportant we denote a step by $\xrightarrow{a}$ *for* $\xrightarrow[s]{a} \cup \xrightarrow[t]{a}$.

Definition 2.4 (stable ST-structures)**.** *An ST-structure* $\mathsf{ST} = (ST, l)$ *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 *(respectively* bounded intersections*) iff for any* (S,T), (S',T'), $(S'',T'') \in ST$ *s.t.* $(S,T) \cup (S',T') \subseteq (S'',T'')$, *then* $(S,T) \cup (S',T') \in ST$ *(rsp.* $(S,T) \cap (S',T') \in ST$*).*

An ST-structure is called stable *iff it is rooted, connected, and closed under bounded unions and intersections.*

Definition 2.5 (adjacent-closure)**.** *An ST-structure* ST *is* 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* $(S,T),(S \cup e,T),(S \cup e,T \cup e') \in \mathsf{ST}$, *with* $e \notin S \wedge e' \notin T \wedge e \neq e'$, *then* $(S,T \cup e') \in \mathsf{ST}$;
3. *if* $(S,T),(S \cup e,T),(S,T \cup e') \in \mathsf{ST} : e \notin S \wedge e' \notin T \wedge e \neq e'$, *then* $(S \cup e,T \cup e') \in \mathsf{ST}$;
4. *if* $(S,T),(S,T \cup e),(S,T \cup \{e,e'\}) \in \mathsf{ST}$, *with* $(e \neq e') \notin T$, *then* $(S,T \cup e') \in \mathsf{ST}$.

Knowing the definition of higher dimensional automata (see [1,2,4]) 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 [4, Def.19] which is used to define homotopy over higher dimensional automata. 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.

The standard example of the square with the empty inside is adjacent-closed but not closed under unions nor under intersections. The example of the parallel switch of Winskel [8] is adjacent-closed and closed under unions, but not closed under intersections (can be pictured as only three sides of a cube in *HDA*).

Definition 2.6 (concurrency and causality)**.** *For a particular ST-configuration* $(S,T) \in \mathsf{ST}$ *define the relations of* concurrency *and* causality *on the events in* S *as:*

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

causality *for* $e,e' \in S$ *then* $e < e'$ *iff* $e \neq e' \wedge \forall (S',T') \subseteq (S,T) : (S',T') \in \mathsf{ST} \Rightarrow (e' \in S' \Rightarrow e \in T')$.

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.

On arbitrary ST-structures the concurrency and causality are not interdefinable (in a standard way e.g. [5, Def.5.6] where concurrency is the negation 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 [5, Sec.5.3] can be stated and proven about stable ST-structures and their causality partial order.

Definition 2.7 (cf. [5, Def.5.1] [6, Def.1.1])**.** *A configuration structure $\mathsf{C} = (E, C)$, is formed of a set E of events and a set of configurations which are subsets of events $C \subseteq 2^E$. A* labeled *configuration structure also has a labeling function of its events, $l : E \to \Sigma$.*

Proposition 2.8. *If an ST-structure ST is rooted, connected, or closed under bounded unions, or intersections, then the corresponding $\mathsf{C}(\mathsf{ST})$ is respectively rooted, connected, closed under bounded unions, or intersections. The mapping $\mathsf{C} : \mathbb{ST} \to \mathbb{C}$ is defined to associate to every ST-structure ST a configuration structure by keeping only those ST-configurations that have $S = T$; i.e., $\mathsf{C}(\mathsf{ST}) = \{T \mid (S, T) \in \mathsf{ST} \wedge S = T\}$, which preserves the labeling.*

Theorem 2.9. *Configuration structures are strictly embedded into ST-structures. Define a mapping $\mathsf{ST} : \mathbb{C} \to \mathbb{ST}$ that associates to every configuration structure C an ST-structure $\mathsf{ST}(\mathsf{C})$ by associating to each configuration $X \in \mathsf{C}$ an ST-configuration $\mathsf{ST}(X) = (X, X) \in \mathsf{ST}(\mathsf{C})$ and for each transition $X \to_C Y \in \mathsf{C}$ an ST-configuration $\mathsf{ST}(X \to_C Y) = (Y, X) \in \mathsf{ST}(\mathsf{C})$. This map ST preserves the asynchronous concurrent steps of the configuration structure, i.e., for each asynchronous step $X \to_C Y \in \mathsf{C}$ there is a chain of single steps in the ST-structure $\mathsf{ST}(\mathsf{C})$ that passes through (Y, X) (thus signifying the concurrent execution of all events in $Y \setminus X$).*

Corollary 2.10. *An ST-structure $\mathsf{ST}(\mathsf{C})$ generated as in Theorem 2.9 is adjacent-closed (though not necessarily closed under bounded unions nor intersections).*

Corollary 2.11. *In an ST-structure $\mathsf{ST}(\mathsf{C})$ generated as in Theorem 2.9 the ST-configurations with $S = T$ correspond exactly to the configurations of C. That is to say that $\mathsf{C}(\mathsf{ST}(\mathsf{C})) \cong \mathsf{C}$.*

Corollary 2.12. *The ST-structure obtained in Th. 2.9 is* "filled in"*, in the sense that any cube is filled in. By a "cube" it 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.*

But there is not a one to one correspondence between ST-structures and the configuration structures because there can be several ST-structures that have the same configuration structure. The example is of one *HDA* square that is filled in and one that is not; 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.

Proposition 2.13. *For stable and adjacent-closed ST-structures and stable configuration structures there is a one-to-one correspondence. (The adjacency is necessary.)*

Similar connections are investigated wrt. the *(impure) event structures* of [6] and wrt. the higher dimensional automata of [3, 4].

References

[1] V. R. Pratt. Modeling concurrency with geometry. In *POPL'91*, pages 311–322, 1991.

[2] V. R. Pratt. Higher dimensional automata revisited. *MSCS*, 10(4):525–548, 2000.

[3] V. R. Pratt. Transition and Cancellation in Concurrency and Branching Time. *MSCS*, 13(4):485–529, 2003.

[4] R. van Glabbeek. On the Expressiveness of Higher Dimensional Automata. *TCS*, 356(3):265–290, 2006.

[5] R. van Glabbeek & U. Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37(4/5):229–327, 2001.

[6] R. van Glabbeek & G. Plotkin. Configuration structures, event structures and Petri nets. *TCS*, 410(41):4111–4159, 2009.

[7] R. van Glabbeek & F. Vaandrager. The Difference between Splitting in n and n+1. *Inf. Comput.*, 136(2):109–142, 1997.

[8] G. Winskel. Event structures. In *Advances in Petri Nets*, volume 255 of *LNCS*, pages 325–392. Springer, 1986.