What is in a Statechart: On the Semantics of Steps*

Andrew C. Uselton
Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794-4400, USA†

November 10, 1994

1 Introduction

Statecharts [Har87] is a highly structured and economical description language for complex, reactive systems [Pnu86, HP85], such as communication protocols and digital control units. Statecharts extend conventional state transition diagrams with three elements dealing with the notions of *hierarchy*, *concurrency* and *communication*. Hierarchy is achieved by embedding one statechart in the state of another statechart. Concurrency is supported by a (mainly) synchronous operation of statechart composition permitting a broadcast-style of communication among the constituent statecharts. The graphical syntax of statecharts is appealing – especially in its use of hierarchy; however, there are four problems with statecharts: its (linear) syntax should be, but isn't, inductively defined; the semantics should be, but isn't, compositional; the semantics should be inductively defined over the syntax; and the semantics for the synchronization mechanism should be isolated from the rest of the language.

It is important to have a *formal semantics* for statecharts so that their behavior can be precisely and unambiguously understood, and safety-critical properties of systems can be formally verified. As pointed out in [HGdR89, Per93] this is no easy task. Statecharts is a rich visual language with a complex synchronization mechanism.

A formal semantics solving the above four problems is difficult, but not impossible. The solution proceeds in four stages, each building upon the last: Inductively define a statecharts algebra, SA; modify the semantics of statecharts to make it compositional; define the semantics inductively over the terms of SA; and define a process algebra with a Plotkin-style structural operational semantics (SOS) in which the synchronization mechanism is isolated in the merge operator. The first two stages are presented in this paper. In the conclusion we summarize our strategy for the remaining two stages.

*This paper is presented as a preliminary dissertation report, and was prepared under the supervision of S. A. Smolka.

†E-mail: uselton@cs.sunysb.edu

Figure 1: An example statechart S and its step semantics $\Psi(S)$

Figure 1(a), which depicts an example statechart S, demonstrates the main features of statecharts. The most prominent ingredients of a statechart are *states*, drawn as rectangles, and *transitions*, drawn as arrows. Each state has a *name*, and states may contain or be *refined* by other states. Refinement in statecharts may be viewed as embedding one finite-state automaton (FSA) in the state of another, and FSAs can be composed in parallel (indicated by a dotted line separating them). Each FSA has an initial or *default* state, indicated by the presence of a small incoming arrow. Thus, in our example, state S is refined by states A and D. State A is in turn refined by states B and C, which are in parallel.

A global state or *configuration* of a statechart is given by the set of states that the statechart occupies in a given instant. For example, statechart S might be in states E and G simultaneously, since they are composed in parallel. When a statechart is "in" a state it is also "in" all the states that enclose that state, so the configuration for the above case would include all the states {S, A, B, C, E, G}, though {E, G} is an unambiguous abbreviation for the configuration.

Each transition has a *label* consisting of two parts, a *trigger* and a *generated action*. The trigger is a set consisting of *primitive* and *negative events*, while the generated action is a set of primitive events. In the case of the transition from state E to state F in Figure 1(a) the trigger is the primitive event a and the generated action is b.

In the original semantics for statecharts [HPSS87], a statechart *reacts* to some *input* by taking a *step* from one configuration to another and producing some *output*. A step is a collection of transitions that are *triggered* by the input. For a transition to be triggered by the input, its source state must be in the current configuration, every primitive event in its trigger must be in the input, and the negative events in its trigger must not appear in the input.

Transitions in parallel components of a statechart are synchronized via broadcast. For instance, in configuration {E, G}, if the statechart receives input a it reacts by taking the transition from E to F, generating (and broadcasting) action b, which triggers the transition from G to H, generating action c. All of this takes place in a single step.

As pointed out in [Mar92] a reactive system must have a defined response for any combination of its inputs, even if it is a null response (but see the footnote on Page 4.1). In configuration $\{D\}$ of Figure 1(a) an input of a would produce the response that the statechart remains in configuration $\{D\}$. On the other hand, in configuration $\{E, G\}$ there is a step to $\{F, H\}$ on inputs ab, ac, or abc,

Figure 2: $=_{\mathrm{SC}}$ fails to be a congruence

as well as on input a. Finally, in configuration $\{E, G\}$ on input *a off* there are two available steps: one to configuration $\{F, H\}$ and one to configuration $\{D\}$.

Pnueli and Shalev [PS91] refine the semantics of [HPSS87] by specifying three requirements to which a semantics for statecharts must conform: The **synchrony hypothesis** (SH) stipulates that the response of a statechart to a set of input events is *maximal*. **Causality** (C) is the requirement that two labels a/b and b/a may not be construed as causing each other when neither a nor b is in the input. **Global consistency** (GC) is the requirement that testing for the absence of an event not lead to a contradiction. For instance two transitions labeled $\overline{a}/b$ and b/a may not synchronize.

We refer to the semantics of [PS91] as the *step semantics* of statecharts and characterize it as a mapping Ψ from statecharts to *labeled transition systems* (LTSs). The LTS $\Psi(S)$ in Figure 1(b) has a state for each configuration of S, an initial state, an alphabet of labels, and a transition relation. For legibility the informal depiction of $\Psi(S)$ in Figure 1(b) omits the *implicit* steps that correspond to input containing more events than necessary to trigger the relevant step(s). For example, rather than a single step from $\{E, G\}$ to $\{F, H\}$ labeled a/bc, the LTS $\Psi(S)$ has sixteen such steps, one for each superset of $\{a\}$ in the set of events $\{a, b, c, on, off\}$.

The existing linear syntax of statecharts is defined in terms of a four-tuple that superficially resembles the definition of an FSA. Therefore there is no real inductively defined syntax of statecharts terms and hence no explicit notion of statecharts composition. Furthermore, the semantics is not compositional, and Figure 2 illustrates why this is so. Figures 2(a) and (b) depict two simple statecharts A and A'. The LTS in Figure 2(c) gives the step semantics for both A and A' (again with implicit steps omitted). Note that the negative events in A' ensure, for example, that there is no step from B' to C' on input ac.

Now define statechart equivalence $=_{\mathrm{SC}}$ by: $\Psi(A) \equiv \Psi(A') \Rightarrow A =_{\mathrm{SC}} A'$, where $\equiv$ denotes LTS isomorphism. Figure 2(d) shows a *statechart context* $X[\cdot]$ with a "place holder" in it. The blank

area is to be filled in with another statechart. If statechart A fills the blank in $X[\cdot]$ then the LTS $\Psi(X[A])$ is the one depicted in Figure 2(e). Finally, Figure 2(f) shows the LTS that results when the statechart A' is used in $X[\cdot]$. Clearly $\Psi(X[A])$ and $\Psi(X[A'])$ are not isomorphic – indeed, they are not even trace equivalent.

Thus this intuitive notion of statecharts equivalence is not a congruence and the step semantics is not compositional. Moreover, this will be the case for any reasonable choice of statecharts composition and equivalence.

In order to define a compositional semantics for statecharts two things are needed. First, there must be some way of composing statecharts into larger statecharts so that one may formalize the definition of a statecharts context. Second, some alternative semantic function Ψ' is needed such that $\Psi'(A) \not\equiv' \Psi'(A')$ in Figure 2. More generally we need the property that for all statecharts A and A': $A =_{\mathrm{SC}'} A' \Rightarrow \forall X[\cdot].\ X[A] =_{\mathrm{SC}'} X[A']$, where $=_{\mathrm{SC}'}$ is an equivalence based on Ψ'.

In this paper we present such a Ψ', which we call $\Psi_>$, by modifying the semantic alphabet of the labeled transition systems to which statecharts are mapped. The new semantic alphabet uses objects called *words*. A word w consists of a set of events together with a transitive ordering relation $>_w$ defined on the set of events. The function ψ constructs a word from an LTS label as follows. Let $l = \tau/acts$ be a semantic label. Then $\psi(l) = (\tau \cup acts, >_{\psi(l)})$ where $a >_{\psi(l)} b$ if there is some syntactic label with a in its trigger and b in its action. Thus the word $(w, >_w)$ for the semantic step in Figure 2 from $\{D, F\}$ to $\{E, G\}$ has $a >_w b$ and $c >_w d$. On the other hand the word $(w', >_{w'})$ for the step from $\{B'\}$ to $\{E'\}$ has $a >_{w'} b$, $a >_{w'} d$, $c >_{w'} b$, and $c >_{w'} d$. The result is that $\Psi_>(A) \not\equiv_> \Psi_>(A')$. The equivalence $\simeq$ over SA terms is defined by $\Psi_>(p) \equiv_> \Psi_>(p') \Rightarrow p \simeq p'$, where $\equiv_>$ is an equivalence over $\mathrm{LTS}_>$ that weakens isomorphism slightly as follows: rather than set equality between two words $(w, >_w)$ and $(w', >_{w'})$, we use a notion of *matching*, written $(w, >_w) \approx (w', >_{w'})$ such that $w = w'$, but $>_w$ and $>_{w'}$ can differ slightly (c.f. Section 5). This weakening of $\mathrm{LTS}_>$ isomorphism has the pleasant property of distinguishing, via $\Psi_>$, between two statecharts terms only when there is a distinguishing context, $X[\cdot]$.

Our main results can be summarized as follows:

- We introduce an inductively defined algebra SA of *statecharts terms* in which statechart contexts can be defined.
- We show that the algebra SA of statecharts terms is in one-to-one correspondence with the set-theoretic syntax of [PS91], modulo some simple syntactic identities.
- We show that $\simeq$ is a congruence for the operators of SA: For all SA terms p and p': $p \simeq p' \Rightarrow \forall X[\cdot].\ X[p] \simeq X[p']$. Thus our semantics is compositional.
- We show that our new semantics agrees with that of [PS91] in the following sense. $\simeq$ is the *largest* congruence contained in $=_{\mathrm{SC}}$, so it only distinguishes statecharts when necessary.

Related Work

[HGdR89] presents a denotational semantics for the *unvollendetes* variant of statecharts that has *time-out* events and *not-yet* events. The semantics of not-yet events is an alternative treatment of negative events consistent with [HPSS87] but not [PS91] in that the treatment of not-yet events

does not support the global consistency requirement (GC). Unvollendetes (or "incompletes") are the terms of an alternative algebra in which a statechart is assembled by connecting the incomplete transitions to and from the various states of the statechart. The semantic domain is the set of possible *histories* of computations. The semantics is compositional and fully abstract. [HRdR92] modifies the semantics of unvollendetes to satisfy global consistency and presents a compositional axiomatization.

In [Mar91, Mar92] Maraninchi presents a related graphical language *Argos* and gives a compositional process-algebraic semantics related to the language *Esterel*, but different from that of [PS91] in several respects. The syntax of Argos is designed to prevent non-determinism and circularity in the semantics.

In [Per93] a compositional denotational semantics based on *event structures* is presented for the negation-free variant of statecharts. Peron presents an axiomatization for unvollendetes based on the semantics.

The remainder of this paper proceeds as follows. Section 2 introduces SPS, the statecharts syntax of [PS91]. Section 3 introduces the algebra SA of statecharts terms. Section 3 ends by showing that the two syntaxes are interchangeable. Section 4 presents the step semantics of [PS91] and the semantic mapping $\Psi : \mathrm{SA} \to \mathbf{LTS}$, and demonstrates that this semantics is not compositional. Section 5 defines the modified semantic mapping $\Psi_>$ that preserves more syntactic information. Section 5 concludes by showing that with the modified semantic mapping, our notion of semantic equality, $\simeq$, is the largest $=_{\mathrm{SC}}$ respecting congruence. Section 6 concludes with a proposal for extending these results in fulfillment of a PhD dissertation.

2 Statecharts Syntax

In [PS91], a statechart is defined as a four-tuple $(\mathrm{E}, N, \mathcal{A}, r)$, where E is a finite set of *primitive events*, N is a finite set of *names* of *boxes*, $\mathcal{A}$ is a finite set of *arrows*, and $r \in N$ is the *root*.[1]

In a statechart-tuple $(\mathrm{E}, N, \mathcal{A}, r)$ the set $\mathrm{E} \subset \mathcal{E}$ gives the primitive event names that appear in the statechart, where $\mathcal{E}$ is the universal set of primitive events. For each element $\pi \in \mathrm{E}$ there is a *negative event* $\overline{\pi}$, and we treat negation as an operator to be applied to events or point-wise to sets of events such that $\overline{\mathrm{E}} = \{\overline{\pi} \mid \pi \in \mathrm{E}\}$. We also use $\mathbf{E}$ for $\mathrm{E} \cup \overline{\mathrm{E}}$. A *label* l has a *trigger* and an *action*. The trigger of a label is a subset of $\mathbf{E}$ and the action is a subset of E. A label l with trigger t and action a is written t/a, and we use the functions $\mathit{trigger}(l) = t$ and $\mathit{action}(l) = a$. The set $\mathit{primitives}(l) \subset \mathcal{E}$ gives all the primitive events π occurring in $\mathit{trigger}(l) \cup \mathit{action}(l)$ as well as those for which $\overline{\pi} \in \mathit{trigger}(l)$. The set $\mathit{events}(l) \subset \mathcal{E} \cup \overline{\mathcal{E}}$ gives all the events (primitive or negative) in l, i.e. $\mathit{events}(l) = \mathit{trigger}(l) \cup \mathit{action}(l)$. Traditionally, sets t and a are denoted in a statechart by listing their elements. For example, the label $\{a, \overline{b}\}/\{c, d\}$ is written $a\overline{b}/cd$. The set of labels generated by events E is $\mathcal{L}(\mathrm{E}) = 2^{\mathbf{E}} \times 2^{\mathrm{E}}$.

For a statechart $(\mathrm{E}, N, \mathcal{A}, r)$ the set of names N is equipped with the functions $\mathit{children}_N : N \to$

[1] In [PS91] the notation $(\Pi, S, \mathcal{T}, r, V)$ was used, and $\mathcal{S}$ was called the set of *states* and $\mathcal{T}$ was called the set of *transitions*. We prefer the usage *boxes* and *arrows* respectively and reserve the terms state and transition for *semantic* concepts. Similarly, the symbol Π for the set of events conflicts with our multiple product operator. We drop the value store V, the last element of the tuple, since it is assumed to be empty in their restricted syntax [PS91]. Addressing value stores is a subject for future work.

2^N, and $type_N : N \to \{and, or\}$, and the partial function $default_N : N \to N$. Thus N should be understood to be the structure $(N, children_N, type_N, default_N)$. When the context is clear we drop the subscript N on these functions. Let $\searrow \subset N \times N$ be the "child of" relation defined by $n \searrow n'$ if $n' \in children(n)$; then $\searrow+$ and $\searrow*$ are the transitive and reflexive, transitive closures of $\searrow$, respectively.

The names in N are organized in a tree-like structure. By this we mean that $\forall n \in N.\ r \searrow* n$ (r is the root), and if $n_1 \searrow n$ and $n_2 \searrow n$ then $n_1 = n_2$ (each name has a unique "parent"). If $type(n) = and$ then the children of n are combined in parallel and we call n an *and*-box. If $type(n) = or$ then the children of n are combined sequentially as in a traditional FSA, and we call n an *or*-box. By convention an *and*-box never has a child that is also an *and*-box. For every *or*-box n with $children(n) \neq \emptyset$, $default(n) \in children(n)$ gives the "initial state" of n's FSA.

In a statechart $(\mathrm{E}, N, \mathcal{A}, r)$ the set $\mathcal{A}$ of arrows is a finite subset of $N \times \mathcal{L}(\mathrm{E}) \times N$. An arrow $\alpha \in \mathcal{A}$ has a *source*, a *label*, and a *target*. For an arrow $\alpha = (n, l, n')$, $source(\alpha) = n$, $target(\alpha) = n'$, and $label(\alpha) = l$. Furthermore, $trigger(\alpha) = trigger(label(\alpha))$, $action(\alpha) = action(label(\alpha))$, and n and n' must be siblings in an *or*-box.[2] That is, there is some n'' with $type(n'') = or$ and $n, n' \in children(n'')$. Finally,

$$\mathrm{E} = \bigcup_{\alpha \in \mathcal{A}} primitives(label(l))$$

The following table gives the tuple elements $(\mathrm{E}, N, \mathcal{A}, r)$ for the statechart depicted in Figure 1:

E	$\{a, b, c, on, off\}$									
N		S	A	B	C	D	E	F	G	H
	children	$\{A, D\}$	$\{B, C\}$	$\{E, F\}$	$\{G, H\}$					
	type	*or*	*and*	*or*	*or*	*or*	*or*	*or*	*or*	*or*
	default	A		E	G					
$\mathcal{A}$	$\{(E, a/b, F), (F, a, E), (G, b/c, H), (H, b, G), (D, on, A), (A, off, D)\}$									
r	S									

We refer to the tuples defined above as SPS (for Syntax of Pnueli and Shalev). One further point about SPS should be mentioned. The actual choice of the names N in an SPS tuple is irrelevant. For this reason we present the following definition of $\equiv_{\mathrm{SPS}}$ – syntactic equality for SPS tuples.

Two SPS tuples are syntactically equal, written $(\mathrm{E}_1, N_1, \mathcal{A}_1, r_1) \equiv_{\mathrm{SPS}} (\mathrm{E}_2, N_2, \mathcal{A}_2, r_2)$, if there is a bijection $h : N_1 \to N_2$ enjoying the properties in Table 1[3]. Henceforth we refer to SPS/ $\equiv_{\mathrm{SPS}}$ as the set SC or simply as statecharts.

[2] In [PS91] the source and the target of an arrow are *sets* of names in $children^+(n)$ for some n. All of the results of this paper are valid for statecharts with this generalized *multi-link* variety of arrow. Presenting the syntax and semantics of multi-link arrows is beyond the scope of this report, since the mechanics of such arrows is complex and would not add any new insights.

[3] Note that $\mathrm{E}_1 = \mathrm{E}_2$ is a consequence of the properties of Table 1.

$$h(r_1) = r_2$$

For all $n, n' \in N_1$ and $l \in \mathcal{L}(\mathrm{E}_1)$ we have:

$$(n, l, n') \in \mathcal{A}_1 \quad \text{if and only if} \quad (h(n), l, h(n')) \in \mathcal{A}_2$$

$$type_{N_1}(n) = type_{N_2}(h(n))$$

$$\{h(n') \mid n' \in children_{N_1}(n)\} = children_{N_2}(h(n))$$

and whenever either side is defined:

$$h(default_{N_1}(n)) = default_{N_2}(h(n))$$

Table 1: Properties of the bijection h witnessing $\equiv_{\mathrm{SPS}}$

The step semantics of statecharts given in [PS91] is defined in terms of SPS tuples. We defer the presentation of the semantics of SPS tuples until Section 4, since the algebra SA, presented in the next section, is useful in that discussion.

3 An Inductively Defined Syntax for Statecharts

Statecharts are constructed from components that themselves resemble (smaller) statecharts, and this leads to the idea of defining operations on statecharts that construct complex statecharts from collections of simpler ones. The algebra SA (for statecharts algebra) introduced in this section captures the structure of SPS tuples. After defining SA we present the translation function ϕ : SA $\rightarrow$ SPS. We conclude this section with several theorems demonstrating that the algebra SA is in one-to-one correspondence with the set of SPS tuples. Appendix B presents the details of the proofs for these theorems and their supporting lemmas.

3.1 Statecharts Algebra SA

The signature of SA consists of two types of operators, a k-place sequential composition operator representing *or*-boxes (i.e. FSA-like structures having k "states") and a binary parallel composition operator representing *and*-boxes. More specifically, for $p_1, \ldots, p_k \in$ SA, $K = \{1, \ldots, k\}$, and finite $\rightarrow \subset K \times \mathcal{L}(\mathcal{E} \cup \overline{\mathcal{E}}) \times K$,

$$[(p_1, \ldots, p_k), \rightarrow]$$

is an *or*-term corresponding to an SPS *or*-box. Terms $p_1, \ldots, p_k$ and arrow set $\rightarrow$ are respectively the boxes and arrows of the underlying FSA. Note that an arrow $(i, l, j) \in \rightarrow$ refers to its source p_i and destination p_j by their places in the list $p_1, \ldots, p_k$. We also say that $type([(p_1, \ldots, p_k), \rightarrow]) = or$.

For $p_1, p_2 \in$ SA,

$$p_1 \times p_2$$

is an *and*-term corresponding to an SPS *and*-box with p_1 and p_2 in parallel, and $type(p_1 \times p_2) = and$.

For convenience we abbreviate the 0-place *or*-term $[(), \emptyset]$ by $\Box$. Finally, an SA-context $X[\cdot]$ is an SA term in which some sub-term has been replaced by $[\cdot]$.

Anticipating the translation function ϕ given below, we present the SA terms corresponding to the statechart of Figure 1.

$$\begin{aligned} p_S &= [(p_A, \Box), \{(1, \mathit{off}, 2), (2, \mathit{on}, 1)\}] \\ p_A &= p_B \times p_C \\ p_B &= [(\Box, \Box), \{(1, a/b, 2), (2, a, 1)\}] \\ p_C &= [(\Box, \Box), \{(1, b/c, 2), (2, b, 1)\}] \end{aligned}$$

3.2 Translating SA into SPS

The function ϕ takes an arbitrary SA term and constructs an SPS tuple. Since an SPS tuple has four parts we define ϕ as follows:

$$\phi(p) = (\mathrm{E}(p), N(p), \mathcal{A}(p), r(p))$$

Similarly $N(p)$ is the structure:

$$(N(p), \mathit{children}_{N(p)}, \mathit{type}_{N(p)}, \mathit{default}_{N(p)})$$

Of the above $N(p)$ is the most complicated, since SA terms have no associated names, and the actual choice of names must be reflected in $\mathcal{A}(p)$ and $r(p)$.

$\mathrm{E}(p)$ simply gathers up all the primitive events mentioned in a term p:

$$\mathrm{E}(p) = \begin{cases} \bigcup_{1 \le i \le k} \mathrm{E}(p_i) \cup \bigcup_{(i,l,j) \in \rightarrow} \mathit{primitives}(l) & p = [(p_1, \ldots, p_k), \rightarrow] \\ \mathrm{E}(p_1) \cup \mathrm{E}(p_2) & p = p_1 \times p_2 \end{cases}$$

$N(p)$ constructs a set of names for the sub-terms of a term p. SA terms do not have names directly associated with them, since a sub-term p' must get a distinct name for each occurrence in p. The scheme presented here for producing names (others could be devised) is to use the natural numbers $\mathcal{N}$ to label sub-terms $p_1, \ldots, p_k$ in $[(p_1, \ldots, p_k), \rightarrow]$ and p_1, p_2 in $p_1 \times p_2$. A string of numbers in $\mathcal{N}^*$ (Kleene star) identifies a unique sub-term of a given term. The notation for a string of numbers of length m is $< i_1, \ldots, i_m >$, and we use the notational conventions: $< \vec{\imath} >$ for a string; $< \vec{\imath}, \vec{\imath}' >$ for the concatenation of strings; $< i, \vec{\imath} >$ for prefixing a number to a string; and $< \vec{\imath}, i >$ for postfixing a number to a string. Otherwise, we may refer to a name simply as n. A string of numbers $< \vec{\imath} >$ has *prefixes* given by $* < \vec{\imath} >= \{< \vec{\jmath} > \mid < \vec{\imath} >=< \vec{\jmath}, \vec{\jmath}' >\}$, and *proper prefixes* $+ < \vec{\imath} >$ given by the same set except excluding $< \vec{\imath} >$. Finally, the *longest common prefix* (lcp) of names n_1 and n_2 is a name n defined such that $*n = *n_1 \cap *n_2$.

In order to define the set $N(p)$ we need the function $\mathit{names} : \mathrm{SA} \rightarrow 2^{\mathcal{N}^*}$, which assigns names to the sub-terms of a term, and $\mathit{subterm} : \mathrm{SA} \rightarrow (2^{\mathcal{N}^*} \rightarrow \mathrm{SA})$, which extracts a named sub-term from a term. We write the sub-term named n of a term p as $\mathit{subterm}_p(n)$. If one imagines a term represented by its abstract syntax tree (AST), then a name is assigned to a sub-term by numbering siblings in the AST from left to right, and collecting the numbers on the path from the root to the

node representing the sub-term. Thus the name of the root of the AST (the sub-term containing the whole term) is the empty list $<>$. $names(p)$ is the set of all such lists from a term p and is defined:

$$\begin{aligned} names(\Box) &= \{<>\} \\ names([(p_1,\ldots,p_k),\rightarrow]) &= \{<>\} \cup \bigcup_{1\le i\le k}\{<i,\vec{\imath}>|<\vec{\imath}>\in names(p_i)\} \\ names(p_1 \times p_2) &= \{<>\} \cup \bigcup_{1\le i\le 2}\{<i,\vec{\imath}>|<\vec{\imath}>\in names(p_i)\} \end{aligned}$$

The following table gives the names of the sub-terms for the statechart of Figure 1:

S	A	B	C	D	E	F	G	H
$<>$	$<1>$	$<1,1>$	$<1,2>$	$<2>$	$<1,1,1>$	$<1,1,2>$	$<1,2,1>$	$<1,2,2>$

Given a term p and a name $n \in names(p)$ we will often want to extract the sub-term of p named by n. $subterm_p : names(p) \rightarrow$ SA is defined such that

$$\begin{aligned} subterm_p(<>) &= p \\ subterm_{[(p_1,\ldots,p_k),\rightarrow]}(<i,\vec{\imath}>) &= subterm_{p_i}(<\vec{\imath}>) \quad 1\le i\le k \\ subterm_{p_1\times p_2}(<i,\vec{\imath}>) &= subterm_{p_i}(<\vec{\imath}>) \quad 1\le i\le 2 \end{aligned}$$

For $n \notin names(p)$, $subterm_p(n)$ is undefined.

Now we are in a position to define $N(p)$. Let p be an SA term, and let $\phi(p_i) = (\mathrm{E}_i, N_i, \mathcal{A}_i, r_i)$ be the translation of the sub-terms of p, i.e. $p = [(p_1,\ldots,p_k),\rightarrow]$ and $1 \le i \le k$, or $p = p_1 \times p_2$ and $1 \le i \le 2$.

- $$N(p) = names(p) - \{<\vec{\imath},j> | type(subterm_p(<\vec{\imath}>)) = type(subterm_p(<\vec{\imath},j>)) = and\}$$

  i.e. remove the names of nested *and*-sub-terms.

- For each $<\vec{\imath}>\in N(p)$

  $$\begin{aligned} children_{N(p)}(<\vec{\imath}>) &= \{<\vec{\imath},\vec{\jmath}> \mid \nexists <\vec{\imath},\vec{\jmath}'>\in N(p).\ <\vec{\jmath}'>\in +<\vec{\jmath}>\} \\ type_{N(p)}(<\vec{\imath}>) &= type(subterm_p(<\vec{\imath}>)) \end{aligned}$$

  for $n \in N$, $type_{N(p)}(n) = or$ and $children_{N(p)}(n) \neq \emptyset$ we have

  $$default_{N(p)}(n) = \begin{cases} <1> & n = <> \\ <i,\vec{\jmath}> & n = <i,\vec{\imath}>,<\vec{\jmath}> = default_{N_i(p_i)}(<\vec{\imath}>) \end{cases}$$

The arrow set and the root are:

- $\mathcal{A}(p) = \bigcup_{<\vec{\imath}>\in N(p)} \{(<\vec{\imath},j>, l, <\vec{\imath},j'>) | subterm_p(<\vec{\imath}>) = [(p_1,\ldots,p_k),\rightarrow], (j,l,j') \in \rightarrow\}$
- $r(p) = <>$

Parallel composition is binary in SA but not in SPS. ϕ handles this difference by removing the extra names corresponding to nested *and*-terms.

A given statechart may correspond to several SA terms, and we don't want to distinguish between them. Two SA terms p and p' are syntactically equivalent, written $p \equiv_{\mathrm{SA}} p'$, if $\phi(p) \equiv_{\mathrm{SPS}} \phi(p')$. The following theorems establish that the syntax SA is interchangeable with the statecharts-tuples syntax of SPS. A detailed presentation of the proofs may be found in Appendix B.

Theorem 3.1 $\equiv_{\mathrm{SA}}$ *is a congruence for the operators of* SA. *That is, if* $p \equiv_{\mathrm{SA}} p'$ *then:*

$$\begin{aligned} [(p_1,\ldots,p_{i-1},p,p_{i+1},\ldots,p_k),\rightarrow] &\equiv_{\mathrm{SA}} [(p_1,\ldots,p_{i-1},p',p_{i+1},\ldots,p_k),\rightarrow] \quad 1 \le i \le k \\ p_1 \times p &\equiv_{\mathrm{SA}} p_1 \times p' \\ p \times p_2 &\equiv_{\mathrm{SA}} p' \times p_2 \end{aligned}$$

The order of application of nested *and*-terms and the order (after the default) of sub-terms in an *or*-term are unimportant. Table 2 gives the set $\mathrm{SA}_{\equiv}$ of equations expressing this.

$p_1 \times p_2$	$\equiv_{\mathrm{SA}}$	$p_2 \times p_1$	SA_1
$p_1 \times (p_2 \times p_3)$	$\equiv_{\mathrm{SA}}$	$(p_1 \times p_2) \times p_3$	SA_2
$[(p_1,\ldots,p_i,\ldots,p_j,\ldots,p_k),\rightarrow]$	$\equiv_{\mathrm{SA}}$	$[(p_1,\ldots,p_j,\ldots,p_i,\ldots,p_k),\rightarrow']$	SA_3
		for $i,j \neq 1$ and $\rightarrow' = \rightarrow[i/j,j/i]$	
where $\rightarrow[i/j,j/i]$ simultaneously replaces is with js and vice versa			

Table 2: $\mathrm{SA}_{\equiv}$: The axiomatization of $\equiv_{\mathrm{SA}}$

Theorem 3.2 $\mathrm{SA}_{\equiv}$ *gives a sound and complete axiomatization for* $\equiv_{\mathrm{SA}}$.

For $p \in \mathrm{SA}$, let $[p]_{\equiv_{\mathrm{SA}}}$ be the equivalence class in $\mathrm{SA}/\equiv_{\mathrm{SA}}$ containing p, and let $\phi_{[\cdot]} : \mathrm{SA}/\equiv_{\mathrm{SA}} \rightarrow \mathrm{SC}$ be defined such that

$$\phi_{[\cdot]}([p]_{\equiv_{\mathrm{SA}}}) = [\phi(p)]_{\equiv_{\mathrm{SPS}}}$$

Theorem 3.3 $\phi_{[\cdot]}$ *is a bijection.*

In light of the above it is clear that the two syntaxes are equivalent, and henceforth we use them interchangeably.

4 The Step Semantics of Statecharts

An SA term (or an SPS tuple) gives the static structure of a statechart, but it does not contain state information. In [PS91], Pnueli and Shalev define the *step semantics* for statecharts in terms of a set of *configurations*, or global states, and a set of *steps*, or semantic transitions between configurations. In this section, we characterize the step semantics of [PS91] as a mapping $\Psi : \mathrm{SA} \rightarrow \mathbf{LTS}$, where

LTS is a domain of *labeled transition systems*. An element of **LTS** is a four-tuple $(Q, \Xi, q_0, \longrightarrow)$, where Q is the set of *states*, Ξ is the *alphabet*, $q_0 \in Q$ is the *start state*, and $\longrightarrow \subseteq Q \times \Xi \times Q$ is the *transition relation*. A transition $(q, \xi, q') \in \longrightarrow$ is usually written $q \xrightarrow{\xi} q'$.

We use LTS isomorphism as our notion of semantic equality. Two LTSs $(Q_1, \Xi_1, q_1, \longrightarrow_1)$ and $(Q_2, \Xi_2, q_2, \longrightarrow_2)$ are isomorphic, written $(Q_1, \Xi_1, q_1, \longrightarrow_1) \equiv (Q_2, \Xi_2, q_2, \longrightarrow_2)$, if there is a bijection h between their states such that $h(q_1) = q_2$ and for $q, q' \in Q_1$ and $\xi_1 \in \Xi_1$ and $\xi_2 \in \Xi_2$: $q \xrightarrow{\xi_1}_1 q'$ if and only if $h(q) \xrightarrow{\xi_2}_2 h(q')$ with $\xi_1 = \xi_2$.

4.1 The Semantic Mapping $\Psi : \mathrm{SA} \rightarrow$ LTS

To define the mapping Ψ, we first need to define, for a given SA term p, the entities $\mathcal{C}(p)$, $\Xi(p)$, $\Delta(p)$, and $\longrightarrow_p$.

$\mathcal{C}(p)$ is the set of *configurations* of p and will be the state set of the LTS $\Psi(p)$. A configuration $c \in \mathcal{C}(p)$ is a maximal set of mutually parallel leaves in $names(p)$, where the leaves of N are the names $\{n \in names(p) | \nexists n' \in names(p).\ n \in +n'\}$. $\mathcal{C} : \mathrm{SA} \rightarrow 2^{\mathcal{N}^*}$ is defined as follows:

$$\begin{aligned} \mathcal{C}(\Box) &= \{\{<>\}\} \\ \mathcal{C}([(p_1, \ldots, p_k), \rightarrow]) &= \bigcup_{1 \leq i \leq k} \{< i, c > | \ c \in \mathcal{C}(p_i)\} \\ \mathcal{C}(p_1 \times p_2) &= \{< 1, c_1 > \cup < 2, c_2 > | \ c_1 \in \mathcal{C}(p_1), c_2 \in \mathcal{C}(p_2)\} \end{aligned}$$

where $< i, c >$ is shorthand for the pointwise (pre-)extension of each name $< \vec{\imath} > \in c$ by the integer i, i.e. $< i, c >= \{< i, \vec{\imath} > \ | \ < \vec{\imath} > \in c\}$. We extend the prefix closure operation by applying it pointwise to sets of names. Thus $*c = \bigcup_{n \in c} *n$.

$\Delta(p)$ is the *initial configuration* of p, and will represent the initial state of $\Psi(p)$. $\Delta(p)$ is the unique configuration that always chooses the default of an *or*-box, and is defined analogously to $\mathcal{C}(p)$.

$$\begin{aligned} \Delta(\Box) &= \{<>\} \\ \Delta([(p_1, \ldots, p_k), \rightarrow]) &= < 1, \Delta(p_1) > \\ \Delta(p_1 \times p_2) &= \Delta(p_1) \cup \Delta(p_2) \end{aligned}$$

Statecharts specify *reactive* systems meaning that a statechart reacts to some *input* and produces some *output*. The semantic alphabet consists of a set of input/output pairs, written ξ_i / ξ_o. Each of ξ_i and ξ_o is a subset of $\mathrm{E}(p)$. By convention ξ_i and ξ_o are denoted by listing their elements, as in ab/cd. (Do not be mislead by the resemblance to syntactic labels – they are different.) The set $\Xi(p) = \mathrm{E}(p) \times \mathrm{E}(p)$ of input/output pairs is the alphabet of $\Psi(p)$.

$\longrightarrow_p \subseteq \mathcal{C}(p) \times \Xi(p) \times \mathcal{C}(p)$ is the set of *steps* of p and will be the transition relation of $\Psi(p)$. To define $\longrightarrow_p$, it is first necessary to identify the *admissible arrow sets* of p, intuitively, the subsets of $\mathcal{A}(p)$ that may be gathered together to form a step. The following definitions are needed to specify which sets $A \subseteq \mathcal{A}(p)$ are admissible sets[4].

Orthogonal: Two arrows α_1 and α_2 are orthogonal, written $\alpha_1 - \alpha_2$ if $source(\alpha_1)$ is in parallel with $source(\alpha_2)$. i.e. $type(lcp(source(\alpha_1), source(\alpha_2))) = and$. Note that $\alpha - \alpha$.

[4] In each case the definition is relative to some given $p \in \mathrm{SA}$. When the context warrants we will subscript the term or function with the particular $p \in \mathrm{SA}$ intended.

Consistent: For a set of arrows $A \subseteq \mathcal{A}(p)$ the set of all arrows orthogonal to every arrow in A is given by the function $consistent : 2^{\mathcal{A}(p)} \rightarrow 2^{\mathcal{A}(p)}$ such that $consistent(A) = \{\alpha \in \mathcal{A}(p) \mid \forall \alpha' \in A.\ \alpha - \alpha'\}$.

Relevant: For a configuration $c \in \mathcal{C}(p)$ the relevant arrows are given by the function $relevant : \mathcal{C}(p) \rightarrow 2^{\mathcal{A}(p)}$ such that $relevant(c) = \{\alpha \in \mathcal{A}(p) \mid source(\alpha) \in *c\}$.

Triggered: For a set of events $Ev \subseteq \mathrm{E}(p)$ the set of arrows triggered by Ev is given by the function $triggered : 2^{\mathrm{E}(p)} \rightarrow 2^{\mathcal{A}(p)}$ such that

$$triggered(Ev) = \left\{ \alpha \in \mathcal{A}(p) \;\middle|\; \begin{array}{lcl} \pi \in trigger(\alpha) & \Rightarrow & \pi \in Ev \\ \overline{\pi} \in trigger(\alpha) & \Rightarrow & \pi \notin Ev \end{array} \right\}$$

Actions: The set of output actions generated by a set of arrows $A \subseteq \mathcal{A}(p)$ is given by the function $actions : 2^{\mathcal{A}(p)} \rightarrow 2^{\mathrm{E}(p)}$ such that $actions(A) = \bigcup_{\alpha \in A} action(\alpha)$.

Enabled: A set of arrows $A \subseteq \mathcal{A}(p)$ in a configuration $c \in \mathcal{C}(p)$ and with input $\xi_i \subseteq \mathrm{E}(p)$ enables the set of arrows given by the function $enabled : \mathcal{C}(p) \times 2^{\mathrm{E}(p)} \times 2^{\mathcal{A}(p)} \rightarrow 2^{\mathcal{A}(p)}$ such that

$$enabled(c, \xi_i, A) = relevant(c) \cap consistent(A) \cap triggered(\xi_i \cup actions(A))$$

When a configuration c and input ξ_i are understood we treat *enabled* as a function from $2^{\mathcal{A}(p)}$ to $2^{\mathcal{A}(p)}$. The **Synchrony Hypothesis**, mentioned in the introduction, is that an arrow set A must satisfy $enabled(A) \subseteq A$ in order to be admissible. **Global Consistency** is the requirement that $A \subseteq enabled(A)$. Thus an admissible arrow set must be a solution to the equation $A = enabled(A)$.

Separable: A set of arrows A is *separable* (for a given configuration and input) if

$$\exists A' \subset A.\ enabled(A') \cap (A - A') = \emptyset$$

and is *inseparable* otherwise. The **Causality** requirement is that an admissible arrow set must be inseparable.

Admissible arrow sets: The admissible arrow sets of a statechart in configuration c with input ξ_i are given by the function $admit : \mathcal{C}(p) \times 2^{\mathrm{E}(p)} \rightarrow 2^{\mathcal{A}(p)}$ such that

$$admit(c, \xi_i) = \{A \subseteq \mathcal{A}(p) \mid A \text{ is inseparable and } enabled(c, \xi_i, A) = A\}$$

We also write $A \in admit(c)$ for $\exists \xi_i.\ A \in admit(c, \xi_i)$[5].

Next configuration: For any relevant, consistent arrow set A, i.e. $A \subseteq relevant(c) \cap consistent(A)$, the next configuration c' is given by the function $next : \mathcal{C}(p) \times 2^{\mathcal{A}(p)} \rightarrow \mathcal{C}(p)$ such that

$$\begin{array}{rcl} next(c, A) & = & (c - \bigcup_{\alpha \in A} \{n \in c \mid source(\alpha) \in *n\}) \\ & & \cup \bigcup_{\alpha \in A} \{< \vec{\imath}, \Delta(subterm_p(\vec{\imath})) > \mid < \vec{\imath} >= target(\alpha)\} \end{array}$$

in particular $next(c, \emptyset) = c$.

[5]This definition conforms to that given in [PS91]. It has been pointed out to the author that there are statecharts for which there is no solution to $A = enabled(A)$ (C.f. Appendix D). Thus $admit(c, \xi_i) = \emptyset$. This conflicts with the claim that statecharts specify reactive systems; wherein there is a defined response to every input. One may choose to set $admit(c, \xi_i) = \{\emptyset\}$ in such a case and doing so would have no effect on the results reported in this paper.

Each admissible arrow set A of p gives rise to one or more steps and $\longrightarrow_p$ may be defined as:

$$\longrightarrow_p = \left\{ (c, \xi_i/\xi_o, c') \;\middle|\; \begin{array}{l} c \in \mathcal{C}(p), \xi_i \in 2^{\mathrm{E}(p)}, A \in \mathit{admit}(c, \xi_i), \\ \xi_o = \mathit{actions}(A), \text{ and } c' = \mathit{next}(c, A) \end{array} \right\}$$

The semantic mapping is $\Psi(p) = (\mathcal{C}(p), \mathrm{E}(p), \Delta(p), \longrightarrow_p)$ and SA terms p and p' are semantically equal, written $p =_{\mathrm{SC}} p'$, if $\Psi(p) \equiv \Psi(p')$.

Remark 4.1 *The example statecharts A and A' and context $X[\cdot]$ of Figure 2 demonstrate that $=_{\mathrm{SC}}$ is not a congruence for* SA. *That is, the step semantics is not compositional.*

Appendix D gives a formal presentation of the above result as well as examples of applying the definitions leading to admissibility. There we show that statechart A has a step (from its initial configuration) on input a, but statechart A' does not, and this is because the arrow set A' is separable. Since the semantics of A and A' are isomorphic but $X[A]$ and $X[A']$ have differently labeled transitions we get the following:

Remark 4.2 *If we use Ψ as our semantic mapping, then any equivalence over LTS that refines trace equivalence will not produce a congruence for* SA.

In Section 5 we present an alternative semantics and show that it is compositional. We conclude this section by showing:

- $=_{\mathrm{SC}}$ is preserved by the *or* operator, and
- *enabledness*, i.e. the existence of a solution to $A = \mathit{enabled}(A)$, is preserved by the *and* operator.

$=_{\mathrm{SC}}$ is not preserved by the *and* operator, and this is because *inseparability*, i.e. that a solution to $A = \mathit{enabled}(A)$ is inseparable is not preserved by the *and* operator[6].

Lemma 4.1 $=_{\mathrm{SC}}$ *is preserved by the* or *operator.*

Proof sketch: Let $p, p' \in$ SA, with $p =_{\mathrm{SC}} p'$, witnessed by $h : \mathcal{C}(p) \to \mathcal{C}(p')$, and let $X[\cdot] = [(p_1, \ldots, p_{i-1}, [\cdot], p_{i+1}, \ldots, p_k), \rightarrow]$ for some $k > 0$ and $1 \leq i \leq k$.

$$p =_{\mathrm{SC}} p' \text{ iff } X[p] =_{\mathrm{SC}} X[p']$$

Observe that the bijection h can be extended to a bijection $h_X : \mathcal{C}(X[p] \to \mathcal{C}(X[p'])$ (see Lemma C.2) such that a $c_p \in \mathcal{C}(p)$ is in one-to-one correspondence with $< i, c_p > \in \mathcal{C}(X[p])$. Verify that a solution $A_{X[p]}$ of $A = \mathit{enabled}_{X[p]}(< i, c_p >, \xi_i, A)$ exists and is inseparable if and only if $A_p = \{(< \vec{\imath} >, l, < \vec{\imath}' >) | (< i, \vec{\imath} >, l, < i, \vec{\imath}' >) \in A_{X[p]}\}$ is a solution to $A = \mathit{enabled}_p(c_p, \xi_i, A)$ and A_p is inseparable. $\alpha \in \mathit{relevant}(< j, c_p >)$, $i \neq j$, is not relevant to $< i, c_p >$. Since $p =_{\mathrm{SC}} p'$ there is an $A_{p'}$ that is a solution to $A = \mathit{enabled}_{p'}(h(c_p), \xi_i, A)$ and is inseparable. The same reasoning as above shows

[6] Detailed proofs of the following two lemmas may be found in Appendix C.

Figure 3: The proof strategy for $p_1 \times [\cdot]$.

that there is a corresponding $A_{X[p']}$ that is a solution to $A = \mathit{enabled}_{X[p']}(h_X(< i, c_p >), \xi_i, A)$ and $A_{X[p']}$ is inseparable. (C.f. Appendix C) □

Inseparability fails to be preserved in some contexts, and this is the only way that a context ever distinguishes two $=_{\mathrm{SC}}$ terms. The following lemma establishes that enabledness is preserved by the *and* operator. If enabledness and inseparability were *both* preserved by the *and* operator then so would be admissibility, and $=_{\mathrm{SC}}$ would be a congruence.

Lemma 4.2 *For $p, p', p_1 \in \mathrm{SA}$ with $p =_{\mathrm{SC}} p'$ witnessed by $h : \mathcal{C}(p) \to \mathcal{C}(p')$ let $c_p \in \mathcal{C}(p), c_1 \in \mathcal{C}(p_1)$. If for some ξ_i there is an $A_p \in \mathrm{admit}_{p_1 \times p}(< 1, c_1 > \cup < 2, c_p >, \xi_i, A_p)$ then there is a solution $A_{p'}$ to*

$$A = \mathrm{enabled}_{p_1 \times p'}(< 1, c_1 > \cup < 2, h(c_p) >, \xi_i, A)$$

Proof sketch: The general proof strategy is summarized in Figure 3. We begin with a step of $p_1 \times p$. Let $X[\cdot] = p_1 \times [\cdot]$ and the step be

$$c \xrightarrow{\xi_i/\xi_o}_{X[p]} c'$$

There is a step

$$c_p \xrightarrow{\xi_i'/\xi_o'}_{p} c_p'$$

of p that corresponds to that of $X[p]$, but may need some extra input from actions generated in p_1. Since $p =_{\mathrm{SC}} p'$ there is also a step

$$h(c_p) \xrightarrow{\xi_i'/\xi_o'}_{p'} h(c_p')$$

$A_{p'}$ gathers the arrows of p_1 in the original step together with the arrows of p' in the latter step and is a solution to $A = \mathit{enabled}(A)$. Showing that $A_{p'}$ is indeed a solution is accomplished by applying the definition of *enabled* to the particular case of $A_{p'}$. Appendix C presents the details. □

So far our notion of semantic equivalence has been the identity relation on LTSs (LTS isomorphism). We have found that there are equivalent SA terms that can be distinguished given the right context, and could be distinguished even if we used some notion of trace equivalence. In Section 5 we introduce a means of identifying when two SA terms p and p' with $p =_{\mathrm{SC}} p'$ have a distinguishing context. A new semantic mapping incorporating this insight leads to an equivalence $\simeq$. We find that $\simeq$ preserves inseparability and is the largest congruence contained in $=_{\mathrm{SC}}$.

5 The Orderly Step Semantics: A Compositional Semantics for SA

In this section we present $\Psi_>$, a new semantic mapping for statecharts terms that we show to be compositional. $\Psi_>$ will be such that $\Psi_>(p) = (\mathcal{C}(p),\ \mathcal{W}(p),\ \Delta(p),\ \succ\!\!\rightarrow_p)$ where $\mathcal{C}(p)$ (the configurations of p) and $\Delta(p)$ (the initial configuration of p) are as before. The semantic alphabet is given a new definition, and is now called $\mathcal{W}(p)$, the set of *words* of p. We use $\succ\!\!\rightarrow_p$ for the transition relation of $\Psi_>(p)$. We refer to the new semantic domain as $\mathbf{LTS}_>$, and thus $\Psi_> : \mathrm{SA} \rightarrow \mathbf{LTS}_>$. Our notion of semantic equivalence $\simeq$ is $\mathrm{LTS}_>$ isomorphism, written $\equiv_>$: For $p, p' \in \mathrm{SA}$, $p \simeq p'$ if $\Psi_>(p) \equiv_> \Psi_>(p')$.

The semantic alphabet $\mathcal{W}(p)$ is built from the sets of events represented in LTS transitions and the arrows sets that lead to them. $\mathcal{W}(p) \subset 2^{\mathrm{E}(p)} \times 2^{\mathrm{E}(p)\times\mathrm{E}(p)}$ has elements that consist of a set of primitive events together with a relation called the *causal ordering* of w.[7]

The function $\psi_A : (\mathrm{E}(p) \times \mathrm{E}(p)) \rightharpoonup \mathcal{W}(p)$ constructs a word from an LTS label ξ_i/ξ_o and an admissible arrow set A, with $A \in \mathit{admit}_p(c, \xi_i)$ (for some $c \in \mathcal{C}(p)$) and $\xi_o = \mathit{actions}(A)$. Let $\Lambda_A = \xi_i \cup \xi_o$ and $R_A \subseteq \Lambda_A \times \Lambda_A$ be defined such that

$$R_A = \{(\lambda, \lambda') | \exists \alpha \in A.\ \lambda \in \mathit{trigger}(\alpha), \lambda' \in \mathit{action}(\alpha)\}$$

and let $R_A^{\xi_i}$ be defined such that

$$R_A^{\xi_i} = R_A - \{(\lambda, \lambda') \in R_A | \lambda' \in \xi_i\}$$

The causal relation $>_A^{\xi_i}$ is the transitive closure of $R_A^{\xi_i}$:

$$>_A^{\xi_i} = (R_A^{\xi_i})^+$$

So $\psi_A(\xi_i/\xi_o) = (\Lambda_A, >_A^{\xi_i})$.

We follow the convention that the causal relation of a word is subscripted with its set, as in $(w, >_w)$, and if no confusion will result we write just w meaning $(w, >_w)$. For a word w we have $\mathit{trigger}(w) = \{\lambda \in w \mid \nexists \lambda'.\ \lambda' >_w \lambda\}$, i.e. the maximal elements of w, and $\mathit{action}(w) = \{\lambda \in w \mid \exists \lambda'.\ \lambda' >_w \lambda\}$.

Let $w = \psi_A(\xi_i/\xi_o)$ for some arrow set A. We say that $\pi \in \mathit{action}(w)$ is *triggered* in A if for each $\lambda \in \mathit{trigger}(w)$ such that $\lambda >_w \pi$ there exists a $\alpha \in A$ with $\lambda \in \mathit{trigger}(\alpha)$ such that

$$\forall \pi' \in \mathit{trigger}(\alpha).\ \pi' \in \mathit{trigger}(w)$$

The intuition behind this definition is that α is the "final cause" of π. We already know that $\overline{\pi'} \in \mathit{trigger}(\alpha) \Rightarrow \pi' \notin \Lambda_A$, since $A = \mathit{enabled}(A)$. Call π *triggerless* in A if π is not triggered in A. If for all $\pi \in \mathit{action}(w)$, π is triggered in A then we say that w is *well-triggered* for A. The following lemma relates LTS labels to $\mathrm{LTS}_>$ labels and asserts that well-triggeredness and inseparability are equivalent notions. A detailed proof appears in Appendix E.

[7] The *words* used here as semantic labels resemble a simplified form of the *clock records* of [HGdR89]; though the syntax and semantics are different.

Lemma 5.1 *Let p be some* SA *term in configuration c, with $A = \text{enabled}(c, \xi_i, A)$ and $\xi_o = \text{actions}(A)$*

1. $\text{trigger}(\psi_A(\xi_i/\xi_o)) = \xi_i$
2. $\text{action}(\psi_A(\xi_i/\xi_o)) = \xi_o - \xi_i$
3. *A is inseparable if and only if $\psi_A(\xi_i/\xi_o)$ is well-triggered.*

Proof sketch: Parts 1 and 2 follow from the definition of ψ_A and the fact that $A = \mathit{enabled}(c, \xi_i, A)$. We show

$$\psi_A(\xi_i/\xi_o) \text{ is well-triggered } \Rightarrow A \text{ is inseparable}$$

If there is a triggerless event $\pi \in \psi_A(\xi_i/\xi_o)$ there is an arrow α such that $\pi \in \mathit{action}(\alpha)$. Let A_s be the set of such α's for all triggerless π. We find that $\mathit{enabled}(A - A_s) \cap A_s = \emptyset$, and A is separable. Next we show

$$A \text{ is inseparable } \Rightarrow \psi_A(\xi_i/\xi_o) \text{ is well-triggered}$$

For $\psi_A(\xi_i/\xi_o)$ to be well-triggered means that each $\alpha \in A$ either has $\mathit{trigger}(\alpha) \subseteq \xi_i$ or each $\pi \in \mathit{trigger}(\alpha) - \xi_i$ is such that $\pi \in \mathit{action}(\alpha')$ for some other arrow. The result then follows from the finiteness of A and the fact that $A = \mathit{enabled}(A)$. □

We say that words $w \in \mathcal{W}(p)$, and $w' \in \mathcal{W}(p')$ *match*, written $w \approx w'$, if $\mathit{trigger}(w) = \mathit{trigger}(w')$, $\mathit{action}(w) = \mathit{action}(w')$ and

$$\forall \lambda \in \mathit{trigger}(w), \lambda' \in \mathit{action}(w) \lambda >_w \lambda' \text{ iff } \lambda >_{w'} \lambda'$$

$\succ\!\!\rightarrow_p \subseteq \mathcal{C}(p) \times \mathcal{W}(p) \times \mathcal{C}(p)$ is the set of *orderly steps* (or *osteps*) of p and will be the transition relation of $\Psi_>(p)$. We obtain $\succ\!\!\rightarrow_p$ from $\longrightarrow_p$ via ψ_A as follows: For $c, c' \in \mathcal{C}(p)$ and $A \in \mathit{admit}(c, \xi_i)$, $\xi_o = \mathit{actions}(A)$, and $c' = \mathit{next}(c, A)$

$$c \xrightarrow{\xi_i/\xi_o}_p c' \quad \Rightarrow \quad c \overset{\psi_A(\xi_i/\xi_o)}{\succ\!\!\longrightarrow}_p c'$$

The new semantic function $\Psi_>$ is

$$\Psi_>(p) = (\mathcal{C}(p), \mathcal{W}(p), \Delta(p), \succ\!\!\rightarrow_p)$$

Recall that LTS isomorphism is given by a bijection $h : \mathcal{C}(p) \rightarrow \mathcal{C}(p')$ that matches states and transitions. We use $\approx$ rather than identity (set equality) between words in matching transitions. Thus $\equiv_>$ is defined such that $h(\Delta(p)) = \Delta(p')$ and for $c, c' \in \mathcal{C}(p)$, $w \in \mathcal{W}(p)$, and $w' \in \mathcal{W}(p')$ we have:

$$c \overset{w}{\succ\!\!\longrightarrow}_p c' \quad \Leftrightarrow \quad h(c) \overset{w'}{\succ\!\!\longrightarrow}_{p'} h(c') \quad \text{with } w \approx w'$$

The new semantic equivalence $\simeq$ is

$$p \simeq p' \text{ iff } \Psi_>(p) \equiv_> \Psi_>(p')$$

We refer to the above semantic mapping $\Psi_>$ and equivalence $\simeq$ the *orderly step semantics*.

An equivalence $\cong \subseteq \text{SA} \times \text{SA}$ is a *congruence* for SA if for all SA terms p and p' and contexts $X[\cdot]$

$$p \cong p' \Rightarrow X[p] \cong X[p']$$

A congruence $\cong$ is $=_{\text{SC}}$ *respecting* if $p \cong p' \Rightarrow p =_{\text{SC}} p'$.

The following theorem is the main technical result of this paper. It shows that the semantics we have presented is compositional, and has the additional merit that $\simeq$ refines $=_{\text{SC}}$ as little as possible. A detailed proof of this result may be found in Appendix E.

Theorem 5.1 $\simeq$ *is the largest* $=_{\text{SC}}$ *respecting congruence.*

Proof sketch: That *neweq* respects *sceq*, i.e. $\simeq \subseteq =_{\text{SC}}$ is easily seen, and Lemmas 4.1 and 4.2 demonstrate that the only thing left to show in establishing that $\simeq$ is a congruence is whether inseparability is preserved by the *and* operator. Given that result an induction on the structure of contexts $X[\cdot]$ gives that for all $p, p' \in \text{SA}$

$$p \simeq p' \Rightarrow X[p] \simeq X[p']$$

If there are SA terms p, p' and p_1 such that $p \simeq p'$ and $X[\cdot] = p_1 \times [\cdot]$ then Lemma 4.2 gives us that for an arrow set $A_{X[p]}$ producing the orderly step

$$< 1, c_1 > \cup < 2, c_p > \stackrel{w}{\rightarrowtail}_{X[p]} \quad < 1, c_1' > \cup < 2, c_p' >$$

with $\xi_i = \mathit{trigger}(w)$ and $\xi_o = \mathit{actions}(A_{X[p]})$, there is a solution $A_{X[p']}$ to

$$A = \mathit{enabled}_{X[p']}(< 1, c_1 > \cup h(< 2, c_p >), \xi_i, A)$$

with $\xi_o = \mathit{actions}(A_{p'})$. It only remains to show that $A_{X[p']}$ is inseparable, or equivalently, that $\psi_{A_{X[p']}}(\xi_i/\xi_o)$ is well-triggered. Let $A_{X[p]} = A_{p_1} \cup A_p$ and $A_{X[p']} = A_{p_1} \cup A_{p'}$ (abusing the notation slightly). For $\xi_i' = \xi_i \cup \mathit{actions}(A_{p_1})$ and $\xi_o = \mathit{actions}(A_p)$ we know

$$\psi_{A_p}(\xi_i'/\xi_o') \approx \psi_{A_{p'}}(\xi_i'/\xi_o')$$

since $p \simeq p'$. Assume some $\pi \in \mathit{action}(w)$ is triggerless in $A_{X[p']}$ the above observation means it is triggerless in $A_{X[p]}$ as well. A contradiction.

$\simeq$ is the largest congruence continued in $=_{\text{SC}}$, since any time

$$p =_{\text{SC}} p' \text{ and } p \not\simeq p'$$

it is quite easy to construct the distinguishing context p_1 such that

$$p_1 \times p \neq_{\text{SC}} p_1 \times p'$$

This is accomplished by finding a step of p not matched by p' due to a difference in the causal ordering of words w and w'. If there is a $\pi >_{w'} \pi'$ but $\pi \not>_w \pi'$ then put an arrow labeled π'/π in p_1. The result is that p will have a step for which the matching arrow set in p' is separable. □

The semantics given by $\Psi_>$ improves upon that of Ψ by being compositional. This has been accomplished by providing the semantic labels of $\text{LTS}_>$ with the causal ordering of events in the syntax. That ordering captures the notion of inseparability in the property of well-triggeredness. Finally, identifying when two semantic labels match weakens the notion of LTS isomorphism just enough to make $\simeq$, not only a congruence, but the largest $=_{\text{SC}}$ respecting congruence.

6 Conclusion and Dissertation Proposal

This paper presents a linear syntax for statecharts in the form of an algebra SA. We have shown that SA is in one-to-one correspondence, modulo $\equiv_{SA}$, with the set-theoretic syntax of [PS91]. Furthermore, we have presented a sound and complete axiomatization of $\equiv_{SA}$ via the equations of Table 2. Thus SA may be used in analyzing the semantics of statecharts, and we have shown that the traditional notion of statecharts semantics, the step semantics, is not compositional. That is, $=_{SC}$ is not a congruence, nor is the step semantics compositional for any equivalence over LTSs that refines trace equivalence. Furthermore, we have shown that *inseparability* is the key feature of the step semantics that is not preserved by $=_{SC}$. Thus there is a need for a new semantic domain and a new semantic mapping for statecharts, and this new semantics must rectify the problem with inseparability.

We have presented such a domain, $LTS_>$, and mapping, $\Psi_>$, and shown this semantics, the *orderly step semantics*, to be compositional. With our characterization of when two semantic labels *match*, and therefore when two elements of $LTS_>$ are isomorphic, we have a notion of statechart equivalence that is not only an $=_{SC}$ respecting congruence, but the *largest* such congruence. This was achieved by using semantic labels that preserve some of the causal ordering information found in the syntax of statecharts. The causal ordering, in turn, supported an alternative characterization of *inseparability* in the notion of *well-triggeredness*.

The results presented are a foundation upon which we propose to construct a dissertation (see Appendix A for a proposed outline of the dissertation). Sections 1 to 4a in the outline will revise and extend what has been presented in this proposal. We briefly discuss the rest of the dissertation in the following paragraphs.

6.1 A Syntax Directed Definition of Orderly Steps

With the orderly step semantics we can construct a syntax-directed definition of $\Psi_>$ as carried out in [US94a]. There we presented Plotkin-style SOS proof rules [Plo81] for a related statecharts semantics. Such a construction improves upon the semantics presented in this paper in two ways. First, the characterization of the transition relation presented in this paper is, in practice, highly redundant. The redundancy can be removed in a natural way, and the modified structure of semantic labels that results is amenable to the above style of proof rules. Second, as presented in this paper the orderly step semantics still relies on a large number of complex and interrelated definitions, and the semantic mapping must be applied fresh to each term of SA. A syntax directed definition of $\Psi_>$ allows one to reason via a few rules about the existence of transitions, and allows one to use transitions already proved for one SA term in reasoning about the transitions of a larger term containing it.

6.2 Statecharts Process Algebra

In [US94b] we showed that a traditional process algebra with *prefix*, *choice*, and *recursion*, need only be extended with the appropriate variant of *merge* (capturing statecharts-like broadcast communication) and a new *state refinement* operator [US93], in order to produce a statecharts-like process algebra, SPA. This improves upon the algebra SA in two ways. First, in SA there is a

term/configuration dichotomy not found in process algebras. In SPA a term **is** a state – simplifying the presentation of the semantics. Second, there is a wealth of published research relating to process algebra operators, the specification of their semantics, and their interactions with each other. Experimenting with language features in SPA is much easier than in SA where each operator must be defined and explored anew.

We propose to extend that work by bringing it into exact correspondence with the semantics presented in this paper (some of which is new). We anticipate finding that the *finite state subset* of SPA corresponds exactly to statecharts. It is well known that a sound and complete axiomatization of bisimulation exists for the finite state subset of CCS [Mil89, Mil90], and we plan to do the same for SPA.

6.3 An Implementation of State Refinement in The Concurrency Factory

We plan to implement state refinement in the Concurrency Factory [CGL+94], an ongoing project at Stony Brook and North Carolina State University aimed at putting process algebra to industrial use. This will allow us to evaluate the usefulness of state refinement as a language construct in isolation from the treatment of communication found in statecharts. If useful, this will be a significant contribution to the Concurrency Factory as a practical design and specification tool. The Concurrency Factory is a Motif/C++/VTView based tool. The implementation will include modifying the `enabled transitions` method of the `system` object, and adding to the user interface to allow the depiction of state refinement style hierarchy. An extended example will be employed to evaluate the usefulness of state refinement.

6.4 Conclusion and Future Work

The semantics of statecharts is rich and complex, and there is a wealth of proposals in the literature to extend statecharts in various ways. There are numerous issues yet to be addressed about statecharts as well as extensions to it. We will survey some of these questions in a concluding section of the proposed dissertation.

References

[BKO88] J. A. Bergstra, J. W. Klop, and E.-R. Olderog. Readies and failures in the algebra of communicating processes. *SIAM Journal of Computing*, 17:1134–1177, 1988.

[CGL+94] R. Cleaveland, J. N. Gada, P. M. Lewis, S. A. Smolka, O. Sokolsky, and S. Zhang. The concurrency factory — practical tools for specification, simulation, verification, and implementation of concurrent systems. In *Proceedings of DIMACS Workshop on Specification of Parallel Algorithms*, Princeton, NJ, 1994.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[HGdR89] C. Huizing, R. Gerth, and W. P. de Roever. Modeling statecharts behavior in a fully abstract way. In *Proc. 13th CAAP*, number 299 in Lecture Notes in Computer Science, pages 271–294. Springer Verlag, 1989.

[HP85] D. Harel and A. Pnueli. On the development of reactive systems. In *Logic and Models of Concurrent Sytems*, number 133 in NATO ASI series, pages 477–498, Berlin, 1985. Springer-Verlag.

[HPSS87] D. Harel, A. Pnueli, J. P. Schmidt, and R. Sherman. On the formal semantics of statecharts. *Proc. 2nd IEEE Symposium on Logic in Computer Science*, pages 54–64, 1987.

[HRdR92] J. J. M. Hooman, S. Ramesh, and W. P. de Roever. A compositional axiomatization of statecharts. *Theoretical Computer Science*, 101:289–335, July 1992.

[Mar91] F. Maraninchi. The Argos language: graphical representation of automata and description of reactive systems. In *IEEE Workshop on Visual Languages*, 1991.

[Mar92] F. Maraninchi. Operational and compositional semantics of synchronous automaton composition. In *Proceedings of CONCUR '92 – Third International Conference on Concurrency Theory*, 1992.

[Mil89] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[Mil90] R. Milner. A complete axiomatization for observational congruence of finite-state behaviours. *Information and Computation*, 81(2):227–247, May 1990.

[Per93] A. Peron. *Synchronous and Asynchronous Models for Statecharts*. PhD thesis, Universita di Pisa-Genova-Udine, 1993.

[Plo81] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[Pnu86] A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In et. al. de Baker, editor, *Current Trends in Concurrency*, number 224 in Lecture Notes inComputer Science, pages 510–584, 1986.

[PS91] A. Pnueli and M. Shalev. What is in a step: On the semantics of statecharts. In *Theoretical Aspects of Computer Software*, number 526 in Lecture Notes in Computer Science, pages 244–264, 1991.

[US93] A. C. Uselton and S. A. Smolka. State refinement in process algebra. In *Proceedings of the North American Process Algebra Workshop*, Ithaca, New York, August 1993. Available as TR 93-1369, Department of Computer Science, Cornell University.

[US94a] A. Uselton and S. Smolka. A compositional semantics for statecharts using labeled transition systems. Technical Report 94/1, State University of New York at Stony Brook, Stony Brook, New York, 1994.

[US94b] A. C. Uselton and S. A. Smolka. A process algebraic semantics for statecharts via state refinement. In *Proceedings of IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET)*, June 1994. To appear.

A Outline of Proposed Dissertation

1. Introduction
   (a) Overview
   (b) Motivation
   (c) Related Work
2. Statecharts Syntax
   (a) Examples
   (b) Pnueli Shalev Tuples
   (c) Statecharts Algebra
3. Statecharts Semantics
   (a) The Step Semantics
   (b) Statecharts Equivalence
4. A Compositional Semantics for Statecharts
   (a) The Semantic Domain $\text{LTS}_{>}$
   (b) A Syntax Directed Definition of Orderly Steps
5. Statecharts Process Algebra
   (a) The General Case
   (b) The Finite State Case
   (c) An Axiomatization of Bisimulation
6. State Refinement in The Concurrency Factory (TCF)
   (a) A Comparison of Statecharts and TCF
   (b) An Implementation of State Refinement in TCF
   (c) An Extended Example of the Use of State Refinement in TCF
7. Conclusion and Future Work

Table 3: The proposed dissertation outline

B On Proving Theorem 3.3: The Equivalence of the Syntaxes SA and SPS

The lemmas and theorems in this appendix support our assertion that the syntaxes SA and SPS are interchangeable. Refer to Section 3 for the translation function ϕ and to Section 2 for the

definition of $\equiv_{\mathrm{SPS}}$.

The following lemma asserts that the structure of a sub-term p' of a term p is included in the structure of p with the only modification that every name $<\vec{j}>$ in $N(p')$ is now prefixed by $<\vec{i}>$, where $p' = subterm_p(<\vec{i}>)$.

Lemma B.1 *Let p be an* SA *term. For all $<\vec{i}>\in N(p)$ with $p' = \mathrm{subterm}_p(<\vec{i}>)$:*

$$N(p') = \{<\vec{j}> \mid <\vec{i},\vec{j}>\in N(p)\}$$

and for all $<\vec{j}>\in N(p')$:

$$\begin{aligned}
\mathrm{type}_{N(p')}(<\vec{j}>) &= \mathrm{type}_{N(p)}(<\vec{i},\vec{j}>) \\
\mathrm{children}_{N(p')}(<\vec{j}>) &= \mathrm{children}_{N(p)}(<\vec{i},\vec{j}>) \\
\mathrm{default}_{N(p')}(<\vec{j}>) &= \mathrm{default}_{N(p)}(<\vec{i},\vec{j}>)
\end{aligned}$$

whenever either side is defined. In particular, note that for all $n, n' \in N(p)$

$$n \searrow\!* \; n' \Leftrightarrow n \in *n'$$

Finally,

$$\begin{aligned}
\mathcal{A}(p') &= \{(<\vec{j}>, l, <\vec{j}'>) \mid (<\vec{i},\vec{j}>, l, <\vec{i},\vec{j}'>) \in \mathcal{A}(p)\} \\
E(p') &= \bigcup_{(<\vec{i},\vec{j}>,l,<\vec{i},\vec{j}'>)\in\mathcal{A}(p)} \mathrm{events}(l) \\
r(p') &= <>
\end{aligned}$$

Proof: By structural induction on terms $p \in$ SA. The base case $p = \Box$ is trivially true since the only name is $<>$. Assume the lemma is true for terms p_i with $p = [(p_1, \ldots, p_k), \rightarrow]$ and $1 \le i \le k$, or $p = p_1 \times p_2$ and $1 \le i \le 2$. By inductive hypothesis we know, for example, that for all $<\vec{i}>\in N(p_i)$:

$$N(subterm_{p_i}(<\vec{i}>)) = \{<\vec{j}> \mid <\vec{i},\vec{j}>\in N(p_i)\} \qquad \text{IH}$$

For $n \in N(p)$ there are three cases:

- $n =<>$: the property is trivially satisfied, that is

  $$N(subterm_p(<>)) = N(p) = \{<\vec{j}> \mid <\vec{j}>\in N(p)\}$$

- $n =< i >$: By assumption $< i >\in N(p)$ so p_i is not a nested *and*-term, and

  $$N(subterm_p(< i >)) = N(p_i) = \{<\vec{j}> \mid <i,\vec{j}>\in N(p)\}$$

  is just a restatement of the definition of $N(p)$. $names(p)$ prefixes an i to the names of p_i, and $N(p)$ only leaves out the nested *and*-terms.

- $n = < i, \vec{\imath} >, < \vec{\imath} > \neq <>$: and using the definition of $N(p)$ and IH we have

$$\begin{aligned} N(subterm_p(< i, \vec{\imath} >)) &= N(subterm_{p_i}(< \vec{\imath} >)) \\ &= \{< \vec{\jmath} > \mid < \vec{\imath}, \vec{\jmath} > \in N(p_i)\} \\ &= \{< \vec{\jmath} > \mid < i, \vec{\imath}, \vec{\jmath} > \in N(p)\} \end{aligned}$$

The proof of the rest is identical. □

Syntactically equivalent statecharts have an identical structure.

Lemma B.2 *Let $p \equiv_{\mathrm{SA}} p'$ with h witnessing this fact. For all $n \in N(p)$:*

$$\mathrm{subterm}_p(n) \equiv_{\mathrm{SA}} \mathrm{subterm}_{p'}(h(n))$$

Proof: We have that $\phi(p) \equiv_{\mathrm{SPS}} \phi(p')$ and $h : N(p) \to N(p')$ is a bijection enjoying the properties in Table 1. We must show that for any $n \in N(p)$ there is a bijection $h' : N(subterm_p(n)) \to N(subterm_{p'}(h(n)))$ also enjoying the properties of Table 1.

In the following let $< \vec{\imath} > \in N(p)$ with $p_1 = subterm_p(< \vec{\imath} >)$ and $p_1' = subterm_{p'}(h(< \vec{\imath} >))$. Consider the subset of $N(p)$ prefixed by $\vec{\imath}$:

$$N_{\vec{\imath}} = \{< \vec{\jmath} > \in N(p) | < \vec{\imath} > \in * < \vec{\jmath} >\}$$

Since h is a bijection, $N_{\vec{\imath}}$ is in 1-to-1 correspondence with its image $h(N_{\vec{\imath}})$ in $N(p')$. For

$$N_{h(\vec{\imath})} = \{< \vec{\jmath} > \in N(p') | h(< \vec{\imath} >) \in * < \vec{\jmath} >\}$$

we have $h(N_{\vec{\imath}}) = N_{h(\vec{\imath})}$. This follows from Lemma B.1 and the fact that h forces a correspondence between the descendants of $< \vec{\imath} >$ and $h(< \vec{\imath} >)$. Furthermore, $N_{\vec{\imath}}$ is in 1-to-1 correspondence with $N(p_1)$, and $N_{h(\vec{\imath})}$ is in 1-to-1 correspondence with $N(p_1')$. We conclude that there is a bijection $h' : N(p_1) \to N(p_1')$, namely, for $< \vec{\imath}_1 > \in N(p_1)$ and $h(< \vec{\imath}, \vec{\imath}_1 >) = < h(< \vec{\imath} >), \vec{\jmath} >$:

$$h'(< \vec{\imath}_1 >) = < \vec{\jmath} >$$

Furthermore, the properties of Table 1 hold between $N_{\vec{\imath}}$ and $N_{h(\vec{\imath})}$ and by Lemma B.1 must hold between $N(p_1)$ and $N(p_1')$. Thus h' is the desired bijection, and $p_1 \equiv_{\mathrm{SA}} p_1'$. □

Theorem B.1 $\equiv_{\mathrm{SA}}$ *is a congruence for the operators of* SA. *That is, if $p \equiv_{\mathrm{SA}} p'$ then:*

$$\begin{aligned} [(p_1, \ldots, p_{i-1}, p, p_{i+1}, \ldots, p_k), \to] &\equiv_{\mathrm{SA}} [(p_1, \ldots, p_{i-1}, p', p_{i+1}, \ldots, p_k), \to] \quad 1 \leq i \leq k \\ p_1 \times p &\equiv_{\mathrm{SA}} p_1 \times p' \\ p \times p_2 &\equiv_{\mathrm{SA}} p' \times p_2 \end{aligned}$$

Proof: Assume $p \equiv_{\mathrm{SA}} p'$ with bijection $h : N(p) \to N(p')$ witnessing this fact. We must show that for each of the above contexts $X[\cdot]$ there is an $h' : N(X[p]) \to N(X[p'])$ enjoying the properties of Table 1.

Part 1: Let $X[\cdot] = [(p_1, \ldots, p_{i-1}, [\cdot], p_{i+1}, \ldots, p_k), \rightarrow]$, and for all $n \in N(X[p])$ define h' as follows:

$$h'(n) = \begin{cases} <> & n =<> \\ < j, \vec{\imath} > & n =< j, \vec{\imath} >, j \neq i \\ < i, h(< \vec{\imath} >) > & n =< i, \vec{\imath} > \end{cases}$$

h' is a bijection since it maps all the names $< i, \vec{\imath} >$ following h, which is a bijection; it maps all of the names not of the form $< i, \vec{\imath} >$ following the identity map, also a bijection; and there is no overlap between the two. It remains to confirm that h' enjoys the properties of Table 1.

Let $n \in N(X[p])$. There are three cases:

- $n =<>$: the only non-trivial property to verify is that

  $$\{h'(n') | n' \in children_{N(X[p])}(n)\} = children_{N(X[p'])}(h'(n))$$

  and the only interesting case is for $n' =< i >$. Since we know that $h(r(p)) = r(p') =<>$ we have that $h'(< i >) =< i >$ and the condition is satisfied.

- $n =< j, \vec{\imath} >, j \neq i$: Since h' is the identity map in this case all the properties are trivially satisfied.

- $n =< i, \vec{\imath} >$: In this case we appeal to Lemma B.1 and the fact that h enjoys the properties of Table 1.

A similar argument confirms that for $n, n' \in N(X[p])$

$$(n, l, n') \in \mathcal{A}(X[p]) \text{ iff } (h(n), l, h(n')) \in \mathcal{A}(X[p'])$$

Part 2: Let $X[\cdot] = p_1 \times [\cdot]$. Note that if $type(p) = and$ then $< 2 > \notin N(X[p])$ and $< 2 > \notin N(X[p'])$. For all $n \in N(X[p])$ define h' as follows:

$$h'(n) = \begin{cases} <> & n =<> \\ < 1, \vec{\imath} > & n =< 1, \vec{\imath} > \\ < 2, h(< \vec{\imath} >) > & n =< 2, \vec{\imath} > \end{cases}$$

The reasoning showing that h' is a bijection is exactly as before. The only reasoning that differs in showing that h' enjoys the properties of Table 1 is that we must confirm that

$$\{h'(n') | n' \in children_{N(X[p])}(<>)\} = children_{N(X[p'])}(h'(<>))$$

Whether $type(p_1) = and$ or not it is trivially verified that the same children of the root result from p_1. We know that $type_{N(p)}(<>) = type_{N(p')}(<>)$, so $< 2 > \in children_{N(X[p])}(<>)$ if and only if $< 2 > \in children_{N(X[p'])}(<>)$. Similarly, if $type_{N(p)}(<>) = type_{N(p')}(<>) = and$ then the bijection h assures us that

$$\{h(n') | n' \in children_{N(p)}(<>)\} = children_{N(p')}(h(<>))$$

and by Lemma B.1 both p and p' contribute corresponding children to the root of $X[\cdot]$.

Part 3: In the case that $X[\cdot] = [\cdot] \times p_2$ the reasoning is symmetric to Part 2.

Thus $\equiv_{\mathrm{SA}}$ is preserved by all the operators of SA, i.e. $\equiv_{\mathrm{SA}}$ is a congruence for the operators of SA. Together with Lemma B.2 the above also gives us that

$$p \equiv_{\mathrm{SA}} p' \Rightarrow \forall X[\cdot].\ X[p] \equiv_{\mathrm{SA}} X[p']$$

That is, syntactic equivalence is substitutive for all SA-contexts. □

Theorem B.2 $\mathrm{SA}_{\equiv}$ *gives a sound and complete axiomatization for* $\equiv_{\mathrm{SA}}$.

Proof: **Soundness:** The proof of soundness proceeds exactly as in Theorem 3.1. For $p = p_1 \times p_2$ and $p' = p_2 \times p_1$ the bijection $h : N(p) \rightarrow N(p')$ is:

$$h(n) = \begin{cases} <> & n = <> \\ < 2, \vec{\imath} > & n = < 1, \vec{\imath} > \\ < 1, \vec{\imath} > & n = < 2, \vec{\imath} > \end{cases}$$

and the properties of Table 1 follow from Lemma B.1. For $p = p_1 \times (p_2 \times p_3)$ and $p' = (p_1 \times p_2) \times p_3$ h is:

$$h(n) = \begin{cases} <> & n = <> \\ < 1, 1, \vec{\imath} > & n = < 1, \vec{\imath} > \\ < 1, 2, \vec{\imath} > & n = < 2, 1, \vec{\imath} > \\ < 2, \vec{\imath} > & n = < 2, 2, \vec{\imath} > \end{cases}$$

Note that $< 2 > \notin N(p)$ and $< 1 > \notin N(p')$. For $p = [(p_1, \ldots, p_i, \ldots, p_j, \ldots p_k), \rightarrow]$ and $p' = [(p_1, \ldots, p_j, \ldots, p_i, \ldots p_k), \rightarrow']$ $(1 < i, j \leq k)$ and $\rightarrow' = \rightarrow [i/j, j/i]$, h is:

$$h(n) = \begin{cases} <> & n = <> \\ < i_0, \vec{\imath} > & n = < i_0, \vec{\imath} >, i_0 \neq i, j \\ < j, \vec{\imath} > & n = < i, \vec{\imath} > \\ < i, \vec{\imath} > & n = < j, \vec{\imath} > \end{cases}$$

and we verify that

$$(i_1, l, i_2) \in \rightarrow \text{ iff } (h'(i_1), l, h'(i_2)) \in \rightarrow'$$

where we use h' such that $< h'(i) > = h(< i >)$ for $1 \leq i \leq k$.

Completeness: Let $p \equiv_{\mathrm{SA}} p'$ with h witnessing this fact. By structural induction on p we show that using $\mathrm{SA}_{\equiv}$ as a set of rewriting rules we can produce p' from p. The base case $p = \square$ is trivial, since $p' = \square$ needs no applications of $\mathrm{SA}_{\equiv}$ at all.

***or*-terms:** Let $p = [(p_1, \ldots, p_k), \rightarrow]$. Table 1 assures us that $type_{N(p)}(<>) = type_{N(p')}(<>)$, and h is a one to one correspondence between $children_{N(p)}(<>)$ and $children_{N(p')}(<>)$, so $p' = [(p'_1, \ldots, p'_k), \rightarrow')$. Let $p'_{h(i)} = subterm_{p'}(h(< i >))$ for $1 \leq i \leq k$. Lemma B.2 gives us that

$$p_i \equiv_{\mathrm{SA}} p'_{h(i)}$$

for $1 \leq i \leq k$, and our inductive hypothesis is that each $p'_{h(i)}$ can be produced from p_i using $\mathrm{SA}_{\equiv}$. We know $h(< 1 >) = < 1 >$ since h must preserve defaults. For each $i \neq 1$ if $h(< i >) \neq < i >$

then an application of SA_3 produces a new term $q \equiv_{\mathrm{SA}} p'$ witnessed by h' with $h'(< i >) =< i >$. Let $p_{h(i)} = subterm_p(h(< i >))$ for $1 \leq i \leq k$. After no more than $k-2$ applications of SA_3 a term $q' = [(p_{h(1)}, \ldots, p_{h(<k>)}), \rightarrow']$ with $q' \equiv_{\mathrm{SA}} p$ results. Theorem 3.1 allows us to substitute $subterm_{p'}(h(< i >))$ for each $p_{h(i)}$ in q' and p' is the result.

***and*-terms:** Let $p = p_1 \times p_2$. Again we have that $p' = p'_1 \times p'_2$. Let $k = |children_{N(p)}(<>)|$. and for all $< \vec{\imath} > \in children_{N(p)}(<>)$

$$subterm_p(< \vec{\imath} >) \equiv_{\mathrm{SA}} subterm_{p'}(h(< \vec{\imath} >))$$

and our inductive hypothesis is that each $subterm_{p'}(h(< \vec{\imath} >))$ can be produced from $subterm_p(< \vec{\imath} >)$ using $\mathrm{SA}_{\equiv}$. In this case applications of both SA_1 and SA_2 may be necessary to produce a term $q' \equiv_{\mathrm{SA}} p$ witnessed by h' such that h' is the identity map on the children of $<>$ Order the children of $<>$ in $N(p)$ and $N(p')$ lexicographically ($< 1, \vec{\imath} >$ before $< 2, \vec{\jmath} >$ and so on), and number the sub-terms $p_1, \ldots, p_k$ and $p'_1, \ldots, p'_k$, respectively. Let h_0 be the permutation of $1, \ldots, k$ that corresponds to h on the children of p and p'.

A finite number of applications of SA_1 and SA_2 will produce a term $q \equiv_{\mathrm{SA}} p$ in which the children of q are in the same order as in p'. To see that this is so consider $q_1 = p_{h_0^{-1}(1)}$ – the term that should end up ordered first – and let $q_1 = subterm_p(< \vec{\imath} >)$. If $< \vec{\imath} >=< \vec{\jmath}, 2 >$ then apply SA_1. Then if $< \vec{\imath} >=< \vec{\jmath}, j, 1 >$ then apply SA_2. Now $q_1 = subterm_{p^?}(< \vec{\jmath}, 1 >)$ where $p^? \equiv_{\mathrm{SA}} p$ is the term that results. After no more than k applications of the above the result is $q_1 = subterm_{p^?}(< 1 >)$ and the same procedure can be applied to q_2 in $subterm_{p^?}(< 2 >)$ and so on (no more than k^2 steps total). It then requires no more than $k-1$ applications of SA_2 to produce $q' \equiv_{\mathrm{SA}} p'$ (sub-terms $q'_1, \ldots, q'_k$) with bijection h' the identity map on the children of the root, i.e. the parenthesizations and order are the same in each up to the first occurrences of *or*-terms. The substitution of the p'_i for the q'_i results in p'.

Thus the equations $\mathrm{SA}_{\equiv}$ are sufficient to prove $p \equiv_{\mathrm{SA}} p'$ any time it is true. □

Theorem B.3 *$\phi_{[\cdot]}$ is a bijection.*

Proof: That $\phi_{[\cdot]}$ in injective follows directly from its definition. Two equivalence classes of SA terms are mapped to the same equivalence class of SPS tuples exactly because the tuples are $\equiv_{\mathrm{SPS}}$, therefore all the members of either equivalence class are $\equiv_{\mathrm{SA}}$ and the two equivalence classes are the same.

That $\phi_{[\cdot]}$ is surjective follows from the fact that translation ϕ can easily be reversed. Let $S = (\mathrm{E}, N, r, \mathcal{A})$ and define ϕ^{-1} as follows:

- For each $n \in N$ order the set $children_N(n)$ ($k = |children_N(n)|$) according to some bijection $ord_n : children_N(n) \rightarrow \{1, \ldots, k\}$ such that if $type_N(n) = or$ then $ord_n(default_N(n)) = 1$ (if it exists).
- For $type_N(n) = or$ with $|children_N(n)| = k$:

  $$\phi^{-1}(n) = [(\phi^{-1}(ord_n^{-1}(1)), \ldots, \phi^{-1}(ord_n^{-1}(k))), \rightarrow_n]$$

  with

  $$A_n = \{(ord_n(n_1), l, ord_n(n_2)) | n_1, n_2 \in children_N(n), (n_1, l, n_2) \in \mathcal{A}\}$$

- For $type_N(n) = and$ with $|children_N(n)| = k$:

$$\phi^{-1}(n) = \phi^{-1}(ord_n^{-1}(1)) \times (\ldots \times (\phi^{-1}(ord_n^{-1}(k-1)) \times \phi^{-1}(ord_n^{-1}(k)))\ldots)$$

That $\phi(\phi^{-1}(S)) \equiv_{\mathrm{SPS}} S$ is easily seen, and therefore there exists an equivalence class $P \in \mathrm{SA}/\equiv_{\mathrm{SA}}$ (namely $[\phi^{-1}(S)]_{\equiv_{\mathrm{SA}}}$) such that $\phi_{[\cdot]}(P) = [S]_{\equiv_{\mathrm{SPS}}}$. □

C On Proving Lemma 4.2: Enabledness is Preserved by $=_{\mathrm{SC}}$

Let p be an SA term in configuration c_p and $X[\cdot]$ a context with $c \in \mathcal{C}(X[p])$ and c_p is part of c, i.e. for $subterm_{X[p]}(<\vec{\imath}>) = p$, $<\vec{\imath}, c_p> \subseteq c$. For $A \subseteq \mathcal{A}(X[p])$ let A_X the subset of A outside p, that is

$$A_X = \{\alpha \in A | <\vec{\imath}> \notin *source(\alpha)\}$$

and let A_p be the subset of A in p, i.e.

$$A_p = \{(<\vec{\jmath}>, l, <\vec{\jmath}'>) | (<\vec{\imath}, \vec{\jmath}>, l, <\vec{\imath}, \vec{\jmath}'>) \in A\}$$

The following lemma establishes that if a term has an admissible arrow set A that each subterm contributing to A has a corresponding admissible arrow set in isolation from the surrounding context. This is the first stage of the scheme depicted in Figure 3.

Lemma C.1 *Let $p, X[\cdot], c, c_p, A, A_p,$ and A_X be as given above. If $A \in \mathrm{admit}_{X[p]}(c, \xi)$ then $A_p \in \mathrm{admit}_p(c_p, \xi \cup \mathrm{actions}(A_X))$*[8].

Proof: We show the lemma is true for contexts

$$[(p_1, \ldots, p_{i-1}, [\cdot], p_{p+1}, \ldots, p_k), \rightarrow] \quad 1 \leq i \leq k$$
$$p_1 \times [\cdot]$$
$$[\cdot] \times p_2$$

and an induction on the structure of $X[\cdot]$ gives the desired result.

***or*-terms:** In this case either A_X is empty or A_p is empty, since arrows outside of p are not orthogonal to arrows inside of p. $A_p \in admit_p(c_p, \xi_i)$, since the only difference between A and A_p is the leading $<\vec{\imath}>$ provided by the syntax.

***and*-terms:** Let $X[\cdot] = p_1 \times [\cdot]$ (the other case is symmetric) with $c \in \mathcal{C}(X[p])$. $c =< 1, c_1 > \cup < 2, c_p >$ for $c_1 \in \mathcal{C}(p_1)$ and $c_p \in \mathcal{C}(p)$. Let $A \in admit_{p_1 \times p}(c, \xi)$ for some ξ, and partition A into $A_1 \subseteq \mathcal{A}(p_1)$ and $A_p \subseteq \mathcal{A}(p)$. Let $\xi' = \xi \cup actions(A_1)$ and observe the following:

- To show $A_p \subseteq enabled_p(c_p, \xi', A_p)$: Verify that
  - $A_p \subseteq relevant_p(c_p)$

[8] Note that the reverse is not true. Consider a pair of arrows α_1 and α_2 labeled a and $a\overline{b}$, respectively. $\{\alpha_1, \alpha_2\}$ may be admissible on input a, but in a context with a third arrow α_3 labeled a/b there is no input that will enable α_1 and α_2 together.

- $A_p \subseteq consistent_p(A_p)$
- $A_p \subseteq triggered_p(\xi' \cup actions(A_p))$

- To show $enabled_p(c_p, \xi', A_p) \subseteq A_p$: Let $\alpha \in enabled_p(c_p, \xi', A_p)$ and observe that
  - $\alpha \in relevant_p(c_p)$
  - $\alpha \in consistent_p(A_p)$
  - $\alpha \in triggered_p(\xi' \cup actions(A_p))$

  So $\alpha \in enabled_{p_1 \times p}(c, \xi, A)$ thus $\alpha \in A_p$.
- To show A_p is inseparable: By contradiction. Let $A'_p \subset A_p$ such that

$$enabled_p(c_p, \xi', A'_p) \cap (A_p - A'_p) = \emptyset$$

  and consider $A' = A_1 \cup A'_p \subset A$. For all $\alpha \in A - A'$ we have that $\alpha \notin enabled_{p_1 \times p}(c, \xi, A')$ so

$$enabled_{p_1 \times p}(c, \xi, A') \cap (A - A') = \emptyset$$

  and A is separable – contradicting $A \in admit_{p_1 \times p}(c, \xi)$.

□

Let $p, p' \in$ SA with $p =_{\mathrm{SC}} p'$, witnessed by $h : \mathcal{C}(p) \rightarrow \mathcal{C}(p')$, and let $X[\cdot]$ be some context such that $subterm_{X[p]}(< \vec{\imath} >) = p$. Define the function $h_X : \mathcal{C}(X[p]) \rightarrow \mathcal{C}(X[p'])$ such that for $c_p \in \mathcal{C}(p)$

$$h_X(c) = \begin{cases} c & < \vec{\imath}, c_p > \not\subseteq c \\ (c - < \vec{\imath}, c_p >) \cup < \vec{\imath}, h(c_p) > & < \vec{\imath}, c_p > \subseteq c \end{cases}$$

Lemma C.2 *Let $p, p', h, c_p, X[\cdot], < \vec{\imath} >$, and h_X be as above, and let A, A_X, and A_p be as in Lemma C.1. Further, let $A_{p'} \in \mathrm{admit}_{p'}(h(c_p))$ be the arrow set in p' corresponding to that in p via the bijection h, and let*

$$A' = A_X \cup \{(< \vec{\imath}, \vec{\jmath} >, l, < \vec{\imath}, \vec{\jmath}' >) | (< \vec{\jmath} >, l, < \vec{\jmath}' >) \in A_{p'}\}$$

h_X is a bijection with $h_X(\Delta(X[p]) = \Delta(X[p'])$, such that

$$h_X(\mathrm{next}(c, A)) = \mathrm{next}(h_X(c), A')$$

Proof: Let $p =_{\mathrm{SC}} p'$ with bijection h witnessing this fact.

or-**terms:** Let $X[\cdot] = [(p_1, \ldots, p_{i-1}, [\cdot], p_{i+1}, \ldots, p_k), \rightarrow]$. So h_X is

$$h_X(< j, c_j >) = \begin{cases} < j, c_j > & j \neq i \\ < j, h(c_j) > & j = i \end{cases}$$

and the result follows from the fact that h is a bijection such that $h(\Delta(p)) = \Delta(p')$ and from the definitions of $\mathcal{C}$ and h_X.

and-**terms:** Let $X[\cdot] = p_1 \times [\cdot]$. So h_X is the bijection

$$h_X(< 1, c_1 > \cup < 2, c_p >) = < 1, c_1 > \cup < 2, h(c_p) >$$

Again, the result follows from the fact that h is a bijection such that $h(\Delta(p)) = \Delta(p')$ and the definitions of $\mathcal{C}$ and h_X. The case $X[\cdot] = [\cdot] \times p_2$ is symmetric.

Induction on the structure of $X[\cdot]$ gives us the result. □

Lemma C.3 $=_{\mathrm{SC}}$ *is preserved by the* or *operator. That is, for all* $k \geq 0$ *and all* k*-place* or *terms* p *and* p'

$$p =_{\mathrm{SC}} p'$$

if and only if for all i, $1 \leq i \leq k$

$$[(p_1, \ldots, p_{i-1}, p, p_{i+1}, \ldots, p_k), \rightarrow] =_{\mathrm{SC}} [(p_1, \ldots, p_{i-1}, p', p_{i+1}, \ldots, p_k), \rightarrow]$$

Proof: We present the details for $\Rightarrow$. $\Leftarrow$ is the same. Let $p =_{\mathrm{SC}} p'$ with bijection $h : \mathcal{C}(p) \rightarrow \mathcal{C}(p')$ witnessing this fact, and let $X[\cdot] = [(p_1, \ldots, p_{i-1}, [\cdot], p_{i+1}, \ldots, p_k), \rightarrow]$. Define the bijection $h_X : \mathcal{C}(X[p]) \rightarrow \mathcal{C}(X[p'])$ as above. To see that for $c, c' \in \mathcal{C}(X[p])$

$$c \xrightarrow{\xi_i/\xi_o}_{X[p]} c' \text{ iff } h_X(c) \xrightarrow{\xi_i/\xi_o}_{X[p]} h_X(c')$$

observe that $c \xrightarrow{\xi_i/\xi_o}_{X[p]} c'$ is one of the three cases

$$\begin{array}{lll} < j, c_j > & \xrightarrow{\xi_i/\xi_o}_{X[p]} & < j, c'_j > \quad j \neq i \\ < j, c_j > & \xrightarrow{\xi_i/\xi_o}_{X[p]} & < j', c_{j'} > \\ < i, c_i > & \xrightarrow{\xi_i/\xi_o}_{X[p]} & < i, c'_i > \end{array}$$

In the first case none of the arrows of p or p' is relevant to the configuration, so the same arrow set leads to the same step in $X[p']$. In the second case even if j or j' equals i none of the arrows of p or p' is orthogonal to arrow (j, l, j'), so again the same step results. In the third case none of the arrows outside of p is relevant, so the arrow set leading to the step must be be matched by an admissible arrow set of p' giving the same step. The proof that $h'(c) \xrightarrow{\xi_i/\xi_o}_{X[p']} h'(c')$ implies $c \xrightarrow{\xi_i/\xi_o}_{X[p]} c'$ is identical. □

If $=_{\mathrm{SC}}$ were preserved by the *and* operator then $=_{\mathrm{SC}}$ would be a congruence, but it is not. The following lemma states that, but for inseparability, $=_{\mathrm{SC}}$ would be preserved by the *and* operator.

In the following lemma let $p, p' \in \mathrm{SA}$ with $p =_{\mathrm{SC}} p'$, witnessed by $h : \mathcal{C}(p) \rightarrow \mathcal{C}(p')$, and $c_p \in \mathcal{C}(p)$. Let $X[\cdot] = p_1 \times [\cdot]$ with $p_1 \in \mathrm{SA}$ and h_X defined as above, and let $c, c' \in \mathcal{C}(X[p])$ with $< 2, c_p > \subseteq c$. Let $A \in \mathit{admit}_{X[p]}(c, \xi_i)$, with $\xi_o = \mathit{actions}(A)$ and $c' = \mathit{next}(c, A)$ so that

$$c \xrightarrow{\xi_i/\xi_o}_{X[p]} c'$$

Finally, let $A_1 = \{\alpha \in A | < 2 > \in * \mathit{source}(\alpha)\}$, and let $A_p \in \mathit{admit}_p(c_p)$, $A_{p'} \in \mathit{admit}_{p'}(h(c_p))$, A_X, and A' be as bofore.

Lemma C.4 *If* A' *is inseparable for* $h_X(c)$ *and* ξ_i *then*

$$h_X(c) \xrightarrow{\xi_i/\xi_o}_{X[p']} h_X(c')$$

Proof: The general proof strategy is summarized in Figure 3. The definition of $\mathcal{C}$ gives us that $c =< 1, c_1 > \cup < 2, c_p >$ and $c' =< 1, c'_1 > \cup < 2, c'_p >$ where $c_1, c'_1 \in \mathcal{C}(p_1)$. Lemma C.1 assures us that $A_1 \in admit_{p_1}(c_1, \xi_i \cup actions(A_p))$ and $A_p \in admit_p(c_p, \xi_i \cup actions(A_1))$. Let $\xi'_i = \xi_i \cup actions(A_1)$ and $\xi'_o = actions(A_p)$. We have

$$c_p \xrightarrow{\xi'_i/\xi'_o}_p c'_p$$

resulting from A_p. This is step 1 in the figure.

By assumption $p =_{\mathrm{SC}} p'$, so there must also be a step

$$h(c_p) \xrightarrow{\xi'_i/\xi'_o}_{p'} h(c'_p)$$

for some arrow set $A_{p'} \in admit(h(c'_p), \xi'_i)$, $\xi'_o = actions(A_{p'})$, and $h(c'_p) = next_{p'}(h(c_p), A_{p'})$. This is step 2 in the figure.

It only remains to show that for $A' = A_1 \cup A_{p'}$ we have

$$A' = enabled_{X[p']}(h_X(c), \xi_i, A')$$

$\xi_o = actions(A')$, and $h_X(c') = next_{X[p']}(h_X(c), A')$. Observe that

$$\begin{aligned} h_X(c') &= h_X(< 1, c'_1 > \cup < 2, c'_p >) \\ &= < 1, c'_1 > \cup < 2, h(c'_p) > \\ &= < 1, next_{p_1}(c_1, A_1) > \cup < 2, next_p(c_p, A_p) > \end{aligned}$$

so $next_{X[p']}(c, A')$ is the correct configuration. Similarly,

$$\begin{aligned} actions(A') &= actions(A_1) \cup actions(A_{p'}) \\ &= actions(A_1) \cup \xi'_o \\ &= actions(A_1) \cup actions(A_p) \\ &= actions(A) \\ &= \xi_o \end{aligned}$$

So the configuration and generated action are the correct values. Now we must show:

- $A' \subseteq enabled_{X[p']}(h_X(c), \xi_i, A')$
- $enabled_{X[p']}(h_X(c), \xi_i, A') \subseteq A'$

To show $A' \subseteq enabled_{X[p']}(h_X(c), \xi_i, A')$ we must show

- $A' \subseteq relevant_{X[p']}(h_X(c))$
- $A' \subseteq consistent_{X[p']}(A')$
- $A' \subseteq triggered(\xi_i \cup actions(A'))$

Table 4: $enabled_{p_A}(DF, \xi_i, A)$ and $enabled_{p_{A'}}(B', \xi_i, A)$

$A' = A_1 \cup A_{p'}$ and we already know $A_1 \subseteq relevant_{p_1}(c_1)$ and $A_{p'} \subseteq relevant_{p'}(h(c'_p))$ so $A' \subseteq relevant_{X[p']}(c_1 \cup h(c'_p))$. Similarly, $A_1 \subseteq consistent_{p_1}(A_1)$ and $A_{p'} \subseteq consistent_{p'}(A_{p'})$. p_1 and p' are in parallel, so all the arrows of A_1 are orthogonal to all the arrows of $A_{p'}$ and vice-versa, and $A_1 \cup A_{p'} \subseteq consistent_{X[p']}(A_1 \cup A_{p'})$. Finally, $A_1 \subseteq triggered_{p_1}((\xi_i \cup actions(A_{p'})) \cup actions(A_1))$ and $A_{p'} \subseteq triggered_{p'}((\xi_i \cup actions(A_1)) \cup actions(A_{p'}))$, so $A_1 \cup A_{p'} \subseteq triggered_{X[p']}(\xi_i \cup actions(A_1 \cup A_{p'}))$. Thus $A' \subseteq enabled_{X[p']}(h_X(c), \xi_i, A')$.

To show that $enabled_{X[p']}(h_X(c), \xi_i, A') \subseteq A'$ consider some $\alpha \in enabled_{X[p']}(h_X(c), \xi_i, A')$. Let $\alpha \in \mathcal{A}(p_1)$. $\alpha \in relevant_{X[p']}(h_X(c))$ so $\alpha \in relevant_{p_1}(c_1)$, $\alpha \in consistent_{X[p']}(A')$ so $\alpha \in consistent_{p_1}(A_1)$, and $\alpha \in triggered(\xi_i \cup actions(A_1 \cup A_{p'}))$ so $\alpha \in triggered((\xi_i \cup actions(A_{p'})) \cup actions(A_1))$. Therefore $\alpha \in enabled_{p_1}(c_1, (\xi_i \cup actions(A_{p'})), A_1)$. But A_1 is a solution of $A_1 = enabled(A_1)$ so $\alpha \in A_1$ and $\alpha \in A'$. Now let $\alpha \in \mathcal{A}(p')$. By the same reasoning we conclude $\alpha \in A'$. So $enabled_{X[p']}(h_X(c), \xi_i, A') \subseteq A'$ and we have that $A' = enabled_{X[p']}(h_X(c), \xi_i, A')$.

If A' is inseparable for $h_X(c)$ and ξ_i then $A' \in admit_{X[p']}(h_X(c), \xi_i)$ and

$$h_X(c) \xrightarrow{\xi_i/\xi_o}_{X[p']} h_X(c')$$

Which is step 3 in the figure. □

D An Example and Demonstration of Remark 4.1

We have claimed that the example of Figure 2 demonstrates that $=_{\mathrm{SC}}$ is not a congruence. This section shows that $=_{\mathrm{SC}}$ fails to be preserved in context $X[\cdot]$ because the (only) solution to

$$A = enabled_{X[p_{A'}]}(\Delta(p_{A'}), \{a\}, A)$$

is separable.

Statechart A of Figure 2 is represented by the SA term

$$p_A = [(\Box, \Box), \{(1, a/b, 2)\}] \times [(\Box, \Box), \{(1, c/d, 2)\}]$$

Statechart A' is

$$p_{A'} = [(\Box, \Box, \Box, \Box), \{(1, a\overline{c}/b, 2), (1, \overline{a}c/d, 3), (1, ac/bd, 4), (2, c/d, 4), (3, a/b, 4)\}]$$

Table 5: $enabled_{X[p_A]}(YDF, \xi_i, A)$

Table 4 shows the values of $enabled_{p_A}(\Delta(p_A), \xi_i, A)$ and $enabled_{p_{A'}}(\Delta(p_{A'}), \xi_i, A)$ for various values of ξ_i and A (we don't list inputs b and d just to keep the table shorter). A solution to $A = enabled(A)$ is indicated in the table by a box around the solution's entry.

Context $X[\cdot]$ is

$$X[\cdot] = [(\Box, \Box), \{(1, b/c, 2)\}] \times [\cdot]$$

For convenience we designate the configurations of $X[p_A]$ and $X[p_{A'}]$ (and give the mapping $h : \mathcal{C}(p_A) \rightarrow \mathcal{C}(p_{A'})$) as follows:

	$\mathcal{C}(X[p_A])$	$h_X(\mathcal{C}(X[p_A])) = \mathcal{C}(X[p_{A'}]))$	
YDF	$\{<1,1>,<2,1,1>,<2,2,1>\}$	$\{<1,1>,<2,1>\}$	YB'
YDG	$\{<1,1>,<2,1,1>,<2,2,2>\}$	$\{<1,1>,<2,2>\}$	YC'
YEF	$\{<1,1>,<2,1,2>,<2,2,1>\}$	$\{<1,1>,<2,3>\}$	YD'
YEG	$\{<1,1>,<2,1,2>,<2,2,2>\}$	$\{<1,1>,<2,4>\}$	YE'
ZDF	$\{<1,2>,<2,1,1>,<2,2,1>\}$	$\{<1,2>,<2,1>\}$	ZB'
ZDG	$\{<1,2>,<2,1,1>,<2,2,2>\}$	$\{<1,2>,<2,2>\}$	ZC'
ZEF	$\{<1,2>,<2,1,2>,<2,2,1>\}$	$\{<1,2>,<2,3>\}$	ZD'
ZEG	$\{<1,2>,<2,1,2>,<2,2,2>\}$	$\{<1,2>,<2,4>\}$	ZE'

Table 6: $enabled_{X[p_{A'}]}(YB', \xi_i, A)$

In configurations YDF and YB' the relevant arrows are

$$relevant_{X[p_A]}(YDF) = \left\{ \begin{array}{lcl} \alpha_1 & : & (D, a/b, E) \\ \alpha_2 & : & (F, c/d, G) \\ \alpha_3 & : & (Y, b/c, Z) \end{array} \right\}$$

$$relevant_{X[p_{A'}]}(YB') = \left\{ \begin{array}{lcl} \alpha_4 & : & (B', a\overline{c}/b, C') \\ \alpha_5 & : & (B', \overline{a}c/d, D') \\ \alpha_6 & : & (B', ac/bd, E') \\ \alpha_7 & : & (Y, b/c, Z) \end{array} \right\}$$

For any choice of $A \subseteq \mathcal{A}(X[p_A])$ we have $consistent_{X[p_A]}(A) = \{\alpha_1, \alpha_2, \alpha_3\}$. Table 5 lists the values of $enabled_{X[p_A]}(YDF, \xi_i, A)$ for each choice of $\xi_i \subseteq \{a, b, c\}$ (we ignore d in the input to keep the table shorter) and $A \subseteq relevant_{X[p_a]}(YDF)$. In $X[p_{A'}]$ the sets A for which $A \subseteq relevant_{X[p_{A'}]}(YB') \cap consistent_{X[p_{A'}]}(A)$ are $A \in \{\emptyset, \{\alpha_4\}, \{\alpha_5\}, \{\alpha_6\}, \{\alpha_7\}, \{\alpha_4, \alpha_7\}, \{\alpha_5, \alpha_7\}, \{\alpha_6, \alpha_7\}\}$. The values of $enabled_{X[p_{A'}]}(YB', \xi, A)$ are in Table 6.

Examine the step

$$YDF \xrightarrow{a/bcd}_{X[p_A]} ZEG$$

of $X[p_A]$. For $\xi_i' = a \cup actions(\{\alpha_3\}) = \{a, c\}$ we observe that $A_{1,2} = \{\alpha_1, \alpha_2\}$ is a solution to $A = enabled_{p_A}(EG, \xi_i', A)$ and is inseparable. Since $p_A =_{\mathrm{SC}} p_{A'}$ there is a step from $h(DF)$ to $h(EG)$ on input ac producing bd as output, namely

$$YB' \xrightarrow{ac/bd}_{p'} ZE'$$

from arrow set $\{\alpha_6\}$. We observe further that $A_{6,7} = \{\alpha_6, \alpha_7\}$ is a solution to

$$A = enabled_{X[p_{A'}]}(YB', \{a\}, A)$$

as reflected in the table. The circled entry in Table 6 is a solution to

$$enabled_{X[p_{A'}]}(YDF, \{a\}, A') \cap (A - A') = \emptyset$$

with $A = A_{6,7}$ and $A' = \emptyset$ exhibiting that $A_{6,7}$ is separable. There is a *causal loop* between α_6 and α_7 and one of b or c must be in the input for $\{\alpha_6, \alpha_7\}$ to be inseparable[9].

E On Proving Theorem 5.1: $\simeq$ is a Congruence

The following lemma demostrates the relationship between semantic words $\mathcal{W}(p)$ and semantic labels $\Xi(p)$ and states that well-triggeredness is equivalent to inseparability. This result is central to our proof that $\simeq$ is a congruence.

Lemma E.1 *Let p be some* SA *term in configuration c, with $A = \text{enabled}(c, \xi_i, A)$ and $\xi_o = \text{actions}(A)$*

1. $\text{trigger}(\psi_A(\xi_i/\xi_o)) = \xi_i$
2. $\text{action}(\psi_A(\xi_i/\xi_o)) = \xi_o - \xi_i$
3. *A is inseparable if and only if $\psi_A(\xi_i/\xi_o)$ is well-triggered.*

Proof: Let $w = \psi_A(\xi_i/\xi_o)$. For part 1 observe that if λ is maximal then $\lambda \in \mathit{trigger}(\alpha)$ for some $\alpha \in A$ and if $\lambda \notin \xi_i$ then $\alpha \notin \mathit{triggered}(\xi_i \cup \mathit{actions}(A))$ contradicting $A = \mathit{enabled}(c, \xi_i, A)$. Conversely, if $\lambda \in \xi_i$ then λ is maximal by the definition of $R_A^{\xi_i}$. For part 2 observe that every $\pi \in \xi_o - \xi_i$ is in the generated action of some arrow, so it is not maximal unless it is in ξ_i; in which case it must be maximal by the definition of $R_A^{\xi_i}$.

For part 3 we show (by contradiction) that if A is inseparable then $\psi_A(\xi_i/\xi_o)$ is well-triggered. Let $w = \psi_A(\xi_i/\xi_o)$ and assume that w is not well-triggered. By assumption there is at least one triggerless literal $\lambda \in \xi_o - \xi_i$, and there is at least one $\lambda' \in w$. $\lambda' R_A^{\xi_i} \lambda$. If λ is triggerless then $\forall \lambda' \in w$. $\lambda' R_A^{\xi_i} \lambda$. λ' is triggerless as well. Since w is finite there cannot be an infinite chain of distinct triggerless λ_i:

$$\cdots \lambda_{i+1} R_A^{\xi_i} \lambda_i R_A^{\xi_i} \cdots R_A^{\xi_i} \lambda_1 = \lambda$$

It must be the case that for some $i \neq j$, $\lambda_i = \lambda_j$. That is, we have a causal loop in $>_w$. Denote this loop

$$\lambda_1 R_A^{\xi_i} \lambda_2 R_A^{\xi_i} \cdots R_A^{\xi_i} \lambda_n R_A^{\xi_i} \lambda_1$$

Wolog assume each λ_i, $1 \leq i \leq n$ is distinct, i.e. we have a minimal loop. Since we defined statecharts (syntactic) labels such that the action and trigger are disjoint we know the $\lambda_i \neq \lambda_{(i+1) \bmod n}$. For each λ_i there is a $\alpha_i \in A$ with $\lambda_i \in \mathit{trigger}(\alpha_i)$ (the α_i are not necessarily all distinct, but there are at least two such α_i). Let $A_L = \{\alpha \in A | \exists \lambda \in \mathit{trigger}(\alpha).\ \lambda \text{ is triggerless}\}$, and let $A' = A - A_L$.

[9] Indeed, there is no inseparable solution to $A = \mathit{enabled}_{X[p_{A'}]}(YB', \{a\}, A)$. We could set $\mathit{admit}(c, \{a\}) = \{\emptyset\}$ in such a situation, since statecharts should specify *reactive* systems, i.e. there should be a defined response (even if it is the null response) to every input, (c.f. the definition of *admit* and the footnote on Page 12.)

Claim:

$$triggered(\xi_i \cup generated(A')) \cap A_L = \emptyset$$

Proof of claim: Let $\xi_i' = \xi_i \cup actions(A')$ and observe that if $\exists \alpha \in A_L$ such that $\alpha \in triggered(\xi_i')$ then $\forall \pi \in trigger(\alpha).\ \pi \in \xi_i'$ and $\forall \overline{\pi} \in trigger(\alpha).\ \pi \notin \xi_i'$, and $A' \subseteq A$ with $A \subseteq triggered(\xi_i \cup actions(A))$. But by assumption no $\pi \in \xi_i'$ is triggerless therefore no $\pi \in trigger(\alpha)$ is triggerless and $\alpha \notin A_L$, a contradiction, so the claim is true.

Now we have

$$enabled(A') \cap (A - A') = \emptyset$$

and A is separable; contradicting the assumption that $A \in admit(c)$. We conclude that $\psi_A(\xi_i/\xi_o)$ is well-triggered.

If w is well-triggered then for each $\pi \in action(w)$ there is some sequence $\{\alpha_1, \ldots, \alpha_n\}$ of arrows each triggered by the last and with $\alpha_1 \in triggered(\xi_i)$ and $\pi \in action(\alpha_n)$. If there were a set A_s such that

$$enabled(A - A_s) \cap A_s = \emptyset$$

then a $\alpha \in \mathcal{A}_s$ with $action(\alpha) = \emptyset$ implies the presence of a $\alpha' \in A_s$ to trigger it. But for any $\alpha \in A$ with $action(\alpha) \neq \emptyset$ each $\pi \in action(\alpha')$ has a the above sequence of triggering arrows, so there will always be some $\alpha \in enabled(A - A_s) \cap A_s$. □

Theorem E.1 $\simeq$ *is the largest* $=_{\mathrm{SC}}$ *respecting congruence.*

Proof: Let $\sim$ be the largest congruence contained in $=_{\mathrm{SC}}$[10], defined by

$$p \sim p' \quad \Leftrightarrow \quad \forall X[\cdot].\ X[p] =_{\mathrm{SC}} X[p']$$

First we must show

$$\simeq \quad \subseteq \quad \sim$$

which we show by establishing that $\simeq$ is an $=_{\mathrm{SC}}$ respecting congruence:

$$p \simeq p' \quad \Rightarrow \quad \forall X[\cdot].\ X[p] =_{\mathrm{SC}} X[p']$$

Then we must show

$$\sim \quad \subseteq \quad \simeq$$

which we show by exhibiting a distinguishing context:

$$p \not\simeq p' \quad \Rightarrow \quad \exists X[\cdot].X[p] \neq_{\mathrm{SC}} X[p']$$

$\simeq$ is an $=_{\mathrm{SC}}$ respecting congruence: We proceed by structural induction on the context $X[\cdot]$. The base case is $X[\cdot] = [\cdot]$ and we must show

$$p \simeq p' \quad \Rightarrow \quad p =_{\mathrm{SC}} p'$$

which follows from the definition of $\simeq$ and Lemma 5.1. Next we must show that $\simeq$ is preserved by the *or* and *and* operators of SA. The result for the *or* operator follows directly from Lemma 4.1.

[10] For a discussion of the existence and uniqueness of $\sim$ see, for example, Section 7 of [BKO88].

Recalling Lemma 4.2 the only issue for the *and* operator is to establish inseparability, which we do now.

Let $p, p' \in \text{SA}$ with $p \simeq p'$, witnessed by $h : \mathcal{C}(p) \to \mathcal{C}(p')$, and $c_p \in \mathcal{C}(p)$. Let $X[\cdot] = p_1 \times [\cdot]$ with $p_1 \in \text{SA}$ and h_X defined as before, and let $c, c' \in \mathcal{C}(X[p])$ with $c_p \subseteq c$. Let $A \in \mathit{admit}_{X[p]}(c, \xi_i)$, $\xi_o = \mathit{actions}(A)$, $w = \psi_A(\xi_i/\xi_o)$ and $c' = \mathit{next}(c, A)$ so that

$$c \stackrel{w}{\longrightarrow}_{X[p]} c'$$

Let $A_{p_1} = A \cap \mathcal{A}(p_1)$, $A_p = A \cap \mathcal{A}(p)$, $\xi_i' = \xi_i \cup \mathit{actions}(A_{p_1})$. We know from Lemma 4.2 $A_p \in \mathit{admit}_p(c_p, \xi_i')$, with $\xi_o' = \mathit{actions}(A_p)$, $c_p' = \mathit{next}(c_p, A_p)$, and there is some $A_{p'} \in \mathit{admit}_{p'}(h(c_p), \xi_i')$ with $\xi_o = \mathit{actions}(A_p)$, and $h(c_p') = \mathit{next}(h(c_p), A_{p'})$. Let $w_p = \psi_{A_p}(\xi_i'/\xi_o')$ and $w_{p'} = \psi_{A_{p'}}(\xi_i'/\xi_o')$. We know further that $A' = A_1 \cup A_{p'} = \mathit{enabled}(h_X(c), \xi_i, A')$, and it remains only to show that A' is inseparable for $h_X(c)$ and ξ_i.

We proceed by contradiction. Let $w' = \psi_{A'}(\xi_i/\xi_o)$. Assume A' is separable, so there is some $\pi \in \mathit{action}(w')$ and some $\alpha \in A'$ with $\lambda \in \mathit{trigger}(\alpha)$ and $\lambda >_{w'} \pi$ such that $\alpha \notin \mathit{triggered}(\xi_i)$. Clearly $\alpha \notin A_{p_1}$. Since $A_{p'} = \mathit{enabled}_{p'}(h(c_p), \xi_i', A_{p'})$ there is some $\lambda' \in \mathit{actions}(A_{p_1})$ with $\lambda' >_{w_{p'}} \pi$. But $\lambda' \not>_{w_p} \pi$ contradicting that $p \simeq p'$.

$\simeq$ is the largest such congruence: Let $p =_{\text{SC}} p'$, witnessed by $h : \mathcal{C}(p) \to \mathcal{C}(p')$, but $p \not\simeq p'$. We exhibit a context $X[\cdot] = p_1 \times [\cdot]$ such that $X[p] \neq_{\text{SC}} X[p']$. This is easily done, since most of the work has gone before. Wolog, assume that there is some configuration $c_p \in \mathcal{C}(p)$ and input ξ_i with $A_p \in \mathit{admit}(c_p, \xi_i)$, etc. Let $A_{p'} \in \mathit{admit}(h(c_p), \xi_i)$, etc. $w_p = \psi_{A_p}(\xi_i/\xi_o)$ and $w_{p'} = \psi_{A_{p'}}(\xi_i/\xi_o)$. We know $\mathit{trigger}(w_p) = \mathit{trigger}(w_{p'})$ and $\mathit{action}(w_p) = \mathit{action}(w_{p'})$ and wolog for some $\pi \in \mathit{trigger}(w_p), \pi' \in \mathit{action}(w_p)$ we have $\pi >_{w_{p'}} \pi'$, but $\pi \not>_{w_p} \pi'$. Let α be an arrow of A_p with $\pi \in \mathit{trigger}(\alpha)$, and let $\xi = \{\pi_\alpha \in \mathit{trigger}(\alpha) | \pi_\alpha \in \mathit{trigger}(w_p)\}$. Thus ξ is the set of events that needs to be in the input ξ_i or α would not be triggered. Create a new $\alpha_s = (1, \pi'/\xi, 2)$ and a statechart $p_1 = [(\Box, \Box), \{\alpha_s\}]$. Let $c_1 =< 1 >$ and $c = c_p \cup c_1$. $A_p \cup \{\alpha_s\} \in \mathit{admit}_{X[p]}(c, \xi_i - \xi)$, but $A_{p'} \cup \{\alpha_s\}$ is separable for $h_X(c)$ and $\xi_i - \xi$. We have constructed a context in which $\psi_{A_{p'} \cup \{\alpha_s\}}(\xi_i - \xi / \xi_o \cup \xi)$ is not well-triggered.

□