What is in a Step: New Perspectives on a Classical Question*

Willem-Paul de Roever[1], Gerald Lüttgen[2], and Michael Mendler[2]

[1] Institute of Computer Science and Applied Mathematics,
Christian-Albrechts-University of Kiel, Germany,
wpr@informatik.uni-kiel.de

[2] Software Technologies and Informatics Theory Research Groups,
Otto-Friedrich-University of Bamberg, Germany,
gerald.luettgen@swt-bamberg.de, michael.mendler@uni-bamberg.de

Abstract. In their seminal 1991 paper "What is in a Step: On the Semantics of Statecharts", Pnueli and Shalev showed how, in the presence of global consistency and while observing causality, the synchronous language Statecharts can be given coinciding operational and declarative step semantics. Over the past decade, this semantics has been supplemented with order-theoretic, denotational, axiomatic and game-theoretic characterisations, thus revealing itself as a rather canonical interpretation of the synchrony hypothesis.

In this paper, we survey these characterisations and use them to emphasise the close but not widely known relations of Statecharts to the synchronous language Esterel and to the field of logic programming. Additionally, we highlight some early reminiscences on Amir Pnueli's contributions to characterise the semantics of Statecharts.

1 Introduction

One of the many contributions of Amir Pnueli to the field of Computer Science is in the semantics of *Statecharts* [27, 28, 25, 51, 15]. Statecharts is a popular language for specifying and developing reactive systems, which was invented by Harel in the early 1980s [22]. It extends Mealy machines with concepts of (i) hierarchy, so that a state may have sub-states; (ii) concurrency, thus allowing states to have simultaneously active sub-states that may communicate via the broadcasting of events; (iii) priority, such that one may express that certain events have priority over others. The novelty at the time was that Statecharts is a visual and executable language that it easily understood by software and systems engineers. It is one of the earliest examples of its kind that embraces model-driven software engineering [23]. Within a decade of its inception, already two dozen variants of Statecharts existed [5]. Some of today's widely used variants are the original *STATEMATE* [25], Matlab/Simulink's *Stateflow* [47], and the UML state-machine dialect *Rhapsody* [24].

* The initial ideas leading to this paper were developed by the authors during the Dagstuhl Seminar 09481 (Synchron 2009) in November 2009.

Towards a semantics. Statecharts belongs to the family of *synchronous* languages which also includes, e.g., Esterel, Signal, Lustre and Argos [6]. Their semantics is based on a cycle-based reaction whereby the events input by the system's environment are sampled first and potentially cause the firing of transitions that may produce new events. The generated events are output to the environment when the reaction cycle ends. The *synchrony hypothesis* [7], which is adopted by all synchronous languages, ensures that this potentially complex, non-atomic reaction is bundled into an atomic *step*. The hypothesis is justified in practice by the fact that reactions can typically be computed much quicker than it takes for new events to arrive from the system's environment.

Obviously, this concept of a step-based reaction still offers several choices as to what exactly constitutes a step [31, 32, 44, 16, 17]. One important choice is whether generated events may be sensed only in the next step, or already in the current step and may thus trigger the firing of further transitions. The first option was adopted by Harel et al. in the "official" but non-compositional semantics of Statecharts as is implemented in STATEMATE [25, 26, 29]. STATEMATE steps are typically run to completion via its so-called super-step semantics for which Damm, Josko, Hungar and Pnueli have later proposed a compositional variant [15] that supports modular system verification [8].

The second option was investigated by Harel, Pnueli, Schmidt and Sherman [28], where a step involves a causal chain of firing transitions. Here, a transition fires if the positive events in its trigger are present (i.e., if they are offered by the system environment or have been generated by a transition that has fired previously in the same step) and if its negative trigger events are absent (i.e., if they are not present). When it fires, the transition may, as part of its action, broadcast new events which, by the principle of *causality*, may trigger further transitions. Only when this chain reaction of firing transitions comes to a halt is a step complete and becomes, according to the synchrony hypothesis, an atomic entity. Unsurprisingly, the semantics of [28] is not compositional since bundling transitions into an atomic step implies forgetting about the transitions' causal justification [32]. This shortcoming has later been remedied in a fully-abstract fashion by Huizing, Gerth and de Roever [33]. In addition, the semantics of [28] is not *globally consistent* as it permits an event to be both present and absent within a step: an event that occurs negatively in the trigger of one firing transition may be generated by a transition that fires later within the same step.

Pnueli & Shalev's contribution. In the words of Pnueli and Shalev, "a proven sign of healthy and robust understanding of the meaning of a programming or a specification language is the possession of both an operational and declarative semantics, which are consistent with one another" [51]. They showed in their seminal paper (cf. Sec. 2) that adding global consistency is the key to achieving this ambitious goal for Statecharts, and this meant to move away from the semantics of [28]. Their operational semantics for Statecharts relies on an iterative *fixed point construction* over a non-monotonic enabledness function for transitions. This construction ensures causality but involves backtracking as soon as a global inconsistency is encountered; in the extreme, this may imply that a

Statechart does not possess any step under the considered input by the environment. Pnueli and Shalev's declarative semantics for Statecharts then identifies the desired fixed points of the enabledness function via a notion of *separability.*

Levi later developed a compositional proof system for the Pnueli-Shalev step semantics in terms of the modal μ-calculus [36] which facilitates the modular verification of Statecharts. A variant of Pnueli-Shalev semantics which disables transitions that may introduce global inconsistency was presented by Maggiolo-Schettini, Peron and Tini in [43]. This semantics was also used in an early axiomatisation of Statecharts by Hooman, Ramesh and de Roever [30].

This paper. Whereas, to the best of our knowledge, the Pnueli-Shalev step semantics has never been implemented in a Statecharts tool, its mathematical elegance has attracted attention by the concurrency theory community. Over the past decade, the semantics has been supplemented with order-theoretic, denotational, algebraic and game-theoretic perspectives, thus further testifying to its robustness (cf. Sec. 3). The *order-theoretic semantics* of Levi [36] fixes the lacking compositionality of the Pnueli-Shalev step semantics by encoding the causality relation between a step's firing transitions via an irreflexive ordering relation. The *denotational semantics* [41] also addresses the compositionality problem and does so in a fully-abstract way. It is based on an intuitionistic logic interpretation of steps, which appreciates the possibility that an event may neither be present nor absent in a step, but that it may be introduced by the system's environment in the middle of a step. The *algebraic semantics* [40] characterises this fully-abstract denotational semantics in terms of equations, thus leading to a step algebra. Finally, the *game-theoretic semantics* [1] interprets Pnueli-Shalev steps via winning strategies in a 2-player maze game.

Other than revealing the Pnueli-Shalev semantics as a rather canonical interpretation of Statecharts steps, the characterisations mentioned above have opened the door for a mathematically exact comparison of Statecharts steps to *Esterel reactions* and for relating Statecharts to *logic programming* (cf. Sec. 4). Esterel is another popular synchronous language that was devised by Berry independently of, but at the same time as, Statecharts [52]. Its semantics differs from the one proposed by Pnueli and Shalev only by the interpretation of negative events. While one may speculate in Statecharts for an event to be absent, the absence of events in Esterel must be proved constructively, which is key to the language's determinism [42]. Negation is also widely studied in the field of logic programming where *stable negation* [49] corresponds to Pnueli and Shalev's reading of negative events in the presence of global consistency.

The aims of this paper are to (i) survey these additional perspectives on the Pnueli-Shalev semantics; (ii) highlight the semantic relationship between Statecharts and Esterel, and between Statecharts and logic programming; (iii) discuss Pnueli and Shalev's results in the light of related work. This offers new insights into *the* classical question in the Statecharts literature: *"What is in a step?"* Last, but not least, the first author recalls some reminiscences on Amir Pnueli's contributions to characterise the semantics of Statecharts (cf. Sec. 5).

2 Pnueli and Shalev's Interpretation of Statecharts

This section provides a brief introduction to Statecharts, and recalls the step semantics presented by Pnueli and Shalev in [51].

2.1 Introduction to Statecharts

A Statechart may best be understood as a hierarchical, concurrent Mealy machine, where *basic* states may be hierarchically refined by injecting other Statecharts. This creates composite states of two possible sorts, which are called *and*-states and *or*-states, respectively. Whereas and-states permit parallel decompositions of states, or-states allow for sequential decompositions. Consequently, an and-state is *active* if all its sub-states are active, and an or-state is active if exactly one of its sub-states is. At any given point during execution, the set of active states is referred to as a *configuration*.

A Statecharts step is defined relative to a configuration C and a set E of events that are given to the system by its environment. Key to a step are transitions t, each of which is labelled by two sets of events: a *trigger*, $\mathsf{trg}(t)$, and an *action*, $\mathsf{act}(t)$. The trigger $\mathsf{trg}(t) = P, \overline{N}$ splits into a set of *positive* events $P \subseteq \Pi$ and *negative* events $\overline{N} \subseteq \overline{\Pi}$, taken from a universe Π of events and their negative counterparts in $\overline{\Pi} =_{\mathrm{df}} \{\overline{e} : e \in \Pi\}$, respectively. For convenience, we define $\overline{\overline{e}} =_{\mathrm{df}} e$. Intuitively, t is *enabled* and thus fires if the set $E \subseteq \Pi$ is such that all events of P but none of N are in E, i.e., if $P \subseteq E$ and $N \cap E = \emptyset$. The effect of firing t is the generation of all events in the action of t, where a transition's action $\mathsf{act}(t) \subseteq \Pi$ consists of positive events only.

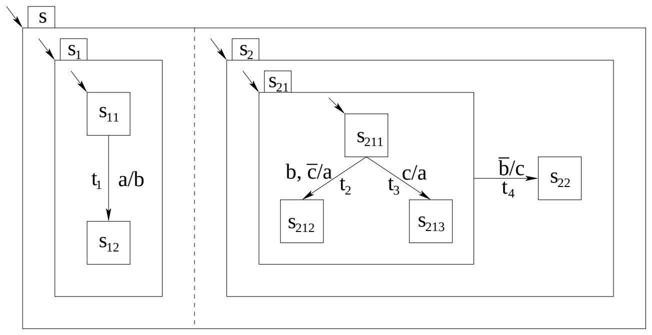


Fig. 1. Example Statechart.

We illustrate the Statecharts language by means of a small example. Consider the Statechart depicted in Fig. 1 with and-state s, or-states s_1, s_2 and s_{21}, and *basic*-states s_{11}, s_{12}, s_{211}, s_{212}, s_{213} and s_{22}. Further assume that all components are in their initial states marked by the small unlabelled arrows, so that the initial configuration C_1 is $\{s, s_1, s_{11}, s_2, s_{21}, s_{211}\}$. If, in this configuration, the environment offers event c, then transitions t_3 and t_4 are enabled. Since they are both placed within the same or-state s_2, only one of them may fire. In a Statecharts dialect that does not give transitions an implicit priority along the state hierarchy, the choice between t_3 and t_4 is *nondeterministic*. Thus, t_3 may fire, generate event a, and change state to s_{213}. Again depending on the Statecharts dialect, this generated event may or may not trigger transition t_1 in the parallel, or orthogonal, state s_1 within the same reaction cycle, i.e., within the same *step*. Hence, the question arises which transitions leaving states in C_1, which we denote by $\mathsf{trans}(C_1)$, may fire together to form a step.

2.2 The Pnueli-Shalev Step Semantics

As stated in the introduction, Pnueli and Shalev defined coinciding operational and declarative semantics of Statecharts configurations in their paper [51]. Given a configuration C, a step in the sense of Pnueli and Shalev comprises a *maximal*, globally consistent and causal, set of transitions in $\mathsf{trans}(C)$, which are mutually *orthogonal*, i.e., "consistent" in Statecharts terminology, and *triggered* by the events offered by the environment or produced by the firing of other transitions in the step.

Transition t is *consistent* with set T of transitions, in signs $t \in \mathsf{consistent}(C, T)$, if t is not in the same "parallel component" as any $t' \in T \setminus \{t\}$. Formally,

$$\mathsf{consistent}(C, T) =_{\mathrm{df}} \{t \in \mathsf{trans}(C) \mid \forall t' \in T.\ t \triangle_C t'\}\ ,$$

where $t \triangle_C t'$ if (i) $t = t'$ or (ii) t and t' are in different substates of an enclosing and-state. Further, transition t is *triggered* by a set E of events, in signs $t \in \mathsf{triggered}(C, E)$, if the positive but not the negative trigger events of t are in E:

$$\mathsf{triggered}(C, E) =_{\mathrm{df}} \{t \in \mathsf{trans}(C) \mid \mathsf{trg}(t) \cap \Pi \subseteq E,\ \overline{(\mathsf{trg}(t) \cap \overline{\Pi})} \cap E = \emptyset\}\ .$$

Finally, transition t is *enabled* in C with respect to set E of events and set T of transitions, if $t \in \mathsf{enabled}(C, E, T)$ where

$$\mathsf{enabled}(C, E, T) =_{\mathrm{df}} \mathsf{consistent}(C, T) \cap \mathsf{triggered}(C, E \cup \bigcup_{t \in T} \mathsf{act}(t))\ .$$

Assuming the transitions in T are known to fire, $\mathsf{enabled}(C, E, T)$ determines the set of all transitions of C that are enabled by the environment events in E and, since generated events are sensed within the same step, the actions of T. In the following, we write $\mathsf{act}(T)$ for the actions $\bigcup_{t \in T} \mathsf{act}(t)$.

Operational semantics. Using this enabledness function $\mathsf{enabled}$, we may now present the *step-construction procedure* of [51] which operationally determines Statecharts steps relative to a configuration C and a set E of environment events:

```
procedure $step\text{–}construction(C,\ E)$;
   var $T := \emptyset$;
   while $T \subset$ enabled$(C, E, T)$ do
      choose $t \in$ enabled$(C, E, T) \setminus T$;
      $T := T \cup \{t\}$
   od;
   if $T =$ enabled$(C, E, T)$ then return $T$
   else $report\ failure$
end $step\text{–}construction$.
```

This step-construction procedure computes sets T of transitions that can fire together in a step. Returning to our example, i.e., the Statechart depicted in Fig. 1, we have $\mathsf{enabled}(C, \{c\}, \emptyset) = \{t_3, t_4\}$. Therefore, $step\text{–}construction(C_1,\ \{c\})$ may choose transition t_3 in its first iteration and, since $\mathsf{enabled}(C, \{c\}, \{t_3\}) = \{t_1, t_3\}$, transition t_1 in its second iteration, before reaching a fixed point and returning $\{t_1, t_3\}$. Due to the presence of statement choose, the procedure may introduce nondeterminism. Indeed, $step\text{–}construction(C_1,\ \{c\})$ may also return $\{t_4\}$ when choosing t_4 instead of t_3 in the first iteration.

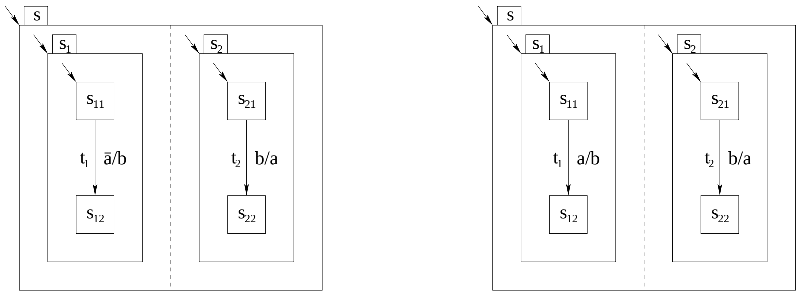


Fig. 2. Further example Statecharts.

When the procedure reports a failure as the result of detecting an inconsistency, i.e., there exists a $t \in T$ such that $t \notin \mathsf{enabled}(C, E, T)$, it may backtrack and possibly make a different choice at statement choose. In particular, the procedure may only report failures and not produce any step. To see this, consider the Statechart shown on the left in Fig. 2 in its initial configuration C_2, and assume the empty environment. In its first iteration, $step\text{–}construction(C_2,\ \emptyset)$ picks the only enabled transition t_1, and then the other transition t_2 in the second iteration. At this point $T = \{t_1, t_2\}$ but $\mathsf{enabled}(C_2, \emptyset, T) = \{t_2\}$, and a failure is reported. No backtracking is possible since there have not been any nontrivial choice points along the computation. Hence, the step-construction procedure does not produce any step. This situation is to be distinguished from an empty response $T = \emptyset$, which is exhibited by the Statechart depicted on the right in Fig. 2 in its initial configuration C_3 and for the empty environment, since $\mathsf{enabled}(C_3, \emptyset, \emptyset) = \emptyset$.

Following Pnueli and Shalev's terminology, a set T of transitions is called *constructible* for a given configuration C and a set E of environment events, if it can be obtained as a result of successfully executing procedure *step-construction*. For each constructible set T, set $A =_{\mathrm{df}} E \cup \mathsf{act}(T) \subseteq \Pi$ is called the *(step) response* of C for E.

Declarative semantics. Pnueli and Shalev also provided an equivalent *declarative* definition of their operational step semantics. Given a configuration C and a set E of environment events, a set T of transitions is called *separable* for C and E if there exists a proper subset $T' \subsetneq T$ such that $\mathsf{enabled}(C, E, T') \cap (T \setminus T') = \emptyset$. Further, T is *admissible* for C and E if T is inseparable for C and E and $T = \mathsf{enabled}(C, E, T)$. This declarative semantics is thus a fixed-point semantics. Observe that, in the absence of negative events, function $\mathsf{enabled}(C, E, \cdot)$ is monotonic, and then the uniquely defined inseparable fixed point coincides with the least fixed point. However, since function $\mathsf{enabled}(C, E, \cdot)$ is in general non-monotonic when transitions with negative trigger events are involved, a unique least fixed point may not exist. In this case, the notion of inseparability chooses distinguished fixed points that reflect causality. Indeed, a separable set of transitions points to a break in the causality chain when firing these transitions.

We illustrate the notion of inseparability by returning to the above examples. For the Statechart depicted in Fig. 1, $\{a, b, c\}$ is a step response of initial configuration C_1 for environment $E =_{\mathrm{df}} \{c\}$. Firstly, as seen above, $T =_{\mathrm{df}} \{t_1, t_3\}$ is a fixed point of $\mathsf{enabled}(C_1, E, T)$. Secondly, it is inseparable for C_1 and E since, for $T' = \emptyset$, we have $\mathsf{enabled}(C_1, E, T') \cap (T \setminus T') = \{t_3, t_4\} \cap T = \{t_3\} \neq \emptyset$, and similarly for the other proper subsets $T' \subsetneq T$. For the initial configurations C_2 and C_3 of the two Statecharts of Fig. 2, $\{a, b\}$ is not a step response for the empty environment. For the Statechart on the left in the figure, $T =_{\mathrm{df}} \{t_1, t_2\}$ is not a fixed point of function $\mathsf{enabled}$ since $\mathsf{enabled}(C_2, \emptyset, T) = \{t_2\} \neq T$. For the Statechart on the right, T is separable; consider $T' = \emptyset \subsetneq T$, for which $\mathsf{enabled}(C_3, \emptyset, T') \cap (T \setminus T') = \emptyset \cap T = \emptyset$.

Main result. We can now state the main result of Pnueli and Shalev's paper [51]:

Theorem 1 (Pnueli & Shalev). *For all configurations C and event sets $E \subseteq \Pi$, a set T of transitions is admissible for C and E if and only if T is constructible for C and E.*

Such a theorem can also be proved for the step semantics of Maggiolo-Schettini, Peron and Tini [43]. Their semantics uses a modified function $\mathsf{enabled}(C, E, T)$ in which a transition t is *not* enabled if its firing would generate an event whose absence is assumed in T, i.e., if $\mathsf{act}(t) \cap \bigcup_{t' \in T} \overline{\mathsf{trg}(t')} \neq \emptyset$. For example, the Statechart in Fig. 2 on the left, which did not have any response for $E = \emptyset$, now has response $\{b\}$. This response is obtained by t_1 firing on the basis of a being absent which then immediately disables t_2. One shows that $\{t_1\}$ is an inseparable with $\mathsf{enabled}(C_2, \emptyset, \{t_1\}) = \{t_1\}$.

3 Developments & New Perspectives

This section surveys four characterisations of the Pnueli-Shalev step semantics which have been developed within the past decade: an *order-theoretic* semantics that encodes causality via an irreflexive ordering relation [36]; a *denotational* semantics that is based on intuitionistic logic [41]; an *algebraic* semantics that specialises axioms of this logic to Statecharts steps [40]; and a *game-theoretic* semantics [1]. In contrast to Pnueli and Shalev's operational and declarative semantics, all four semantics presented here are compositional, and the denotational and algebraic semantics are fully-abstract.

3.1 Configuration Syntax

This paper focuses on the semantics of single Statecharts steps, since the semantics across steps is clear and well understood. It will therefore be convenient to reduce the Statecharts notation to the bare essentials and identify a Statecharts configuration with its set of leaving transitions, to which we — by abuse of terminology — also refer as *configuration*. We formalise configurations using the following, simple syntax, where $I \subseteq \Pi \cup \overline{\Pi}$ and $A \subseteq \Pi$:

$$C \quad ::= \quad 0 \mid I/A \mid C \| C \, .$$

Intuitively, 0 stands for the configuration with the empty behaviour. Configuration I/A encodes a transition t with $\mathsf{trg}(t) = I$ and $\mathsf{act}(t) = A$. When triggered, transition t fires and generates the events in A. Transitions I/A with empty trigger, i.e., $I = \emptyset$, are simply written as A below. If we wish to emphasise that trigger I consists of the positive events $P \subseteq \Pi$ and the negative events $\overline{N} \subseteq \overline{\Pi}$, i.e., $I = P \cup \overline{N}$, then we denote transition I/A by $P, \overline{N}/A$. Finally, configuration $C_1 \| C_2$ describes the parallel composition of configurations C_1 and C_2. Observe that 0 coincides semantically with a transition with empty action; nevertheless, it seems natural to include 0. Using this syntax, we may encode the initial configuration C_1 of our example Statechart of Fig. 1 as

$$a/b \parallel b, \overline{c}, \overline{e_3}, \overline{e_4}/a, e_2 \parallel c, \overline{e_2}, \overline{e_4}/a, e_3 \parallel \overline{b}, \overline{e_2}, \overline{e_3}/c, e_4 \, .$$

Here, the e_i are distinguished events not occurring in the triggers or actions of the Statechart's transitions. These events allow us to encode *nondeterministic choice*, including *state hierarchy*, via parallel composition and event negation, although one can do without them as is shown in [41]. Assuming the environment injects event c, Pnueli and Shalev's step-construction procedure may first fire transition t_3 and then t_1 within a single step from configuration C_1, thereby reaching configuration $\{s, s_1, s_{12}, s_2, s_{21}, s_{213}\}$. This latter configuration is represented by $0 \parallel \overline{b}/c$ in our syntax.

For simplicity, the following exposition focuses on Statecharts configurations with respect to the empty environment only. This is not a restriction, however, since considering the steps of a configuration C relative to a set $E \subseteq \Pi$ of environment events is equivalent to considering the steps of the configuration $C \| E$ relative to the empty set of environment events.

3.2 Order-Theoretic Perspective

The first results for turning Pnueli and Shalev's step construction into a compositional semantics for Statecharts were obtained by Uselton and Smolka [56], Levi [36], and Maggiolo-Schettini, Peron and Tini [43, 44]. They observed that Statecharts may be viewed as process terms in the style of process algebra, whose semantics is given by a compositional translation into labelled transition systems. Each transition represents a step of a configuration decorated with an action label specifying the synchronous interaction with the environment. It turned out that for this structured operational semantics to work, labels must be order-relational structures as opposed to simple first-order events, in order to encode sufficient causal information. In this section we recall the basic elements of this order-theoretic approach, following essentially the exposition of Levi in [36], albeit in a simplified form.

The set of *(causality) labels* $\Sigma(\Pi)$, or *basic actions* in the terminology of Levi, is the set of pairs $(\ell, \prec)$. Here, $\ell \subseteq \Pi \cup \overline{\Pi}$ is a *consistent* subset of positive or negative events, i.e., $\ell \cap \overline{\ell} = \emptyset$, and $A \prec B$ is an irreflexive and transitive causality ordering on subsets $A, B \subseteq \ell$, where $B = \emptyset$ or $B = \{b\}$ for $b \in \Pi$. Irreflexivity means that $A \prec \{b\}$ implies $b \notin A$, and transitivity requires that, if $A \prec \{b\}$ and $b \in B \prec C$, then $(B \setminus \{b\}) \cup A \prec C$.

Causality labels represent globally consistent and causally closed interactions that are composed from Statecharts transitions. Specifically, every transition $t \in \mathsf{trans}(C)$ leaving a configuration C induces a causality label $\mathit{lab}(t) =_{\mathrm{df}} (\ell_t, \prec_t)$, where $\ell_t =_{\mathrm{df}} \mathsf{trg}(t) \cup \mathsf{act}(t)$ and $\prec_t =_{\mathrm{df}} \{\mathsf{trg}(t) \prec_t \{e'\} : e' \in \mathsf{act}(t)\}$. It is assumed without loss of generality that transitions are nontrivial in the sense that $\mathsf{trg}(t) \cap \mathsf{act}(t) = \emptyset$ and, for no $e \in \Pi$, both $e, \overline{e} \in \mathsf{trg}(t) \cup \mathsf{act}(t)$. Then, ℓ_t is consistent, $\prec_t$ is irreflexive and, trivially, transitive. For instance, the transitions $t_1 =_{\mathrm{df}} a/b$ and $t_2 =_{\mathrm{df}} b, \overline{c}/d$ correspond to the labels $\sigma_i =_{\mathrm{df}} \mathit{lab}(t_i) = (\ell_i, \prec_i)$ with $\ell_1 = \{a, b\}$, $\{a\} \prec_1 \{b\}$, and $\ell_2 = \{b, \overline{c}, d\}$ with $\{b, \overline{c}\} \prec_2 \{d\}$. The joint execution of t_1 and t_2 would be the label $\sigma_3 =_{\mathrm{df}} (\ell_3, \prec_3)$ such that $\ell_3 = \{a, b, \overline{c}, d\}$ with causalities $\{a\} \prec_3 \{b\}$, $\{b, \overline{c}\} \prec_3 \{d\}$ and $\{a, \overline{c}\} \prec_3 \{d\}$. Here, the last pair arises from the combined reaction of t_1 triggering t_2; its presence is enforced by transitivity of $\prec_3$.

As causality labels are compositional generalisations of transitions, each $\sigma = (\ell, \prec) \in \Sigma(\Pi)$ has an associated set of *trigger* and *action events*, viz., $\mathsf{trg}(\sigma) =_{\mathrm{df}} \{e \in \ell \mid \neg\exists C \subseteq \ell.\, C \prec \{e\}\}$ and $\mathsf{act}(\sigma) =_{\mathrm{df}} \ell \setminus \mathsf{trg}(\sigma)$. Thus, a transition $t = I/A$ has $\mathsf{trg}(\mathit{lab}(t)) = I$ and $\mathsf{act}(\mathit{lab}(t)) = A$ as expected. For the label σ_3 from above, we get $\mathsf{trg}(\sigma_3) = \{a, \overline{c}\}$ and $\mathsf{act}(\sigma_3) = \{b, d\}$, which are the same trigger and action as in $\mathit{lab}(t_4)$, for $t_4 =_{\mathrm{df}} a, \overline{c}/b, d$. However, the latter does not express the causality contained in σ_3, viz., that event b is a consequence of a alone, while d depends on both a and $\overline{c}$. It is this extra causality information which makes labels compositional: σ_3 is the combined execution of t_1 and t_2 as opposed to t_4 which is a single atomic transiton.

Labels, like transition sets, can be enabled or disabled by the environment. A consistent $\ell \subseteq \Pi \cup \overline{\Pi}$ *enables* an action σ if $\mathsf{trg}(\sigma) \cap \Pi \subseteq \ell$ and $\overline{\mathsf{trg}(\sigma)} \cap \ell = \emptyset$. It *disables* σ if $\overline{\mathsf{trg}(\sigma)} \cap \ell \neq \emptyset$. For consistent and *complete* (or *binary*) ℓ, i.e.,

$\Pi \subseteq \ell \cup \overline{\ell}$, both notions are complementary. Note that if $\emptyset$ enables σ then no trigger is needed to execute σ.

Next we define the operation of parallel composition between causality labels $\sigma_1 = (\ell_1, \prec_1)$ and $\sigma_2 = (\ell_2, \prec_2)$ to form the full causal and concurrent closure of all interactions coded in two orderings. Due to nondeterminism, the composition $\sigma_1 \times \sigma_2$ does not yield a single causality label but rather a set of them. They are obtained as the maximal irreflexive and transitive sub-orderings of the transitive closure $(\prec_1 \cup \prec_2)^+$. Here, the transitive closure of $\prec_1 \cup \prec_2$ is the smallest relation $\prec$ with $\prec_1 \cup \prec_2 \subseteq \prec$ such that, if $A \prec \{b\}$ and $b \in B \prec C$, then $(B \setminus \{b\}) \cup A \prec C$. Now, $(\ell, \prec) \in \sigma_1 \times \sigma_2$ if (i) $\ell = \ell_1 \cup \ell_2$, (ii) $(\ell, \prec)$ is a causality label, and (iii) $\prec$ is maximal in $(\prec_1 \cup \prec_2)^+$.

For example, we have $\mathit{lab}(t_1) \times \mathit{lab}(t_2) = \{\sigma_3\}$, where t_1, t_2 and σ_3 are as before, which confirms formally that σ_3 is the composition of t_1 and t_2. Note that Cond. (ii) implies that $\ell_1 \cup \ell_2$ must be consistent. Hence, $\mathit{lab}(\overline{a}/b) \times \mathit{lab}(b/a) = \emptyset$ which reflects the fact that both transitions can never be part of the same step due to global consistency. Cond. (iii) resolves cyclic dependencies: Consider actions $\mathit{lab}(a/b) = (\{a, b\}, \prec_1)$, $\mathit{lab}(b/c) = (\{b, c\}, \prec_2)$ and $\mathit{lab}(c/a) = (\{c, a\}, \prec_3$), which are consistent but their combined transitive closure $(\prec_1 \cup \prec_2 \cup \prec_3)^+$ has reflexive cycles $\{e\} \prec \{e\}$, for $e \in \{a, b, c\}$. The maximal irreflexive and transitive sub-orderings are given by $\sigma_4 =_{\mathrm{df}} (\ell, \{a\} \prec \{b\} \prec \{c\}, \{a\} \prec \{c\})$, $\sigma_5 =_{\mathrm{df}} (\ell, \{b\} \prec \{c\} \prec \{a\}, \{b\} \prec \{a\})$, $\sigma_6 =_{\mathrm{df}} (\ell, \{c\} \prec \{b\} \prec \{a\}, \{c\} \prec \{a\})$, where $\ell = \{a, b, c\}$. Then, $\mathit{lab}(a/b) \times \mathit{lab}(b/c) \times \mathit{lab}(c/a) = \{\sigma_4, \sigma_5, \sigma_6\}$ which describes the three ways in which transitions a/b, b/c and c/a can partake in the same step. They show that the environment needs to provide at least one of the triggers $\mathsf{trg}(\sigma_4) = \{a\}$, $\mathsf{trg}(\sigma_5) = \{b\}$ or $\mathsf{trg}(\sigma_6) = \{c\}$ to generate the combined action $\mathsf{act}(\sigma_4) = \{b, c\}$, $\mathsf{act}(\sigma_5) = \{c, a\}$ or $\mathsf{act}(\sigma_6) = \{a, b\}$, respectively.

We can now define the initial causality labels of a configuration C presented as a one-step reaction relation $C \mapsto \sigma$, for $\sigma \in \Sigma(\Pi)$, by induction on C:

- $0 \mapsto (\ell, \emptyset)$ for all binary $\ell \subseteq \Pi \cup \overline{\Pi}$;
- $t \mapsto \mathit{lab}(t)$, and $t \mapsto (\ell, \emptyset)$ for all binary $\ell \subseteq \Pi \cup \overline{\Pi}$ which disable $\mathit{lab}(t)$.
- $C_1 \mapsto \sigma_1$ and $C_2 \mapsto \sigma_2$ implies $C_1 \parallel C_2 \mapsto \sigma$ for all $\sigma \in \sigma_1 \times \sigma_2$.

Observe that transitions not only generate *active* steps $t \mapsto \mathit{lab}(t)$ but also *passive*, or *idle*, steps $t \mapsto (\ell, \emptyset)$ with $\overline{\mathsf{trg}(\mathit{lab}(t))} \cap \ell \neq \emptyset$ in which they are disabled. This resolves conflicting choices and introduces internal nondeterminism. For example, although $\mathit{lab}(\overline{a}/b) \times \mathit{lab}(\overline{b}/a) = \emptyset$, there are active and passive steps $\overline{a}/b \mapsto \mathit{lab}(\overline{a}/b)$ and $\overline{b}/a \mapsto (\{\overline{a}, b\}, \emptyset)$, respectively, which combine $\mathit{lab}(\overline{a}/b) \times (\{\overline{a}, b\}, \emptyset) = \{\mathit{lab}(\overline{a}/b)\}$. Symmetrically, there is a passive step $\overline{a}/b \mapsto (\{a, \overline{b}\}, \emptyset)$ and active step $\overline{b}/a \mapsto \mathit{lab}(\overline{b}/a)$ giving $(\{a, \overline{b}\}, \emptyset) \times \mathit{lab}(\overline{b}/a) = \{\mathit{lab}(\overline{b}/a)\}$.

The following theorem is a key result of Levi [36]:

Theorem 2 (Correctness & Completeness). *If C is a configuration and $A \subseteq \Pi$, then A is a Pnueli-Shalev step response of C if and only if there exists a causality label σ with $C \mapsto \sigma$ such that $\emptyset$ enables σ and $A = \mathsf{act}(\sigma)$.*

Levi defines the labelled transition system across all steps $C_i \overset{\sigma:\kappa}{\mapsto} C_{i+1}$ of a Statechart, compositionally in the full syntax including choice and hierarchy.

The additional flag $\kappa \in \{\overline{\varepsilon}, \varepsilon\}$ in Levi's label indicates if the step is idle or non-idle. These flags are needed for compositionality with respect to choice, which is not part of the syntax considered here. Without the flag both 0 and $a/\emptyset$, say, would have the same initial labels, viz. $(\{a\}, \emptyset)$ and $(\{\overline{a}\}, \emptyset)$, and thus be semantically indistinguishable. However, they induce different behaviour in the context $(\cdot + \emptyset/b) \parallel \emptyset/a$: The configuration $(0 + \emptyset/b) \parallel \emptyset/a$ must always produce events $\{a, b\}$, whereas in $(a/\emptyset + \emptyset/b) \parallel \emptyset/a$ the transition $\emptyset/b$ can be preempted by $a/\emptyset$ making a step on its own triggered by the parallel transition $\emptyset/a$ in the context. Hence, $(a/\emptyset + \emptyset/b) \parallel \emptyset/a$ not only has the response $\{a, b\}$ but also $\{a\}$. Levi's flags avoid the confusion between 0 and $a/\emptyset$ since the initial step $0 \mapsto (\{a\}, \emptyset) : \overline{\varepsilon}$ of the former is idle while the intial step $a/\emptyset \mapsto (\{a\}, \emptyset) : \varepsilon$ of the latter is non-idle.

Further, Levi presents a compositional μ-calculus verification system for these labelled transition systems [36]. However, no congruence and full-abstraction results are proven. In the work of Maggiolo-Schettini, Peron, Tini [43, 44], a similar order-theoretic refinement for Pnueli-Shalev semantics, as well as for the modified semantics mentioned in Sec. 2.2, is developed, together with congruence results for several behavioural preorders. It has been shown by Lüttgen, von der Beeck and Cleaveland [38] that the two levels of the order-theoretic semantics, i.e., configurations and causality labels $\Sigma(\Pi)$, can also be flattened into a single labelled transition system with first-order labels in which special clock transitions mark the beginning and end of a step.

3.3 Denotational Perspective

While Pnueli and Shalev's declarative step semantics corresponds to their operational step semantics, it is not denotational because it lacks compositionality as an interaction with the environment is only allowed at the beginning of a step but not during a step. The denotational perspective presented in this section does away with this shortcoming.

Interaction steps. The idea is to read a configuration C of a Statechart as a specification of a set of *interaction steps* between the Statechart and all its possible environments. This set is nonempty since one may always construct an environment that disables those transitions in C that would cause a global inconsistency and, thus, failure in the sense of Pnueli and Shalev. Formally, an interaction step is a monotonic sequence $M = (M_0, M_1, \ldots, M_n)$ of reactions $M_i \subseteq \Pi$, where $M_{i-1} \subsetneq M_i$ for all i. Each reaction contains events representing both environmental input and the Statechart's response. Intuitively, by the requirement for monotonicity, such a sequence extends the communication potential between the Statechart and its environment, until this potential is exhausted.

An interaction step is best understood as a separation of a Pnueli-Shalev step response M_n. Each M_i extends M_{i-1} by new environmental stimuli plus the Statechart's response to these. Here, responses are computed according to Pnueli and Shalev, except that events not contained in M_n are assumed to be absent in M_i. In this way, global consistency is interpreted as a logical specification of the full interaction step M and not only relative to a single reaction M_i. In other words, each interaction step separates a Pnueli-Shalev step response into causally closed sets of events. Each passage from M_{i-1} to M_i represents a non-causal "step" triggered by the environment. This creates a separation between M_{i-1} and M_i in the spirit of Pnueli and Shalev: as all events generated by the transitions enabled under M_{i-1} are contained in M_{i-1}, their intersection with $M_i \setminus M_{i-1}$ is empty.

Interpreting configurations, logically. Transitions $P, \overline{N}/A$ of the considered configuration C are interpreted on interaction steps $M = (M_0, M_1, \ldots, M_n)$ as follows. For each M_i, either (1) all events in A are also in M_i, or (2a) one or more events in A are not in M_i and $P \not\subseteq M_i$, or (2b) one or more events in A are not in M_i, and some event $e \in N$ is in M_j for some $i \leq j \leq n$. Intuitively, case (1) corresponds to the situation in which the transition is enabled and thus fires, or where the environment ensures that all events of the transition's action are provided. Cases (2a) and (2b) correspond to the situation where the transition is not enabled since not all positive trigger events are present (2a), or not all negative trigger events are absent because they are provided later in the sequence (2b). Case (2b) ensures that, as desired above, global consistency is enforced over the whole interaction step M.

Remarkably, this interpretation corresponds exactly to the one of intuitionistic logic [14] when reading negative events $\overline{e}$ as $\neg e$, and transition slashes "/" as logical implication. The composition of events in triggers or actions, as well as parallel composition "$\|$" on configurations, may simply be understood as conjunction, and our interaction steps M are nothing but linear Kripke structures. This correspondence with propositional intuitionistic logic over linear Kripke structures leads us to a general semantic relation $\models$, namely the logical satisfaction relation. Formally, an interaction step $M = (M_0, M_1, \ldots, M_n)$ *satisfies* configuration C, in signs $M \models C$, if $M, i \models C$ for all $0 \leq i \leq n$, where

$$\begin{array}{ll} M, i \models 0 & \textit{always} \\ M, i \models I/A & \textit{if } (I \cap \Pi \subseteq M_i \textit{ and } \overline{(I \cap \overline{\Pi})} \cap M_n = \emptyset) \textit{ implies } A \subseteq M_i \\ M, i \models C_1 \| C_2 & \textit{if } M, i \models C_1 \text{ and } M, i \models C_2 \,. \end{array}$$

If $M \models C$ we also say that M is an *(interaction) model* of C. The above definition is a shaved version of the standard semantics of propositional intuitionistic logic [14]. Configuration 0 is identified with *true* and, if $I = \emptyset$ for a transition I/A, the semantics of I/A reduces to $A \subseteq M_i$. Now we have $M \models C$ if and only if C is valid in the linear Kripke structure M. Note that for interaction steps of length one, the notions of interaction model and classical model coincide, and we simply write M_1 for (M_1).

Response models. The step responses of a configuration C in the sense of Pnueli and Shalev are now exactly those interaction models M of C of length one, called *response models*, that are *not* suffixes of interaction models $N{=}(N_0, \ldots, N_m, M)$ of C with $m \geq 0$. If such a singleton interaction model was suffix of a longer interaction model, then — according to the argumentation above — the reaction would be separable and hence not causal. Thus, we have the following theorem which is proved in [41]:

Theorem 3 (Correctness & Completeness). *If C is a configuration and $M \subseteq \Pi$, then M is a Pnueli-Shalev step response of C if and only if M is a response model of C.*

We illustrate our notion of response model by means of a few examples:

– Firstly, consider the configuration $\overline{a}/b$ which exhibits the Pnueli-Shalev step response $\{b\}$ for the empty environment. Indeed, $\{b\}$ is a response model, i.e., a model and not a suffix of a longer interaction model. The only possibility would be the interaction step $(\emptyset, \{b\})$, but this is not an interaction model since $(\emptyset, \{b\}), 0 \not\models \overline{a}/b$: by definition, we have to consider $\emptyset \subseteq \emptyset$ and $\{a\} \cap \{b\} = \emptyset$ implies $\{b\} \subseteq \emptyset$, and this implication is false because $b \notin \emptyset$.
– Secondly, configuration $C_2 =_{\mathrm{df}} \overline{a}/b \,\|\, b/a$ has no response model. Although $\{a, b\}$ is a classical model of C_2, it may be left-extended to the interaction model $(\emptyset, \{a, b\})$. Note in particular that $(\emptyset, \{a, b\}), 0 \models \overline{a}/b$: by definition, we have to consider $\emptyset \subseteq \emptyset$ and $\{a\} \cap \{a, b\} = \emptyset$ implies $\{b\} \subseteq \emptyset$, and this implication trivially holds. In other words, event a is absent at position 0 of the interaction step $(\emptyset, \{a, b\})$ since it is added later in the step, namely at position 1, and thus is *not* absent.
– Thirdly, consider configuration $C_3 =_{\mathrm{df}} a/b \,\|\, b/a$ with its Pnueli-Shalev step response $\emptyset$. It is easy to see that $\emptyset$ is trivially a response model. In contrast, the set $\{a, b\}$ — while being a classical model of C_3 — is not a response model since the suffix extension $(\emptyset, \{a, b\})$ is an interaction model of C_3.
– Fourthly, configuration $\overline{a}/b \,\|\, \overline{b}/a$ offers two response models, namely $\{a\}$ and $\{b\}$, which are exactly the configuration's Pnueli-Shalev step responses. As in the example regarding configuration C_2 above, neither response model can be left-extended to an interaction model of length greater than one.

Full abstraction. The interaction models of a configuration C encode all possible interactions of C with all its environments and nothing more. Firstly, any differences between the interaction models of C are differences in the interactions of C with its environments and thus can be observed. Secondly, any observable difference in the interaction of C with its environments should imply a difference in the interaction models, and this holds by the very construction of interaction models. Therefore, the above interaction step semantics provides the desired compositional and fully abstract semantics for Pnueli-Shalev steps:

Theorem 4 (Compositionality & Full Abstraction). *Let C_1, C_2 be configurations. Then, C_1 and C_2 have the same interaction models if and only if, for all configurations C_3, the parallel configurations $C_1 \| C_3$ and $C_2 \| C_3$ have the same Pnueli-Shalev step responses.*

The proof of this theorem can be found in [41], where interaction steps are called *sequence structures* and where interaction models are referred to as *sequence* *models*. Most notably, the proof shows that it is sufficient to consider interaction models of lengths 1 and 2 only. This leads to a strategy for implementation, e.g., via encoding such interaction models using *binary decision diagrams* [13]. Finally, it should be remarked that the denotational approach has been generalised from single-step configurations to a full Statecharts language in [39].

3.4 Algebraic Perspective

We now turn to characterising the Pnueli-Shalev step semantics, or more precisely the largest congruence contained in equality on step responses, in terms of axioms. These are derived from general axioms of propositional intuitionistic formulas over linear Kripke models. Thus, the algebraic characterisation presented here is closely related to the above denotational characterisation.

Table 1. Axiom system for the Pnueli-Shalev step semantics

(A1)	$C_1 \parallel C_2$	$=$	$C_2 \parallel C_1$	
(A2)	$(C_1 \parallel C_2) \parallel C_3$	$=$	$C_1 \parallel (C_2 \parallel C_3)$	
(A3)	$C \parallel C$	$=$	C	
(A4)	$C \parallel 0$	$=$	C	
(B1)	$P, I/P$	$=$	0	
(B2)	$I/A \parallel I/B$	$=$	$I/(A \cup B)$	
(B3)	I/A	$=$	$I/A \parallel I, J/A$	
(B4)	$I/A \parallel A, J/B$	$=$	$I/A \parallel A, J/B \parallel I, J/B$	
(B5)	$P, \overline{N}/A$	$=$	0	*if* $P \cap N \neq \emptyset$
(C1)	$P, \overline{N}/A$	$=$	$P, \overline{N}/A, B$	*if* $N \cap A \neq \emptyset$
(C2)	$P, \overline{N}/A$	$=$	$P, e, \overline{N}/A \parallel P, \overline{N}, \overline{e}/A$	*if* $N \cap A \neq \emptyset$
(C3)	$I, \overline{N}/B \parallel P, \overline{N}/A$	$=$	$\{I, \overline{N}, \overline{e}/B \,:\, e \in P\} \parallel P, \overline{N}/A,$	*if* $N \cap A \neq \emptyset$ *and* $P \neq \emptyset$

Our axioms system is displayed in Table 1, where $A, B, N, P \subseteq \Pi$, $I, J \subseteq \Pi \cup \overline{\Pi}$ and $e \in \Pi$, and where C, C_1, C_2, C_3 are configurations. Axioms (A1)–(A4) are fairly natural, and we thus concentrate on explaining the remaining, more interesting axioms. Axiom (B1) describes that, if the firing of a transition merely reproduces in its action some of the events required by its trigger, then we might just as well not fire the transition at all. As a special case, $I/\emptyset = 0$. Axiom (B2) encodes that two transitions with the same trigger will always fire together and produce the events in both of their actions. Axiom (B3) states that, by adding in parallel to a transition I/A a transition $I, J/A$ with the same action A but additional trigger events J, the behaviour remains unchanged. Logically speaking, "guarding" via a trigger is a *weakening* operation.

Axiom (B4) is a version of the *cut* rule known from logic and reflects the chain-reaction character of firing transitions. The left-hand side $I/A \parallel A, J/B$ represents a situation in which there is a transition $A, J/B$ that is waiting, among other preconditions J, for the events in A that will be produced when transition I/A fires. Hence, it is safe to add transition $I, J/B$ to the right-hand side. Axiom (B5) deals with inconsistencies in triggers. If an action A is guarded by a trigger $P, \overline{N}$ in which some event is required to be both present and absent, i.e., $P \cap N \neq \emptyset$, then this transition will never become enabled and is thus equivalent to 0.

The remaining Axioms (C1)–(C3) are concerned with conflicts between the trigger and action of a transition. They axiomatise the effect of transitions that produce a failure under certain trigger conditions. More precisely, these axioms involve a transition $P, \overline{N}/A$ with $N \cap A \neq \emptyset$, whose firing leads to a global inconsistency. Such a transition rejects the completion of all steps in which its trigger $P, \overline{N}$ is true. Thus, since $P, \overline{N}/A$ can never fire in a consistent way, the step construction cannot terminate in a situation in which trigger $P, \overline{N}$ holds true. In other words, whenever all events in P have become present, the step construction must continue until at least one event in N is present in order to inactivate the transition. If this does not happen, the step construction fails. Axioms (C1)–(C3) formalise three different consequences of this.

Axiom (C1) reflects the fact that, since $P, \overline{N}/A$ can never contribute to a completed step if $N \cap A \neq \emptyset$, we may add arbitrary other events B to its action, without changing its behaviour. Logically, this axiom corresponds to the laws $e \wedge \neg e \equiv \mathit{false}$ and $\mathit{false} \supset B \equiv \mathit{true}$, for any B. Axiom (C2) offers a second way of reading the inconsistency between triggers and actions. Since at completion time any event e is either present or absent, the same rejection that $P, \overline{N}/A$ produces can be achieved by $P, \overline{N}, e/A \,\|\, P, \overline{N}, \overline{e}/A$. This is because if e is present at completion time, then $P, \overline{N}, e/A$ raises the failure; if e is absent, then $P, \overline{N}, \overline{e}/A$ does the job. This is essentially the law $\neg e \wedge \neg\neg e \equiv \mathit{false}$ in logic. It is important to observe that the side condition $N \cap A \neq \emptyset$ is necessary: For example, $\emptyset/A$ is different from $e/A \,\|\, \overline{e}/A$ because in a parallel context A/e the latter fails (no step) while the former has the response $A \cup \{e\}$.

Finally, consider Axiom (C3). Instead of saying that $P, \overline{N}/A$ generates a failure if all events in P are present and all events in N are absent, we might say that, if all events in N are absent, then at least one of the events in P must be absent, provided the step under consideration is to be completed without failure. But then any parallel component of the form $I, \overline{N}/B$ can be replaced by the parallel composition $\|\{I, \overline{N}, \overline{e}/B \,:\, e \in P\}$. The reason is that, if $I, \overline{N}/B$ fires at all in the presence of transition $P, \overline{N}/A$, then at least one of the weaker transitions $I, \overline{N}, \overline{e}/C$ will be able to fire at some point, depending on which of the events in $P \neq \emptyset$ it is that will be absent to avoid failure. Again there is a logic equivalent for this, namely the law $\neg(p_1 \wedge p_2) \equiv \neg p_1 \vee \neg p_2$ that holds for *linear* Kripke structures. Last, but not least, it is important to note that configuration $P, \overline{N}/A$, for $N \cap A \neq \emptyset$, is not the same as configuration 0, since the former inevitably produces a failure if its trigger is true, while 0 does not respond at all.

Theorem 5 (Correctness & Completeness). *$C_1 = C_2$ can be derived from the axioms of Table 1 via standard equational reasoning if and only if, for all interaction steps M, $M \models C_1$ iff $M \models C_2$.*

A proof of this theorem be found in [40]. In that paper, a more general syntax for configurations has been employed in which transition actions may be arbitrary configurations. Last, but not least, it should be remarked that Axioms (B3) and (C1)–(C3) are unsound for Maggiolo-Schettini, Peron and Tini's variant of the Pnueli-Shalev step semantics [43].

3.5 Game-Theoretic Perspective

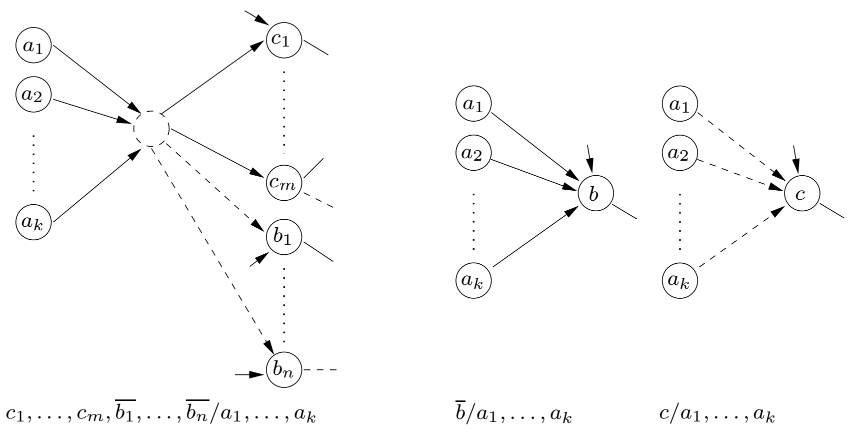
During the 1990s, a promising alternative to the traditional operational and denotational semantics of programming languages emerged. Game-theoretic models, which had long been used in descriptive set theory, economics and engineering control theory, were identified as a surprisingly powerful setting for dealing with system-environment interactions in a compositional fashion. For example, in the semantics of discrete reactive systems, games were applied to capture notions of refinement sensitive to input/output causality [3]. More specifically on the topic of this paper, it has been demonstrated that 2-player positional games provide a natural way of characterising different step semantics in synchronous programming. Game theory handles cyclic causal dependencies of non-monotonic behaviours by accounting for the system and environment dichotomy through the binary polarity of player and opponent. The swapping of roles gives constructive meaning to negation, and different forms of winning conditions generate different response semantics with varying degrees of constructiveness [2, 1].

In the following let us recall the main result from [1] as it applies to the Pnueli-Shalev semantics. To this end we first introduce the notion of a *maze* as the game equivalent of a configuration. A maze is a labelled transition system $M = (\mathsf{S}_\iota, \mathsf{S}_\tau, \xrightarrow{\iota}, \xrightarrow{\tau})$ consisting of disjoint sets of *visible rooms* S_ι and *secret rooms* S_τ, together with accessibility relations $\xrightarrow{\gamma} \subseteq \mathsf{S} \times \mathsf{S}$ between rooms $\mathsf{S} = \mathsf{S}_\iota \cup \mathsf{S}_\tau$ with two possible labels $\gamma \in \{\iota, \tau\}$. The transitions represent valid moves or *corridors*; a transition $m \xrightarrow{\iota} m'$ corresponds to a visible corridor connecting room m with m', whereas $m \xrightarrow{\tau} m'$ is a secret corridor. Designating a room or corridor as secret makes it unobservable, i.e., abstracts from it semantically.

A maze M acts as the game board on which two *players* A and B compete with each other to conquer rooms by taking alternate turns in moving along the corridors. When the play enters a room m in which player A receives the turn, then m becomes part of A's territory. If A now moves to some connected room m' through a visible corridor $m \xrightarrow{\iota} m'$, then A must hand over to B who then plays from m'. On the other hand, if A moves along a secret corridor $m \xrightarrow{\tau} m'$, then A keeps their turn and continues to play from m'. Room m may later be revisited in the play and, depending on who has the turn then, m may either fall to the other player B, or possession of m is perpetuated by A. We assume that the players use positional and consistent strategies. A *strategy* is a function that determines the next move of a player at every stage of a play in which they receive the turn. A strategy is *positional* if the decision only depends on the room from which the move is made, and not on the history of the play. This implies that every time a player receives the turn in a given room, they will take the same corridor out of it. A strategy is called *consistent* if all the positions ever occupied by a player are never lost to the opponent, and also if the player never enters a room left to the opponent. A consistent strategy keeps player A safely within a region $R_A \subseteq \mathsf{S}$, while at the same time it ensures that the opponent is confined to a region $R_B \subseteq \mathsf{S}$ from which they cannot escape.

In general, the objective of the game is that of defending regions (R_A, R_B), called *front lines*, according to a given *winning condition.* The winning condition that we are interested in here is *reactiveness.* A strategy is *reactive* if the player always eventually hands over to the opponent to make them appear in a visible room or get stuck in a secret room. We say that A *defends* front line (R_A, R_B) if A has a positional and consistent reactive strategy for all plays starting from R_A with A as the first player, and from R_B where B is the first to move. Reactive strategies permit infinite plays but require the player to be reactive in the sense that they are never embarrassed about a move when challenged and always generate a proper response (i.e., hand over to the opponent in a visible room) in finite time, though we do not insist that the player can stop the opponent from ever challenging again. In analogy with evaluation strategies in functional programming such defensible front lines are called *lazy* [1].

With every configuration C we associate a maze M_C such that the events Π correspond to the visible rooms S_ι and transitions $\mathsf{trans}(C)$ to secret rooms S_τ. The two sets (R_A, R_B) of a front line for M_C constitute a possible reactive response of C such that R_A and R_B will contain events that are present and absent, respectively. It turns out that the maximal lazy front lines of M_C are essentially the synchronous step responses of C as conceived by Pnueli and Shalev.


a_1
a_2
a_k
c_1
c_m
b_1
b_n
$c_1, \ldots, c_m, \overline{b_1}, \ldots, \overline{b_n}/a_1, \ldots, a_k$
a_1
a_2
a_k
b
$\overline{b}/a_1, \ldots, a_k$
a_1
a_2
a_k
c
$c/a_1, \ldots, a_k$
Fig. 3. Coding of transitions.

The maze M_C is obtained by observing that a transition $t = P, \overline{N}/A$ of C expresses the fact that action $a \in A$ is caused to be in R_A (i.e., present) if, for all $c \in P$, c is in R_A *and*, for all $\overline{b} \in \overline{N}$, b is in R_B. This conjunction can be modelled canonically by considering the transition t as an intermediate (secret) room and by adding (i) a visible corridor between each $a \in A$ and t; (ii) a visible corridor between t and each $c \in R_A$; and (iii) a secret corridor between t and each $b \in N$ as seen in Fig. 3 on the left. The graphical convention used here is that visible rooms/corridors are drawn with solid lines and secret rooms/corridors with dashed lines. When all transitions $t \in C$ have been represented in this way, they form a maze M_C of secret rooms connected through events Π as the visible rooms. Clearly, this translation is compositional where $C_1 \| C_2$ is the set-theoretic union of the mazes M_{C_1} and M_{C_2}. Some simplifications are possible, e.g., a transition like $c/a_1, \ldots, a_k$ with only one trigger may be coded without the intermediate room as a bundle of secret corridors from a_i to c. Similarly, a transition $\overline{b}/a_1, \ldots, a_k$ is simply a bunch of visible corridors from a_i to b. This is illustrated in Fig. 3 on the right.

Theorem 6 (Correctness & Completeness). *Let C be a configuration and M_C be the maze associated with C. Then, $A \subseteq \Pi$ is a Pnueli-Shalev step response of C if and only if there exists a lazy front line $(R_A, \mathsf{S} \setminus R_A)$ in M_C such that $A = R_A \cap \Pi$.*

The proof of this theorem can be found in [1]. Note how the game model accommodates both the failure and nondeterminism of step responses. Depending on M_C, it may happen that there is no strategy to avoid a (visible) room m being visited by both players infinitely often. This corresponds to Pnueli and Shalev's step-construction procedure returning a failure. Also, a room m may occur in two different lazy front lines, which yields nondeterministic behaviour.

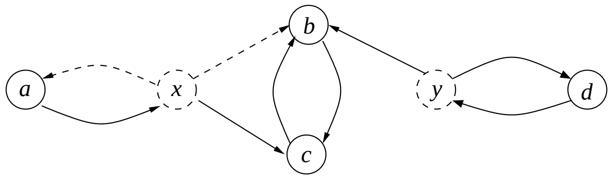


Fig. 4. The maze M_C for component $C = \overline{c}/b \parallel \overline{b}/c \parallel c, \overline{a}, \overline{b}/a \parallel b, d/d$ with maximal lazy front lines $(\{b, x, y\}, \{a, c, d\})$ and $(\{c, y\}, \{b, d\})$.

In general, the responses of a configuration C are the maximal lazy front lines in the maze M_C. Every *binary* front line, i.e., a front line (R_A, R_B) with $R_A \cup R_B = \mathsf{S}$, is trivially maximal. Yet, maximal lazy front lines need neither be uniquely defined nor two-valued. For the maze M_C in Fig. 4 we find that there are two maximal lazy front lines $(\{b, x, y\}, \{a, c, d\})$ and $(\{c, y\}, \{b, d\})$. Of those only the former is binary and thus a Pnueli-Shalev step response according to Thm. 6. Consider $R_A = \{b, x, y\}$ and $R_B = \{a, c, d\}$ first. From all rooms in R_A player A has a strategy to make the opponent take the turn in one of the visible rooms R_B. E.g., from x we move secretly to b keeping the turn and then continue visibly to c where the opponent must continue. Also, from R_B the opponent must immediately hand back to A in a room of R_A with the first move. This keeps A consistently in R_A and B in R_B and makes B always eventually take the turn in a visible room. Similarly, one shows that the front line $R_A = \{c, y\}$ and $R_B = \{b, d\}$ is defensible. This, too, is a maximal front line because none of the remaining rooms a or x can be defended consistently as part of R_A or of R_B. Indeed, the Pnueli-Shalev semantics eliminates this non-binary solution $(\{c, y\}, \{b, d\})$ by backtracking so that configuration C is deterministic.

4 Related Work, Esterel & Logic Programming

This section discusses the Pnueli-Shalev step semantics in the light of the rich volume of related work which was either triggered by or performed orthogonally to the research in Statecharts and its semantics. In particular, this section compares the Pnueli-Shalev step semantics to the constructive semantics of the synchronous language *Esterel* [7, 52], with the aim of highlighting the close semantic relationship between Statecharts and Esterel. We also relate the Pnueli-Shalev step semantics to the so-called stable models of *logic programming* [49], a field in which the interpretation of negation plays a prominent role, too.

4.1 Related Work

Defining a synchronous step response involves the incremental firing of transitions which may both trigger and inhibit each other via broadcasting events. The deterministic stabilisation of this micro-scheduling process is highly non-trivial in the presence of cyclic dependencies and negative trigger conditions. Pnueli and Shalev's approach is one of many conceivable ways of defining a consistent scheduling strategy. We set the scene here with a brief survey of work on synchronous step semantics based on how event absence is treated in the step construction. We do not consider semantics that evade the consistency problem by banning negation such as *Modecharts* [34] or *UML state machines* and their derivatives [55].

The first take on the problem was the view underlying the first formal Statecharts semantics [28]. It does not consider the constraint of global consistency so that the absence of an event remains a local, or a transient, condition which may be overridden within the same step.

The second take is to break causality cycles systematically, for which we can identify two strands. One option is to delay the broadcast of events into the next step, so as to avoid *instantaneous* broadcast. This is the approach adopted in Leveson et al's *RSML* [35] and the step semantics of *STATEMATE* [26]. The other option to break cycles by default may be subsumed under Boussinot's slogan "no instantaneous reaction to [event] absence," according to which a negative trigger event tests for absence in the *previous* step rather than the current one. This interpretation has gained some importance in the synchronous programming community. Examples are Boussinot's *Reactive-C* [9], Boussinot and de Simone's synchronous reactive calculus *SL* [12], Mandel and Pouzet's functional reactive programming language *ReactiveML* [45], and Boussinot and Dabrowski's *FunLoft* [11] which is a globally asynchronous, locally synchronous model of multi-threading. The idea is also applied in logic programming, specifically in Saraswat, Jagadeesan and Gupta's language *tcc* for timed concurrent constraint programming [53].

The third take permits both instantaneous reaction to absence and instantaneous event propagation under the constraint of *global consistency.* All these step semantics construct maximal causally-closed and consistent sets of transitions. There are surprisingly many strategies for doing this which have found their applications. One important split arises in the operational model of step construction from the question of who is responsible for event absence: the system or the environment. The system view of absence underlies the work of Maggiolo-Schettini, Peron and Tini [43] and of Lüttgen, von der Beeck and Cleaveland [37], as discussed above. Dual to this view is the environment view in which absence is defined externally and thus is not determined until the step is complete and closed off against the environment. This is logically the most tricky scenario as it involves constructive anticipation and forces one to deal with non-causal programs, i.e., potential failure and deadlock behaviour. A rather useful systematics for this class of semantics has been introduced by Boussinot in his *Sugarcubes* report [10]. Boussinot's classification is based on a *potential function* π which is used at stage i of the step construction to speculate about which events $\pi(i) \subseteq \pi$ may potentially be broadcast later. By complement, all other events $\Pi \setminus \pi(i)$ are deemed absent at stage i. If $\sigma(i)$ is the set of events that have been broadcast by stage i, then a transition t is triggered if $\mathsf{trg}(t) \cap \Pi \subseteq \sigma(i)$ and $\overline{\mathsf{trg}(t)} \cap \pi(i) = \emptyset$. If π is *correct* — i.e., it contains all events that are eventually broadcast: $\sigma(j) \subseteq \pi(i)$ for $i \leq j$ —, then one does not get an inconsistency failure and does not need to backtrack. Boussinot shows that every correct potential function leads to a deterministic but possibly deadlocking step [10].

The most prominent representatives in this category are Pnueli and Shalev's semantics [51], Philipps and Scholz' μCharts variation [50] of it, and Berry's constructive semantics for Esterel [52] discussed in detail below, which corresponds to a correct potential function. The Pnueli-Shalev step semantics is obtained for the trivial potential function $\pi(i) = \sigma(i)$, which permits full speculation so that all events not currently broadcast can be taken to trigger absences. Such π is not correct and, consequently, one has failure and nondeterminism. However, the scheduling cannot deadlock.

All approaches to global consistency reported so far are *operational*, in the sense that they can be implemented by some form of scheduling. This is different from the *logical* approach described in Sec. 3.3, which employs intuitionistic logic for interpreting negative events. Of course, we can also apply classical logic; a configuration C then induces Boolean equations over events which describe the necessary and sufficient conditions for each event to be present in a step of C. Each classical solution is called a *logically coherent* step. A program is *logically correct* if all its configurations have exactly one logically coherent step under every input stimulus. This is the *logical behavioural semantics* [52] that is applied in the visual language *Argos* [46], one of the early synchronous languages developed by Maraninchi around 1991. Argos is well-known for the invention of a fully semantical and thus compositional version of inter-level transitions. There are, of course, many other truth-value interpretations of configurations such as (a) the Kleene-style ternary interpretation [19] which is related to Esterel's constructive semantics discussed in Sec. 4.2, and (b) the various models of normal logic programming mentioned in Sec. 4.3.

4.2 Relation to Esterel

Esterel is a textual, imperative language for specifying the behaviour of reactive systems, which has been developed by Berry and colleagues since the early 1980s [7, 52], concurrently to and independently of Harel's Statecharts. A visual version of Esterel is André's *SyncCharts* [4] which is implemented as *Safe State Machines* in the embedded-software development tool *SCADE* [18]. Similar to Statecharts, Esterel provides primitives for decomposing reactions sequentially and concurrently, where concurrent reactions may involve a complex exchange of events. In Esterel terminology, one speaks of the *emission of signals* rather than the generation of events or the firing of transitions.

Like the semantics of Statecharts, the semantics of Esterel is designed around the concept of a step, called an *instant*, and it also supports the principles of synchrony and causality. Unlike the Pnueli-Shalev semantics of Statecharts, however, those Esterel programs for which the step construction does not complete, are rejected by the Esterel compiler. As a further distinction from Statecharts steps, Esterel instants are guaranteed to be deterministic. Esterel's semantics has significantly evolved over the years. In [52], Berry describes a much improved version that is founded on the idea of *constructiveness* and that encodes the principle of causality in a precise way, and not in an approximative way as earlier Esterel versions did. He also establishes the coincidence of three constructive styles of Esterel semantics — a *behavioural* semantics, an *operational* semantics, and a *circuit* semantics —, thereby testifying to the mathematical elegance and robustness of Esterel.

Table 2. The *Must* and *Cannot* functions for computing Esterel instants

$\mathit{Must}(0, S) =_{\mathrm{df}}$	$\emptyset$
$\mathit{Must}(I/A, S) =_{\mathrm{df}}$	$\begin{cases} A & \text{if } I \subseteq S \\ \emptyset & \text{otherwise} \end{cases}$
$\mathit{Must}(C_1 \| C_2, S) =_{\mathrm{df}}$	$\mathit{Must}(C_1, S) \cup \mathit{Must}(C_2, S)$
$\mathit{Cannot}(0, S) =_{\mathrm{df}}$	$\overline{\Pi}$
$\mathit{Cannot}(I/A, S) =_{\mathrm{df}}$	$\begin{cases} \overline{\Pi \setminus A} & \text{if } I \cap \overline{S} = \emptyset \\ \overline{\Pi} & \text{otherwise} \end{cases}$
$\mathit{Cannot}(C_1 \| C_2, S) =_{\mathrm{df}}$	$\mathit{Cannot}(C_1, S) \cap \mathit{Cannot}(C_2, S)$

The behavioural semantics of Esterel is declarative and based on computing the fixed point of a reaction function that is the analogue of Pnueli and Shalev's *enabled* function. As for Statecharts events, Esterel signals may be present or absent. While the presence of signals in Esterel is always derived from *emit* statements explicitly contained in the program text, the absence of a signal is inferred indirectly from the absence of emit statements. Esterel's reaction function collects all those signals e as being present that *must* be emitted under the assumption that certain signals are asserted by the system environment or emitted earlier within the instant. However, in addition and unlike Pnueli and Shalev's *enabled* function, Esterel's reaction function also records signals e that *cannot* be emitted as being absent. Hence, both the presence and the absence of signals must be shown constructively in Esterel; in contrast to the Pnueli-Shalev step semantics, the absence of a signal is not inferred by speculation.

To be more precise, we define the semantics of Esterel instants for our configuration syntax [52]. As indicated above, Esterel's reaction function operates on sets $S \subseteq \Pi \cup \overline{\Pi}$ coding explicit presence and absence statuses of signals. These are determined by two functions, $\mathit{Must}(C, \cdot)$ and $\mathit{Cannot}(C, \cdot)$, each of which takes a set of consistent signal statuses and returns a set of positive or negative signal statuses, respectively, for a given configuration C. The formal definition of both functions is displayed in Table 2. Here, a set $S \subseteq \Pi \cup \overline{\Pi}$ is called consistent, if S does not contain both $e, \overline{e}$ for any $e \in \Pi$. The Esterel reaction function $\mathit{esterel}(C, \cdot)$ for configuration C is now defined as $\mathit{esterel}(C, S) =_{\mathrm{df}} \mathit{Must}(C, S) \cup \mathit{Cannot}(C, S)$, which is monotonic in S and preserves consistency. The Esterel semantics of C is then the least fixed point of $\mathit{esterel}(C, \cdot)$.

As an example, consider the configuration $C =_{\mathrm{df}} \overline{a}/b \,||\, \overline{b}/a$. According to Esterel's semantics, the absence of neither signal a nor b can be inferred since either signal may potentially be emitted; formally, $\mathit{Must}(C, \emptyset) = \mathit{Cannot}(C, \emptyset) = \emptyset$ and indeed $\emptyset$ is the least fixed point of $\mathit{esterel}(C, \cdot)$. Hence, the Esterel compiler cannot determine the status of signals a and b and thus rejects C as not being causal. In contrast, Pnueli and Shalev's step-construction procedure may initially assume that a is absent, or alternatively that b is absent, and thus infer two possible steps: step $\{b\}$ in the former case and step $\{a\}$ in the latter case. As suggested by this example, it is the constructive treatment of negation in Esterel that ensures the determinism of Esterel instants. Technically, this constructiveness ensures that Esterel's reaction function $\mathit{esterel}(C, \cdot)$ is monotonic, which is not the case for Pnueli and Shalev's *enabled* function. Thus, the least fixed point of $\mathit{esterel}(C, \cdot)$ is guaranteed to exist. The following theorem, which is proved in [42], relates the *least* fixed point property of Esterel to inseparability in the sense of Pnueli and Shalev. Recall here that inseparability reflects causality, i.e., a separable set of signal statuses points to a break in the causality chain when emitting signals.

Theorem 7 (Inseparability in Esterel). *Let C be a configuration. Then, S is the least fixed point of $\mathit{esterel}(C, \cdot)$ if and only if S is a fixed point of $\mathit{esterel}(C, \cdot)$ and inseparable for C.*

Esterel programs describing an instant may also be given a denotational semantics in terms of response models similar to Sec. 3.3, since Esterel instants are constructive in the sense of intuitionistic logics. This can be achieved by reading the behavioural Esterel reaction function as a formula in intuitionistic logic over signal statuses. Details of this denotational, model-theoretic approach can be found in [42]. In a similar spirit can the game-theoretic approach to the Pnueli-Shalev step semantics presented in Sec. 3.5 be adapted to Esterel instants, as is shown in [2, 1]. Formally, it can be proved that every constructive Esterel instant of a configuration C is also a Pnueli-Shalev step response of C. Interestingly, the configuration of Fig. 4 that we have found to only have a single Pnueli-Shalev step response is non-constructive under Esterel's semantics. This means that Pnueli-Shalev steps are more liberal than Esterel instants even on deterministic behaviours.

4.3 Relation to Logic Programming

The simplest declarative view of Statecharts configurations is to consider each transition as a logical implication between atomic propositions stating the presence or absence of events within a synchronous step. For instance, $a, \overline{b}/c$ states that "*whenever a is present and b is absent then c is present.*" In logic syntax we would write $(a \wedge \neg b) \supset c$, as suggested in Sec. 3.3. In this way, a configuration C turns into a set of propositional Horn clauses with negative atoms, or a logic program in which all atoms are ground. While negation is not part of standard definite Horn clause programming, it is a central feature of *normal logic programming* (NLP) which permits negative literals in clause bodies and queries. It is thus natural to relate the Pnueli-Shalev step semantics of Statecharts with constructive interpretations of negation in logic programs.

Not surprisingly, NLP exhibits problems of compositionality and full-abstraction very similar to those that have hampered the development of Statecharts semantics. The gap between the declarative, model-theoretic semantics and the operational semantics is even bigger in NLP. Specifically, if the operational model of NLP is based on a strong sequential execution model, then the order in which clauses and literals are executed is constrained. The standard operational model of *negation-as-finite-failure* (NF) is based on *SLDNF resolution.* This is, essentially, a top-down, depth-first search in which all clauses and propositions are evaluated according to a deterministic rule selection strategy. For instance, under strict left-to-right selection, the program $a/a \parallel a, b/c \parallel \overline{c}/d$ loops for query $d?$. It needs to resolve atom $c?$ due to the third clause and then atom $a?$ as the first condition of $a, b/c$. In this process, however, the search gets caught in the looping clause a/a. On the other hand, if the first clause's body is commuted to $a/a \parallel b, a/c \parallel \overline{c}/d$, then the query $c?$ has finite failure, and $d?$ evaluates to true. Clearly, such intensional features of clause scheduling are difficult to capture by compositional model-theoretic or domain-theoretic techniques. Note that the step semantics of both Pnueli-Shalev and Esterel are better behaved, because of their implicit concurrent evaluation which makes trigger conjunction and parallel composition commutative. In both semantics, the program $a/a \parallel a, b/c \parallel \overline{c}/d$ generates a single step with a and b absent and d present.

Despite the problems with the standard operational SLDNF semantics, various types of declarative models based on three-valued and many-valued interpretations have been developed in the literature to approximate SLDNF for certain classes of NLP programs. We refer the reader to [54, 20] for a detailed survey of the results. It has been observed in [1] that Pnueli and Shalev's interpretation of steps coincides exactly with the so-called *stable models* introduced by Gelfond and Lifschitz [21]. Consider configuration C as a propositional logic program. Given a set of events $E \subseteq \Pi$, let C_E be the program in which (i) all transitions with negative triggers in E are removed, i.e., we drop from C all $P, \overline{N}/A$ with $N \cap E \neq \emptyset$; and (ii) all remaining transitions are relieved from any negative events, i.e., every $P, \overline{N}/A$ with $N \cap E = \emptyset$ is simplified to P/A. The pruned program C_E has no negations, and thus it has a unique minimal classical model M. A classical model of C_E is a set $M \subseteq \Pi$ making all transitions/clauses of C_E true, i.e., for all P/A from C_E for which $P \subseteq M$ we have $A \subseteq M$. A set $M \subseteq \Pi$ is called a *stable model* of C if M is the minimal classical model of C_M. It has been shown in [21, 49] that stable models yield a more general semantics which consistently interprets a wider class of NLP programs than SLDNF.

Theorem 8 (Correctness & Completeness). *$M \subseteq \Pi$ is a stable model of configuration C if and only if M is a Pnueli-Shalev step response of C.*

The proof of this theorem is straightforward via the denotational characterisation theorem (Thm. 3), together with the observation that $(M_0, M_1, \ldots, M_n) \models C$ in the sense of Sec. 3.3 if and only if all M_i are classical models of C_{M_n}, i.e., $M_i \models C_{M_n}$. In one direction suppose that M is the minimal classical model of C_M, i.e., $M \models C_M$ and thus $M \models C$. For every $M' \subsetneq M$ with $(M', M) \models C$ we would have $M' \models C_M$, thus contradicting that M was assumed to be minimal. Hence, M is a response model of C. Vice versa, suppose M is a response model of C. Then, $M \models C$ and thus $M \models C_M$. Further, for any other classical model $M' \subsetneq M$ of C_M, we would have $(M', M) \models C$. However, since M is a response model of C, this is impossible. This proves that M is a minimal model of C_M.

It is interesting to note that, while Pnueli and Shalev's notion of synchronous steps has not had much impact on synchronous programming tools, stable models have gained practical importance for NLP as the semantical underpinning of *answer set programming* [48]. From a wider perspective, therefore, it is fair to say that Pnueli-Shalev steps have indeed been implemented successfully in software engineering, albeit in a different domain. In addition, the theoretical results obtained around the Pnueli-Shalev semantics have ramifications in NLP. For instance, Thm. 4 of Sec. 3.3 implies that the standard intuitionistic semantics of logic provides a compositional and fully-abstract semantics for ground NLP programs under the stable interpretation.

5 Reminiscences on Amir Pnueli's First Contributions to the Semantics of Statecharts (by Willem de Roever)

The first time Amir Pnueli mentioned Statecharts to me was in 1984 during a summer school in La Colle sur Loup, North of Nice, in France. He also mentioned that he had invented the term "reactive systems" together with David Harel, during a joint air-plane flight, for the type of systems he was trying to characterise. I was immediately enthused by the concept of Statecharts: a clear pictorial specification that could be executed, with all the operators one needed, instead of using those cumbersome algebraic notations we were wrestling with! This was what we needed to make formal specification accessible to a much larger community of users, I thought.

When, through the Esprit funding programme of Basic Research of the European Community which was launched in 1985, the opportunity presented itself to cooperate with Amir on the semantics and proof theory of Statecharts within a European project (Descartes), we immediately grasped it and started visiting each other accompanied by our teams.

5.1 Visit to the Weizmann Institute in the Mid 1980s

I recall a visit to Amir in 1986 at the Weizmann Institute, accompanied by Rob Gerth, Cees Huizing, Ton Kalker and Ruurd Kuiper, in order to attend one of AdCad's first schools on Statecharts. We listened to the members of the new AdCad company which David Harel, Amir and Haggi and Ido Lachover had founded to commercially develop the STATEMATE system, enabling the execution of Statecharts and Activity Charts.

We had a great time! At the weekends we were taken on outings to Mitzpe Ramon, the remains of a large crater in the Negev, not far from Bersheva, and by Haggi Lachover to a cave where some remains of the Neanderthal man were discovered. And during the week we had these brilliant expositions of Statecharts and their semantics. For Amir and David had recognized very early that devising the "right" semantics for Statecharts would be a really challenging problem for us semanticists!

The discussions centered on ... 5 different semantics for Statecharts. Full Stop! This is amazing! We, computer scientists, are accustomed, indeed, to a range of semantics for programming languages and concepts, culminating in the "best" semantics, the so-called fully abstract one, which doesn't introduce any unobservable differences. But for Statecharts there seemed to exist widely different semantics, which, in a sense, contradicted each other. Of course we didn't truly believe this at first, and, helped by the probing minds of Rob Gerth and Cees Huizing [32], we obtained criteria on which to judge the semantics:

– **Responsiveness**, which guarantees an instantaneous response to a request for reaction, as dictated by Gérard Berry's synchrony hypothesis [7].
– **Modularity**, which consists of two properties: (1) The composition of two reactive systems is defined on the basis of their observable behaviours; there exist no additional inner details of the execution which can only be seen by the other system. (2) When an event is generated, it is broadcast all around the system and is immediately available to everyone.
– **Causality**, i.e., for every event that is generated, there is a causal chain of events that leads to that event.

To us, whether meeting at the Weizmann Institute with Amir and David or working at the EUT in Eindhoven, these were the three criteria which a reasonable semantics for Statecharts should satisfy. But somehow it turned out to be very difficult to meet these criteria simultaneously. And that explains why Amir introduced 5 different semantics for Statecharts.

5.2 The 5 Different Semantics

For instance, there was **semantics A** [26] adopted in the STATEMATE system, in which the events that are generated as a reaction to some input can only be sensed in the step following that input, i.e., semantics A was not responsive. Certainly we should be able to do better than that!

This led to **semantics B** [28] which was responsive, but required the introduction of the notion of *micro-steps*: every observable action, i.e., every *macro-step*, was divided into an arbitrary finite number of micro-steps. Of this one, Rob Gerth, Cees Huizing and I developed a fully-abstract version [33]. The problem with this semantics is that if you take micro-steps in a different order, one may get a different observable result. So, semantics B turned out to be too subtle and too nondeterministic to be of practical use.

This led to **semantics C** [51], also known as the *Pnueli-Shalev semantics*, which overcomes this problem by demanding *global consistency* of every micro-step. Relative to this semantics, Jozef Hooman, Ramesh and I [30] developed a sound and complete compositional Hoare logic for Statecharts, and Francesca Levi [36] a sound and complete compositional proof system for checking μ-calculus properties of Pnueli-Shalev Statecharts. However, semantics C does not fully solve the problem of *modularity*, i.e., the behaviour of a process cannot be explained in terms of macro-steps only.

This led to **semantics D** in which all events that are generated during some macro-step are considered as if they were present right from the start of the step, no matter at which particular micro-step they were generated. As a consequence, the macro-behaviour of a process suffices to describe its interactions with other processes. Early versions of the languages Esterel, Lustre and Argos follow this approach. The advantage of semantics C over D, however, is that the first respects causality: each reaction can be traced back to an input from the environment via a chain of reactions, each causing the next one. In semantics D, however, it is possible that reactions trigger themselves! I.e., there is a problem with *causality*.

And then there is **semantics E**, modeling the current implementation of Statecharts in STATEMATE, which is an "acceleration" of semantics A. Events are generated at the next step, but before the reaction of the system has completely died out no input from the environment is possible.

Fig. 5, taken from [32], shows how each version of the semantics is an attempt to improve upon the other one. The discovery of Rob Gerth and Cees Huizing in 1988 that no semantics for reactive systems can be responsive, modular and causal at the same time, contributed a lot to the clarification of our many discus-

$$\begin{tikzcd}
A \arrow[r, "\textit{repsonsiveness}"] \arrow[d, "\textit{responsiveness}"] & B \arrow[r] & C \arrow[r, "\textit{modularity}"] & D \arrow[l, bend left, "\textit{causality}"] \\
E \arrow[u, bend left, "\textit{modularity}"] & & &
\end{tikzcd}$$

Fig. 5. Overview of the relationships among semantics A–E [32].

sions with Amir on the semantics of Statecharts: *there exists no best semantics for them!* This helps us to explain the situation so aptly described by:

– Michael von der Beeck in [5], in which he lists more than 20 different semantics for Statecharts published by 1994;
– Andrea Maggiolo-Schettini, Adriano Peron and Simone Tini in [44], who employ some variants of Statecharts' step semantics using SOS semantics to study (pre-)congruence properties of their preorders and equivalences;
– Rick Eshuis in [16], who identifies a set of constraints ensuring that Pnueli-Shalev, STATEMATE and UML semantics coincide, if observations are restricted to linear, stuttering-closed and separable properties;
– Sharam Esmaeilsabzali, Nancy A. Day and Joanne M. Atlee in [17], who address the following two problems for Statecharts and for a large number of other languages that subscribe to the synchrony hypothesis: (1) When should one choose which semantic variant? (2) How can different semantic variants be compared, and on the basis of which criteria?

Thus one observes that, within a time span of 25 years, the focus of providing semantics for the concept of Statecharts and related languages has shifted from looking for one ideal "best" semantics to the realization that such a quest is hopeless. Instead, the insight has been gained that one either should look for a set of restrictions on the environment and one's observations such that these semantic differences disappear; or one accepts that different applications require different specification mechanisms, assisted by a catalogue of possible criteria one might want to be met.

References

[1] J. Aguado and M. Mendler. Constructive semantics for instantaneous reactions. *Theoretical Computer Science*, 2010. To appear. A preliminary version of this article is available as Tech. Rep. 63/2005, Univ. of Bamberg, 2005.
[2] J. Aguado, M. Mendler, and G. Lüttgen. A-maze-ing Esterel. In *SLAP '03*, vol. 88 of *ENTCS*, pp. 21–37, 2004.
[3] L. de Alfaro and T. A. Henzinger. Interface automata. In *ESEC/FSE '01*, pp. 109–120. ACM Press, 2001.
[4] C. André. Computing SyncCharts reactions. In *SLAP '03*, vol. 88 of *ENTCS*, pp. 3–19, 2004.
[5] M. von der Beeck. A comparison of Statecharts variants. In *FTRTFT '94*, vol. 863 of *LNCS*, pp. 128–148, 1994.

[6] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proc. of the IEEE*, 91(1):64–83, 2003.

[7] G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[8] T. Bienmüller, W. Damm, and H. Wittke. The STATEMATE verification environment – Making it real. In *CAV 2000*, vol. 1855 of *LNCS*, pp. 561–567, 2000.

[9] F. Boussinot. Reactive C: An extension of C to program reactive systems. *Software – Practice and Experience*, 21(4):401–428, 1991.

[10] F. Boussinot. SugarCubes implementation of causality. Tech. rep. RR-3487, INRIA, 1998.

[11] F. Boussinot and F. Dabrowski. Safe reactive programming: The FunLoft proposal. In *MULTIPROG: Programmability Issues for Multi-Core Computers*. Informal workshop proceedings, 2008.

[12] F. Boussinot and R. de Simone. The SL synchronous language. *IEEE Trans. on Software Engineering*, 22(4):256–266, 1996.

[13] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[14] D. van Dalen. Intuitionistic logic. In *Handbook of Philosophical Logic*, vol. III, chap. 4, pp. 225–339. Reidel, 1986.

[15] W. Damm, B. Josko, H. Hungar, and A. Pnueli. A compositional real-time semantics of STATEMATE designs. In *Compositionality: The Significant Difference*, vol. 1536 of *LNCS*, pp. 186–238, 1998.

[16] R. Eshuis. Reconciling Statechart semantics. *Science of Computer Programming*, 74(3):65–99, 2009.

[17] S. Esmaeilsabzali, N. A. Day, and J. M. Atlee. Big-step semantics. Tech. rep. CS-2009-05, Univ. of Waterloo, 2009.

[18] Esterel Technologies. SCADE Suite. www.esterel-technologies.com.

[19] M. Fitting. A Kripke-Kleene semantics for logic programs. *Logic Programming*, 2(4):295–312, 1985.

[20] M. C. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11(2):91–116, 1991.

[21] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pp. 1070–1080, 1988.

[22] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[23] D. Harel. Statecharts in the making: A personal account. In *History of Programming Languages*, pp. 5/1–5/43. ACM Press, 2007.

[24] D. Harel and E. Gery. Executable object modeling with Statecharts. *IEEE Computer*, pp. 31–42, July 1997.

[25] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtul-Trauring, and M. Trakhtenbrot. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Trans. on Software Engineering*, 16(4):403–414, 1990.

[26] D. Harel and A. Naamad. The STATEMATE semantics of Statecharts. *ACM Trans. on Software Engineering Methodology*, 5(4):293–333, 1996.

[27] D. Harel and A. Pnueli. On the development of reactive systems. In *Logics and Models of Concurrent Systems*, vol. F-13 of *NATO ASI Series*, pp. 477–498. Springer, 1985.

[28] D. Harel, A. Pnueli, J. P. Schmidt, and R. Sherman. On the formal semantics of Statecharts. In *LICS '87*, pp. 54–64. IEEE Computer Society Press, 1987.

[29] D. Harel and M. Politi. *Modeling Reactive Systems with Statecharts: The STATEMATE Approach*. McGraw Hill, 1998.

[30] J. J. M. Hooman, S. Ramesh, and W.-P. de Roever. A compositional axiomatization of Statecharts. *Theoretical Computer Science*, 101:289–335, 1992.

[31] C. Huizing and W.-P. de Roever. Introduction to design choices in the semantics of Statecharts. *Information Processing Letters*, 37:205–213, 1991.

[32] C. Huizing and R. Gerth. Semantics of reactive systems in abstract time. In *REX Workshop '91*, vol. 600 of *LNCS*, pp. 291–314, 1992.

[33] C. Huizing, R. Gerth, and W.-P. de Roever. Modeling Statecharts behavior in a fully abstract way. In *CAAP '88*, vol. 299 of *LNCS*, pp. 271–294, 1988.

[34] F. Jahanian and A. K. Mok. Modechart: A specification language for real-time systems. *IEEE Trans. on Software Engineering*, 20(12):933–947, 1994.

[35] N. G. Leveson, M. Heimdahl, H. Hildreth, and J. D. Reese. Requirements specification for process-control systems. *IEEE Trans. on Software Engineering*, 20(9):684–707, 1994.

[36] F. Levi. A compositional μ-calculus proof system for Statecharts processes. *Theoretical Computer Science*, 216(1-2):271–311, 1999.

[37] G. Lüttgen, M. von der Beeck, and R. Cleaveland. Statecharts via process algebra. In *CONCUR '99*, vol. 1664 of *LNCS*, pp. 399–414, 1999.

[38] G. Lüttgen, M. von der Beeck, and R. Cleaveland. A compositional approach to Statecharts semantics. In *FSE 2000*, ACM Software Engineering Notes, pp. 120–129, 2000.

[39] G. Lüttgen and M. Mendler. Statecharts: From visual syntax to model-theoretic semantics. In *Integrating Diagrammatic and Formal Specification Techniques*, pp. 615–621. Austrian Computer Society, 2001.

[40] G. Lüttgen and M. Mendler. Axiomatizing an algebra of step reactions for synchronous languages. In *CONCUR '02*, vol. 2421 of *LNCS*, pp. 163–174, 2002.

[41] G. Lüttgen and M. Mendler. The intuitionism behind Statecharts steps. *ACM Trans. on Computational Logic*, 3(1):1–41, 2002.

[42] G. Lüttgen and M. Mendler. Towards a model-theory for Esterel. In *SLAP '02*, vol. 65:5 of *ENTCS*, 2002.

[43] A. Maggiolo-Schettini, A. Peron, and S. Tini. Equivalences of Statecharts. In *CONCUR '96*, vol. 1119 of *LNCS*, pp. 687–702, 1996.

[44] A. Maggiolo-Schettini, A. Peron, and S. Tini. A comparison of Statecharts step semantics. *Theoretical Computer Science*, 290(1):465–498, 2003.

[45] L. Mandel and M. Pouzet. ReactiveML: A reactive extension to ML. In *PPDP '05*. ACM Press, 2005.

[46] F. Maraninchi and Y. Rémond. Argos: An automaton-based synchronous language. *Comput. Lang.*, 27(1/3):61–92, 2001.

[47] The Mathworks. Stateflow user's guide. www.mathworks.com.

[48] I. Niemelä, P. Simons, and T. Syrjänen. Smodels: A system for answer set programming. In *Workshop on Non-Monotonic Reasoning*, Breckenridge, Colorado, USA, April 2000.

[49] D. Pearce. From here to there: Stable negation in logic programming. In D. M. Gabbay and H. Wansig, editors, *What is Negation?*, pp. 161–181. Kluwer, 1999.

[50] J. Phillips and P. Scholz. Compositional specification of embedded systems with Statecharts. In *TAPSOFT '97*, vol. 1214 of *LNCS*, pp. 637–651, 1997.

[51] A. Pnueli and M. Shalev. What is in a step: On the semantics of Statecharts. In *TACS '91*, vol. 526 of *LNCS*, pp. 244–264, 1991.

[52] D. Potop-Butucaru, S. A. Edwards, and G. Berry. *Compiling ESTEREL*. Springer, 2007.

[53] V. A. Saraswat, R. Jagadeesan, and V. Gupta. Foundations of timed concurrent constraint programming. In *LICS '94*, pp. 71–80. IEEE Computer Society Press, 1994.

[54] J. C. Shepherdson. Logics for negation as failure. In *Logic from Computer Science*, pp. 521–583. Springer, 1991.

[55] A. Taleghani and J. M. Atlee. Semantic variations among UML State Machines. In *MoDELS '06*, vol. 4199 of *LNCS*, pp. 245–259, 2006.

[56] A. C. Uselton and S. A. Smolka. A compositional semantics for Statecharts using labeled transition systems. In *CONCUR '94*, vol. 836 of *LNCS*, pp. 2–17, 1994.