On Statecharts with Overlapping

DAVID HAREL
The Weizmann Institute of Science
and
CHAIM-ARIE KAHANA
Bar-Ilan University

The problem of extending the language of statecharts to include overlapping states is considered. The need for such an extension is motivated and the subtlety of the problem is illustrated by exhibiting the shortcomings of naive approaches. The syntax and formal semantics of our extension are then presented, showing in the process that the definitions for conventional statecharts constitute a special case. Our definitions are rather complex, a fact that we feel points to the inherent difficulty of such an extension. We thus prefer to leave open the question of whether or not it should be adopted in practice.

Categories and Subject Descriptors: D.2.1 [**Software Engineering**]: Requirements/Specifications; D.2.2 [**Software Engineering**]: Tools and Techniques

General Terms: Design, Languages

Additional Key Words and Phrases: Higraphs, reactive systems, statecharts, visual language

1. INTRODUCTION

Motivated by the problem of specifying the behavior of complex reactive systems, statecharts have been introduced [4] as an extension of conventional finite-state machines and their state transition diagrams. The extended features present in statecharts include the encapsulation of states and the partitioning of states by cooperating orthogonal components, with broadcast communication. The former allows for hierarchical descriptions and interlevel transitions, and the latter makes it possible to specify multilevel concurrency and chain-reaction effects.

The language of statecharts has been used in the specification of a number of real-world systems, including significant parts of the avionics package for an advanced fighter aircraft at the Israel Aircraft Industries. The language is

This work is partially based on "State Overlapping in Statecharts," by C.-A. Kahana, M.S. thesis, Dept. of Mathematics and Computer Science, Bar-Ilan Univ., Ramat Gan, Israel, 1986. D. Harel's research was partially supported by a grant from the Gutwirth Foundation.
Authors' addresses: D. Harel, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, Israel; C.-A. Kahana, 37 Arazim St., Bnei-Brak, Israel.

also at the heart of the STATEMATE system, a graphical working environment for engineers involved in specifying and designing large systems [7].

Several subsequent papers have proposed formal syntax and semantics for statecharts [6, 9, 10, 11, 13]. To a large extent, these papers are devoted to the delicate semantic issues raised by the asynchronous nature of concurrency with chain reactions, and the zero-time nature of transitions with broadcast events and actions. (Similar semantic issues have been tackled in the context of the Esterel language; see [1] and [2].) As far as the syntax is concerned, the differences between the versions of statecharts appearing in these references are mostly in the labels allowed along transitions; that is, in the structure of the events, conditions, and actions. The syntax of the charts themselves, although presented nongraphically,[1] corresponds essentially to that appearing in [4].

In this paper we are interested in enriching the basic structure of the state space in statecharts to accommodate overlapping, as suggested in [4, Sect. 6.2]. Our work is motivated mainly by the fact that we have been repeatedly approached by users of statecharts, asking that we extend the language to include overlapping. These people often indicate that, in their opinion, the issue is trivial ("all you have to do is to allow me to draw states whose borderlines cross—the meaning is obvious—and eliminate any graphic editor's complaints when I do so"). Some also point to the fact that other languages provide analogous power, say, by subroutines or tasks. Our intuition was that extensions to graphical languages are particularly problematic, since a user has a large amount of liberty in drawing the figures, and features can easily be made to mesh with others in ways that might not have been anticipated by the language designer.

The desire to accommodate overlapping states is also consistent with [5], where *higraphs* are proposed as the set-theoretic graphical model underlying statecharts. The "blobs" in a higraph are organized as Euler circles (i.e., Venn diagrams) with a partitioning mechanism for describing Cartesian products. However, blob intersection has a natural set-theoretic semantics, and our feeling was that, when the temporal, dynamic semantics of states and events were to be adopted, the problem would become far more difficult.

Accordingly, the main purpose of our research here is to investigate the extent to which overlapping states can be conveniently added to statecharts. As the reader will see, the issue is indeed far from being trivial. The paper shows that mathematically such an extension is possible. However, the syntax and semantics we provide are quite complicated. We believe that the complication is in some way inherent in the desire to so extend the language, in the sense that any serious attempt to carry out the extension would result in similar complications. We thus leave the final judgment of whether the benefits outweigh the cost to the reader.

In providing our extension, we all but ignore the details of the labels. This decision eliminates some of the standard semantic issues of concurrency

[1] An exception is [10], in which a graphical, compositional syntax and semantics are presented.

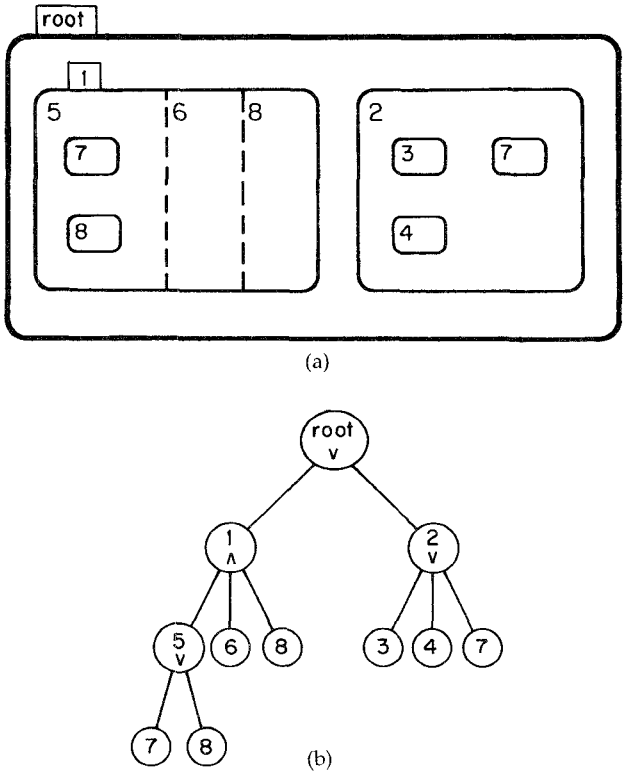


Figure 1

mentioned above, but these can be dealt with quite adequately using any of the approaches discussed in, for example, [6], [9]–[13].

States in a statechart are organized as an AND/OR tree (see Figure 1). They can be repeatedly decomposed into OR-substates (actually, XOR-substates) or AND-components, and the source and target states of a transition are allowed to reside on any level. To complete the (underspecified) definition of a transition entering a nonatomic state, one may use default arrows or history connectors. As we shall see in Section 2, in many cases a dynamic specification can benefit greatly from being able to define states that overlap and contain common substates, thus relaxing the restriction that the basic state structure is a tree. We also point to some of the subtleties of the problem, by way of discussing pitfalls that arise when this idea is addressed naively.

Section 3 introduces the extended state structure, and in Section 4 we incorporate transitions. When applied to a tree structure, our definitions can be seen to coincide with those of [4] and [6].

2. MOTIVATION AND OVERVIEW

Perhaps the most obvious justification for overlapping states is to avoid

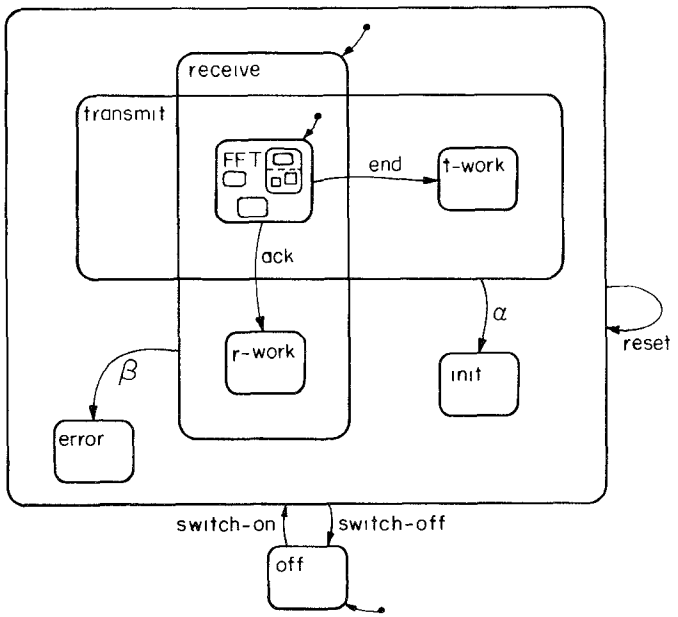


Figure 2

having to duplicate otherwise identical subcharts. Consider Figure 2,[2] which is a (rather crude and vastly underspecified) statechart of a simple modem. Both the *receive* and *transmit* states have an intricate *FFT* computation as one of their substates, which one might prefer to specify as in Figure 2, rather than having it attached as an orthogonal component or having it appear twice in its entirety.

Overlapping states are useful also for clustering states by common events and for setting priorities. Assume that a memory controller can receive requests from five potential customers, $u_0, \ldots, u_4$. Figure 3 is a self-explanatory statechart, in which the states that respond to each request are gathered into a common parent, from which the common events emanate naturally. This figure may also be viewed as a priority graph, in which a request from u_0 has the highest priority, one from u_2 has higher priority than one from u_4, and one from u_1 has higher priority than one from u_3.

However, a closer look at Figures 2 and 3 reveals a problem. Although the state structures in both figures are virtually identical, the intended meaning of the overlapping is quite different. In Figure 2 the modem cannot receive and transmit at the same time, and the system is really intended to be exclusively either in *receive.FFT* or in *transmit.FFT* (i.e., the system's *configuration* cannot include both *states*). In Figure 3, on the other hand, we obviously intend the system to be in both p_4.*wait* and p_3.*wait* simultaneously, in order to be able to respond to any request.

[2] This example, as well as those in Figures 3 and 7, are based on examples appearing in [3].

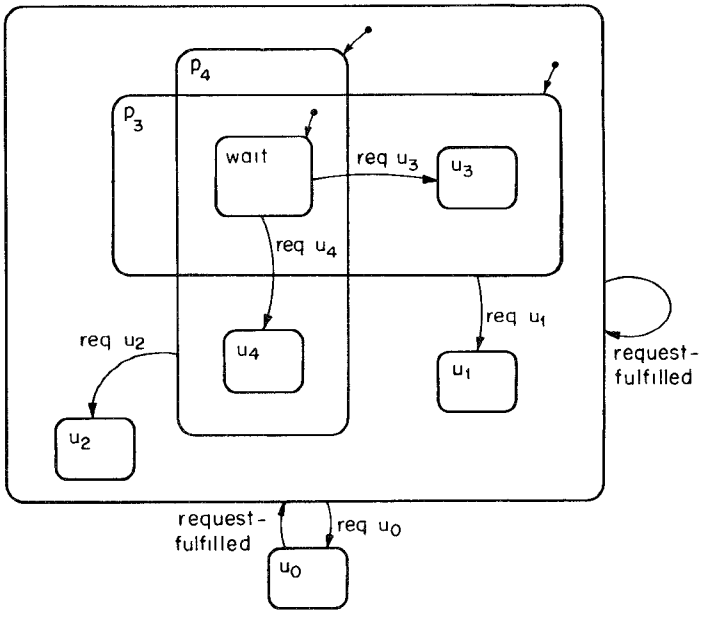


Figure 3

We might attempt to distinguish between these cases by means of two syntactically distinct types of overlapping; say, with dots signifying exclusivity, as shown graphically in Figure 4. This has a number of drawbacks. First, the syntax would have to be very restrictive, so as to avoid contradictions like the one in Figure 5, where the two types of intersection entail that it is impossible to be in the state lying inside the intersection, since we would have to be in *both* of B and C, as well as in only *one* of them. Second, restricting the overlapping feature to take on only XOR and AND meanings renders it but a minor improvement over conventional statecharts, in which these are the only two ways of decomposing states. The real challenge raised by the mechanism of overlapping is to model other combinations of states, such as a nonexclusive OR (which, in a certain technical sense, cannot be described with an ordinary statechart), or intricate combinations of more than two states. Third, it may be desirable to make the actual relationship between overlapping states dynamically flexible, depending, say, on the state we are coming from or on the particular event that has occurred. Having two (or more) kinds of overlapping, each giving rise to a fixed Boolean combination between its constituent states, is therefore not good enough. Finally, even if we agree to make do with XOR alone, we would still have to provide means for specifying which of the two states in question we want to enter when taking a given transition.

Our approach is to allow a single kind of intersection, but endow it with flexible semantics, so that the system can actually be in any subset of the intersecting states at any given time. A transition can then be extended by

Figure 4

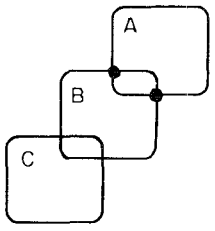
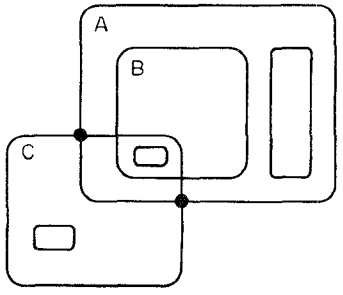


Figure 5



specifying which states we insist on entering when taking it, and which we do not mind not entering (although we might be entering their descendants). This idea complicates things considerably. A transition is no longer given by the set of lowest-level source and target states (and a label). It may now also contain information about nonbasic states that are to be entered in the process of reaching the target, and those that are not to be entered.

Graphically, this is specified by arrows that are allowed to skip edges (see [4, Fig. 45] or [5, Fig. 28][3]). Thus, in Figure 6, if we are in F and β occurs, we will enter both C and B (and remain also in A). If γ occurs, we enter C without entering B. Taking the α arrow when in state G entails entering B and C but not A. We should remark that skipping an edge does not necessarily prevent the system from being in the skipped state. In fact, whether or not the skipped state is really entered can depend on the current configuration. (Figure 8 indeed contains such a case.) For example, taking the δ arrow from H in Figure 6 results in the system entering A. As shown later, this is because there must be at least one continuous sequence of ancestoral states leading to the root from any atomic state we happen to be in.

There is another important motivation for adding overlapping states to statecharts, which causes an additional complication. It involves the desire to use overlapping for graphical specification of synchronization mechanisms. Consider Figure 7, which contains five orthogonal components, A through E, that are suppose to be synchronized from left to right in a pipeline-like manner. The figure shows how the intended behavior can be depicted graphically, without the need for explicit message passing or other symbolic means. The system starts out in its five *wait* states. When α arrives, A enters A_1,

[3] This figure was erroneously labeled Figure 27 by the typesetter of [5].

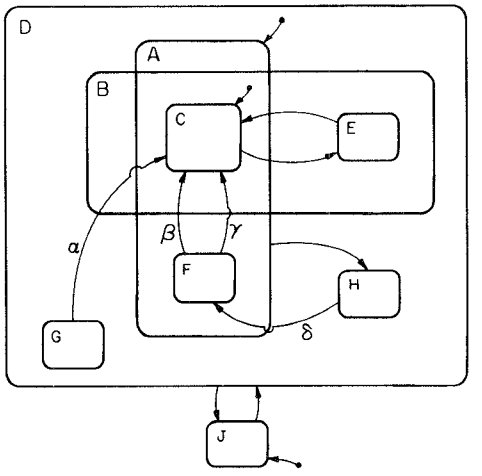


Figure 6

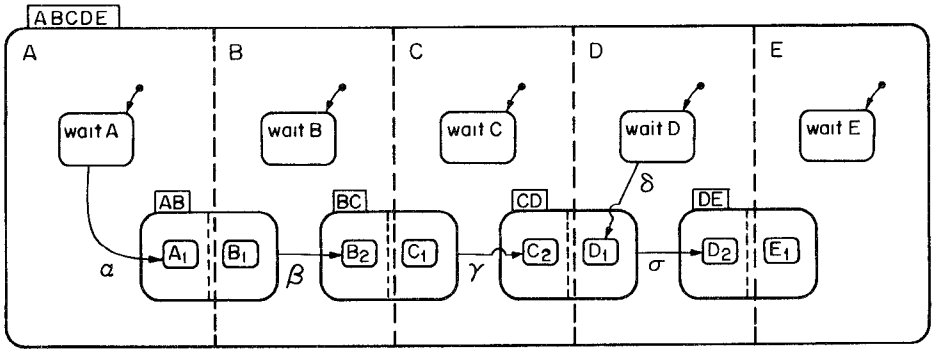


Figure 7

but the event also synchronizes B to B_1. The other components remain in their *wait* states. When in this AB pair, β causes B to move over to B_2, but also to synchronize C to C_1. The next transition is different; here we have chosen to skip the boundary of CD when γ arrives, entering C_2 without synchronizing D to D_1. The synchronization of D will take place when δ occurs.

While the figure appears to capture this intended behavior quite well, it illustrates two rather peculiar, yet desirable, properties that we would like to incorporate into our semantics. They involve states like AB, which are direct offspring of an AND state (i.e., state $ABCDE$), yet are themselves of type AND. The first property is that, unlike normal direct offspring of an AND state, the system need not necessarily be in every such offspring whenever it is in its parent. Thus, when we are in state $ABCDE$ in Figure 7 (say, by being in the five *wait* states) we need not be in any of the four synchronizing states, AB through DE.

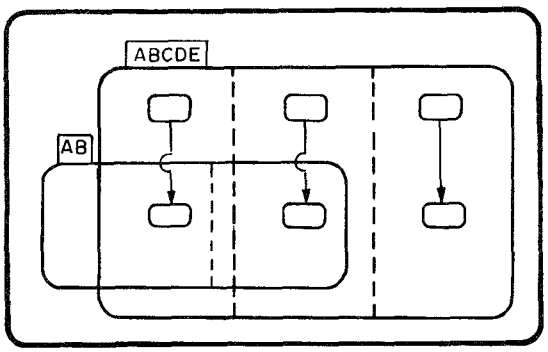
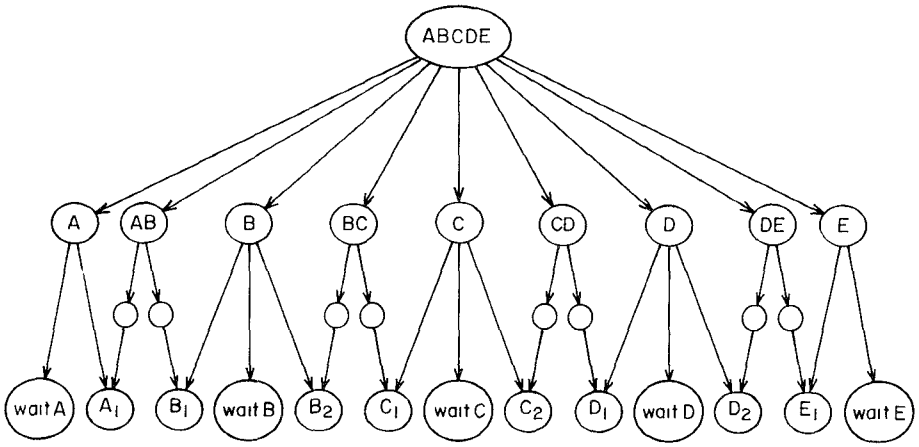
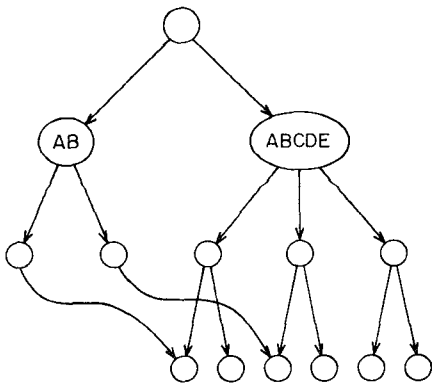


Figure 8



(a)



(b)

Figure 9

The other special property is that, although the system can be in *some* of an AND state's orthogonal components without being in the state itself, once the system happens to be in *all* of its components, it is considered to be in the state itself too. For example, we may be in C_2 without being in CD, but once we also enter D_1, the intention is that *now* we are in CD, so that event σ, for example, will take effect if and when it occurs. This property should not hold unless the synchronizing state is a direct offspring of the synchronized AND state; in Figure 8, for example, entering two components of AB by the skipping arrows will not necessarily cause the system to be in AB. The difference is that in Figure 7 state AB is a substate of $ABCDE$, whereas in Figure 8 AB is its *sibling*. Graphically, the difference is that in Figure 8 AB is drawn entirely within $ABCDE$. To illustrate this difference, see Figure 9, which contains the two corresponding trees.

Having described some of the issues that our syntax and semantics address, we can now get into the definitions themselves.

3. OVERLAPPING STATES

3.1 Syntax

As in [6], the syntax of states consists of a set of states S, and two functions:

(1) $\rho: S \to 2^S$, and
(2) $\psi: S \to \{\text{AND}, \text{OR}\}$.

These are the hierarchy function and the type function, respectively. Hence, $\rho(x)$ is the set of direct descendant states of state x. Let ρ^* and ρ^+ be the reflexive transitive closure of ρ and the irreflexive transitive closure, respectively.

The functions ρ, ρ^*, and ρ^+ are extended to sets of states by

$$\rho(X) = \bigcup_{x \in X} \rho(x).$$

If $x \in S$, we require that $x \notin \rho^+(x)$. We also require the existence of a special state *root* in S such that $\rho^*(root) = S$. Note that these restrictions limit the state-graph to be a connected directed acyclic graph (DAG). A conventional (nonoverlapping) statechart, satisfying the additional restriction $\rho(x) \cap \rho(y) \neq \varnothing \to x = y$, is, in fact, a tree. (Compare Figure 1 with Figure 9.) A state x is *basic* if $\rho(x) = \varnothing$; for a basic state, we require that $\psi(x) = \text{AND}$. Here are some additional definitions:

—**superstate**$(x) = \{y \mid x \in \rho(y)\}$ (**superstate** is just ρ^{-1}).

—**ancestoral**$(x, y) \equiv x \in \rho^*(y) \vee y \in \rho^*(x)$ (x and y are ancestoral).
—**disjoint**$(x, y) \equiv \rho^*(x) \cap \rho^*(y) = \varnothing$ (x and y share no common substate).
—**overlap**$(x, y) \equiv \neg$**ancestoral**$(x, y) \wedge \neg$**disjoint**(x, y) (x and y are neither ancestoral nor disjoint, so they must be overlapping).

Figure 10

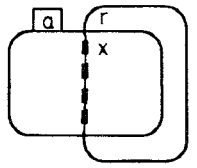
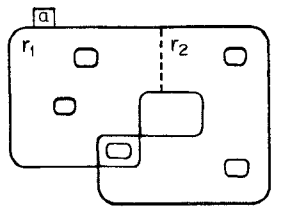


Figure 11



We also extend **superstate** and **overlap** to sets (the latter in the second component only), by

$$\mathbf{superstate}(X) \equiv \bigcup_{x \in X} \mathbf{superstate}(x),$$

$$\mathbf{overlap}(x, Y) \equiv \exists y \in Y.(\mathbf{overlap}(x, y)).$$

Although the semantics provided in the paper works well without any additional restrictions on the syntax, some cases do not lend themselves to satisfactory visual depiction. It might thus be a good idea to enforce the following restrictions: They are merely aimed at making drawings feasible. Assume that a, a_1, and a_2 are distinct AND states; that r, r_1, and r_2 are distinct OR states; and that x, y, and z are distinct states of either type. The restrictions are as follows:

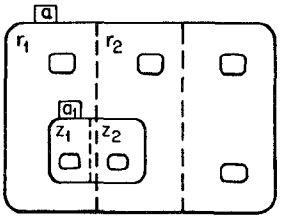
—If the AND state a is not basic, then $\mathbf{superstate}(\rho(a)) = \{a\}$. This prevents the likes of Figure 10, where the substate x of an AND state has an additional superstate.

—Two direct substates of a state are never ancestoral. Formally, $\mathbf{superstate}(x) \cap \mathbf{superstate}(y) \neq \varnothing \rightarrow \neg\mathbf{ancestoral}(x, y)$.

—Direct OR substates of an AND state have no common descendants. Formally, $r_1, r_2 \in \rho(a) \rightarrow \mathbf{disjoint}(r_1, r_2)$. Thus, Figure 11 (in which the state a curves around and overlaps with itself) is forbidden.

—If the AND states a and a_1 are not basic states and if $a_1 \in \rho(a)$, there must be at least two OR states $r_1, r_2 \in \rho(a)$ such that both $\mathbf{overlap}(a_1, r_1)$ and $\mathbf{overlap}(a_1, r_2)$ hold. (See Figure 12; in fact, by this restriction, any AND substate of an AND state must be essentially as in Figure 12.) This restriction also helps clause (2′) in Section 3.2 to deal correctly with the distinction between the AND and OR substates of an AND state.

Figure 12



3.2. Semantics

In [6] a legal configuration of a nonoverlapping statechart is defined as follows:

—The *least common ancestor* (**lca**) of a set of states X is a state x for which $X \subseteq \rho^*(x)$ (i.e., x is a common ancestor) and $\forall s \in \rho^+(x).(X \not\subseteq \rho^*(s))$ (i.e., it is the least such).

—Two distinct states x and y are *orthogonal* if $\psi(\mathbf{lca}(\{x, y\})) = \text{AND}$. In addition, a state x is considered to be orthogonal to itself. A set of states is *orthogonal* if its states are pairwise orthogonal.

—A *maximal orthogonal* set is an orthogonal set that cannot be extended (by adding elements) to a larger orthogonal set.

—A *legal configuration* is a maximal orthogonal set of *basic* states.

The *upward closure* of a configuration with respect to the **superstate** operation (i.e., the reflexive transitive closure of ρ^{-1}, applied to the configuration) can be shown to satisfy the following, for every x in C:

(1) $\psi(x) = \text{OR} \rightarrow |\rho(x) \cap C| = 1$;

(2) $\psi(x) = \text{AND} \rightarrow \rho(x) \subset C$; and

(3) $\mathbf{superstate}(x) \subset C$.

These three clauses mean, respectively, that *exactly one* direct descendant of an OR state of C is in C, that *every* direct descendant of an AND state of C is in C, and that C is indeed upward closed.

The definition of a configuration given above is not suitable for the overlapping states case, since here the **lca** of a set of states need not be unique; hence, the orthogonality relation between two states is not well defined. Moreover, here it is not enough to describe which basic states the system is in, since, for example, in Figure 13 being in the basic state z does not determine which of x and/or y we are in. Instead, we base our definition of a legal configuration on a modification of the three clauses for the upward closure.

The first clause, $\psi(x) = \text{OR} \rightarrow |\rho(x) \cap C| = 1$, is too strong, since an OR state might have two (or more) substates in the configuration (when they overlap). To overcome this, we define paths. We say that $\mathbf{path}(x, y, S)$ holds, for states x and y and a set of states S, if there exists a sequence of states $\{x_i\}_{i=1}^n$, all of which are in S, and such that $x_{i+1} \in \rho(x_i)$, $x_1 = x$, and $x_n = y$. In particular, $\mathbf{path}(x, x, S)$ holds iff $x \in S$.

Figure 13

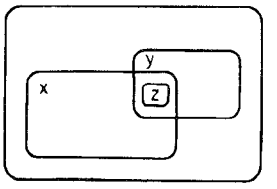
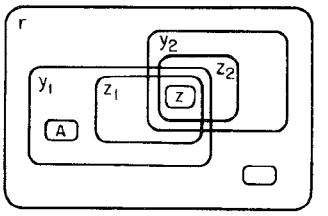


Figure 14



Clause (1) now becomes clause (1′):

(1′)

$$\psi(x) = \text{OR} \rightarrow (\rho(x) \cap C \neq \emptyset \wedge \forall y_1, y_2 \in \rho(x) \cap C.\ \exists z.\ (\mathbf{path}(y_1, z, C) \wedge \mathbf{path}(y_2, z, C))).$$

Figure 14 is an example illustrating clause (1′). For $C = \{r, y_1, y_2, z_1, z_2, z\}$, both $\mathbf{path}(y_1, z, C)$ and $\mathbf{path}(y_2, z, C)$ hold, so y_1 and y_2 can both be in the configuration C, although r is an OR state.[4]

The second clause, $\psi(x) = \text{AND} \rightarrow \rho(x) \subset C$, should now be limited to force C to contain only OR substates of an AND state that is in C. An AND substate of an AND state, such as a_1 in Figure 12, behaves differently. As explained in Section 2, if its substates are all in C, then it must be there too, but the fact that its superstate (a in Figure 12) is in C does not force it to be there also. Hence, clause (2) now becomes

(2′)

$$\psi(x) = \text{AND} \rightarrow \forall y \in \rho(x).((\psi(y) = \text{OR} \rightarrow y \in C) \wedge ((\psi(y) = \text{AND} \wedge \forall z \in \rho(y).(\psi(z) = \text{OR} \rightarrow \rho^*(z) \cap C \neq \emptyset)) \rightarrow y \in C)).$$

In Figure 12, the states covered by the internal $\forall$ quantifier in (2′) are z_1 and z_2 (x and y in (2′) stand for a and a_1 in Figure 12, resp.).

The third clause, $\mathbf{superstate}(x) \subset C$, is now replaced by

(3′) $$x \neq root \rightarrow \mathbf{superstate}(x) \cap C \neq \emptyset,$$

since, as in Figure 13, z can be in the configuration without it necessarily containing both x and y.

[4] The requirement $\exists z.(\mathbf{path}(y_1, z, C) \wedge \mathbf{path}(y_2, z, C))$ could not be easily replaced by $(\rho^*(y_1) \cap \rho^*(y_2) \cap C) \neq \emptyset$, since then we would have to find other means for preventing the set $\{r, y_1, y_2, z_2, A, z\}$, which contains both A and z, from being a configuration.

To summarize, a nonempty set of states C is a *configuration* if it satisfies clauses (1′), (2′), and (3′), for each $x \in C$. As the reader can check, this definition coincides with the usual definition (of the upward closure of a configuration) for the nonoverlapping case [6].

4. INCORPORATING TRANSITIONS

4.1 Syntax

For the nonoverlapping case, a *transition* is defined in [6] to be a triple $\langle S, L, T \rangle$, where

—S, a set of states, is the *source*;
—L, is the *label* of the transition (a formal definition appears in [6]); and
—T, a set of states, is the *target*.

The source of the transition in Figure 15, for example, is $\{s_1, s_2\}$, while the target is $\{t_1\}$. A transition may be taken in a system configuration[5] SC whose state configuration is C if it is structurally relevant to C (i.e., $S \subseteq C$) and if its label is enabled in SC.

As discussed in Section 2, an arrow in a statechart with overlapping states may skip over state borderlines. Hence, to distinguish in the nongraphical syntax between transitions α and β of Figure 16, we must define the target set T to consist of all states that the arrow crosses, as well as those incident with the arrow's head. Thus, writing T_t (resp., S_t, E_t) for the target set (resp., source set, exit set) of a transition t, we obtain, in this case, $T_\alpha = \{A, D\}$ and $T_\beta = \{B, D\}$. For the transition of Figure 15, the target set is no longer $\{t_1\}$; it now becomes $\{t_3, t_2, t_1\}$. However, it turns out that this extension is not enough. Consider transitions α and β of Figure 17. $T_\alpha = T_\beta = \{B\}$ and $S_\alpha = S_\beta = \{1\}$; but after taking α in configuration $\{R, A, 1\}$, we find ourselves in $\{R, A, B, 2\}$, whereas after taking β in the same configuration, we are in $\{R, B, 2\}$. The point is that we must capture in our syntax the fact (easily depicted graphically) that β leaves A but α does not. We thus want to define the *exit set* E of α and β, by $E_\alpha = \{1\}$, and $E_\beta = \{1, A\}$. We should remark that the exit set E is a new, fourth component of a transition, whereas our new target set replaces the old one.

Formally, then, a transition in a statechart with overlapping states is a quadruple $\langle S, E, L, T \rangle$, where the source S and label L are as above, E is the exit set (which we take to include S too; i.e., $S \subseteq E$), and T is the target set of states entered and not only those at the head of the arrow.

[5] The term is defined formally in [6], and the changes required for the overlapping states case are few and not directly relevant to the goals of this paper. Intuitively, a system configuration is a state configuration together with the values of variables and conditions, the events taking place, etc. A label "*ev*[*cond*]/*act*" is enabled in a system configuration if the event *ev* occurred and the condition *cond* is true. Upon taking the transition, the action *act* is executed. In our case, as mentioned earlier, we must deal with upward closed configurations, not just configurations consisting of sets of basic states.

Figure 15

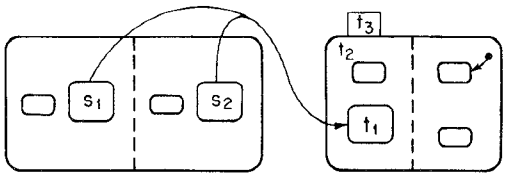


Figure 16

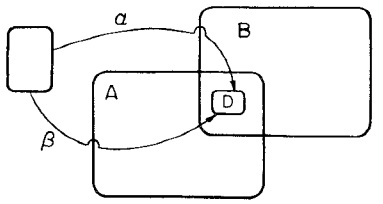
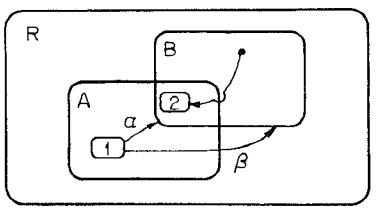


Figure 17



Consider Figure 18. Here, $S_\alpha = \{1\}$, and $E_\alpha = \{1, A, B\}$, since the source is captured graphically by the arrow's tail, while the exit set contains the states actually crossed by the arrow. From a semantic point of view, although $E_\alpha = E_\beta = \{1, A, B\}$, β will not be taken in configuration $C = \{R, 1, A\}$, since $S_\beta = \{1, B\}$ and $S_\beta \not\subseteq C$. The transition α, on the other hand, will be taken in this configuration. However, the implications of taking α and β are similar in terms of the states actually exited and entered, since an implication is a function of the exit set. Thus, S can be viewed as a condition, and E as a parameter, for calculating the results of taking the transition. As far as the target set T is concerned, we do not need this separation.

Before going on, let us make two more remarks on the syntax. First, although we have to specify skipping over a state borderline on state entry, we have found skipping on *exit* (e.g., arrow α in Figure 19) to be inessential. For example, in Figure 19, arrow α can be replaced by arrow β, since their target, being exclusive of state B, forces them both to leave it.

Second, we do not keep track of the precise itinerary of an arrow in a chart, but only its source, exit set, and target. Hence, arrow α in Figure 20 is not interpreted as the statechart designer probably meant, and we do not supply formal means for capturing such intentions. Note also that the target and exit sets are themselves unordered. Hence, transitions β and γ in Figure 20 are equivalent. It may be worthwhile to distinguish between such cases in a more sophisticated semantics for statecharts. Perhaps this can be done by introducing a new, more refined level of steps, beyond the microsteps of [6].

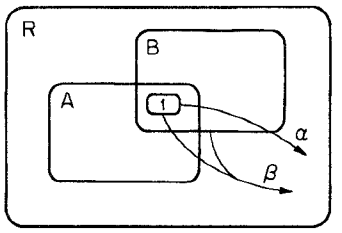


Figure 18

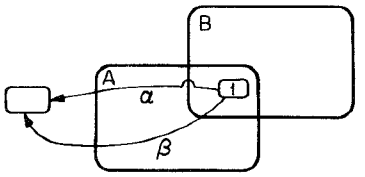


Figure 19

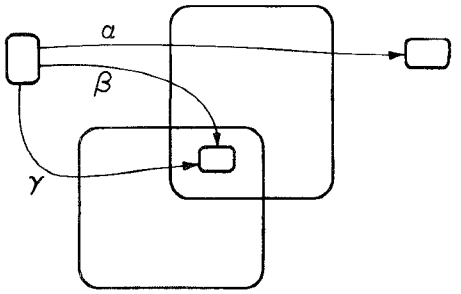


Figure 20

Now that we have a syntax for transitions, we can proceed to the semantics. We define the states that are actually left as a result of taking a transition in some configuration C (Section 4.2) and the states that are entered (Section 4.3). These capture the semantic implications of a single transition, defining a new configuration. Two or more transitions taken concurrently are discussed in Section 4.4.

4.2 States Left by a Transition

In this section we define the set of states left by some transition $\langle S, E, L, T \rangle$ from some configuration C. We denote this set by **left**$(C, \langle S, E, L, T \rangle)$. In the conventional, nonoverlapping case [6], the states actually left when taking a transition are exactly those in the subchart rooted at the **lca** of the transition. The **lca** of a transition is either the **lca** of the source and target taken together or its closest OR superstate. This simple idea is not adequate here. For example, it does not explain why transition β in Figure 21 does not cause A_3 to reenter its default.

In the overlapping case, a transition must cause all states in the exit set E to be left, including their descendants. The states that overlap with E should also be left (with *their* descendants). For example, both states A and B, and their descendants, are left when α is taken in Figure 22. Note that state B is

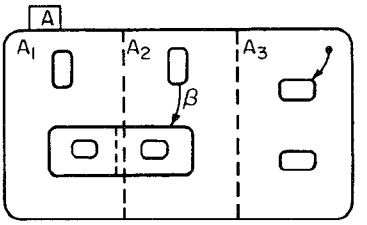


Figure 21

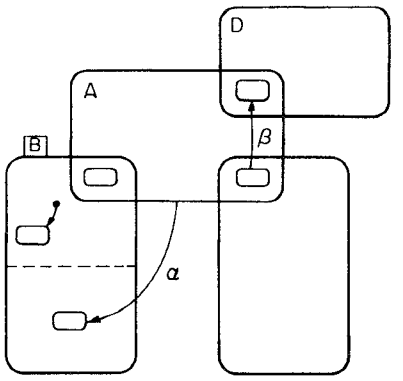


Figure 22

reentered with a new default in its upper component. Formally, our first approximation to the set of states left by a transition with exit set E is

$$\rho^*(E \cup \{x \mid \textbf{overlap}(x, E)\}).$$

An exception to this definition may occur when the entire transition is located inside the state, like state A when transition β is taken in Figure 22. In this case, A overlaps with the exit set of β but should not be left. Formally, the exception includes the states in **arrow-inside**(E), where

$$\textbf{arrow-inside}(E) = \{x \mid x \notin E \wedge E \cap \rho^+(x) \neq \varnothing\}.$$

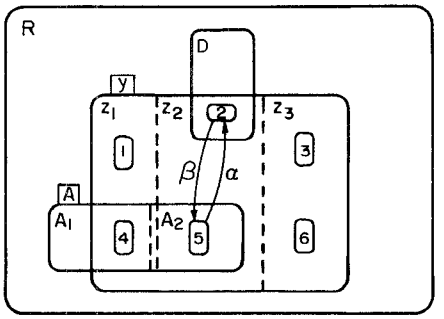
In other words, if a state x is not exited, but at least one of its descendants is, then the transition is entirely inside x.

Another exception can be found in Figure 23. State 4 therein is left when taking α in configuration $\{R, A, A_1, A_2, 4, 5\}$, since A is left and D is entered. However, we do not want it to be left if the configuration was $\{R, y, z_1, z_2, z_3, 4, 5, 6, A, A_1, A_2\}$. To capture such cases, let us define the following:

$$\begin{aligned}&\textbf{and-super-neighbors}(E, C)\\ &\quad = \{z \mid \psi(z) = \text{OR} \wedge \rho^*(z) \cap E = \varnothing \wedge\\ &\qquad \exists y \in (\textbf{superstate}(z) \cap \textbf{arrow-inside}(E) \cap C).(\psi(y) = \text{AND})\}.\end{aligned}$$

Now, ρ^*(**and-super-neighbors**(E, C)) is taken as an additional set of states that are not to be left. Note that z, in this definition, represents states z_1 and z_3 of Figure 23, but not state z_2, since $\rho^*(z_2) \cap E \neq \varnothing$.

Figure 23



Another set of states that *should* be left consists of those that are inconsistent with the target set of the transition. For example, state 1 is left when β is taken in Figure 23. Formally, this set is defined as

$$\textbf{contradict}(T) = \{x \mid \forall G.(T \cup \{x\} \subseteq G \rightarrow G \text{ is not a configuration})\}.$$

Before continuing, let us summarize formally what we have done so far. Our present version of the states left when taking the transition $\langle S, E, L, T\rangle$ in configuration C is

$$\begin{aligned}&\textbf{left}(C, \langle S, E, L, T\rangle)\\ &\quad = ((\rho^*(E \cup \{x \mid \textbf{overlap}(x, E)\}) - (\textbf{arrow-inside}(E)\\ &\qquad \cup \rho^*(\textbf{and-super-neighbors}(E, C)))) \cup \textbf{contradict}(T)) \cap C.\end{aligned}$$

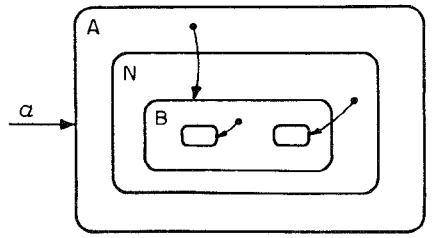
Note that, in contrast with the states left by a transition in the nonoverlapping case, **left** depends not only on the transition but also on the configuration. This is a subtle and important difference. In the nonoverlapping states case, we can preprocess the statechart by computing the consequences of the transitions once, in advance. Here, the effects of a transition are dynamic and have to be computed anew for different configurations.

4.3 States Entered by a Transition

In the nonoverlapping case, if x is the **lca** of the transition taken, the set entered after leaving $\rho^*(x)$ is defined to be a function of x, the target set T, and default arrows.[6] (This function is called "C" in [6], but since we use C for a configuration, we shall later call it **added**.) Our syntax includes a default function, denoted $\delta: S \rightarrow 2^S$. This function provides, for a state x, the set of states pointed to by the default arrow of x. In the nonoverlapping case, the

[6] It may also depend on the history of the computation, if the statechart contains history connectors. In this paper we assume no such history connectors and no labels on default arrows. We do this in order to highlight the special problems with overlapping states, with as little overhead as possible. An expanded version of our semantics that deals with history of various kinds and labeled default arrows appears in [12].

Figure 24



elements of $\delta(x)$ are restricted to be in $\rho^+(x)$. Here we need to allow $\delta(x)$ also to contain states that overlap with x. We thus have

$$\delta(x) \subseteq (\rho^+(x) \cup \{y \mid \mathbf{overlap}(x, y)\}).$$

Processing a transition when in configuration C starts by removing the set $\mathbf{left}(C, \langle S, E, L, T \rangle)$ from C and adding the target set T. Omitting the parameters of **left** for simplicity, we thus obtain a set of states, called **mandatory**, that must appear in the new configuration:

$$\mathbf{mandatory} = (C - \mathbf{left}) \cup T.$$

Let us now deal with defaults. For a conventional, nonoverlapping statechart, a default arrow is taken in a state whenever the choice of a substate to enter had not been already made by the transition itself or by a default arrow that was taken in one of its superstates. For example, in Figure 24, when entering A by transition α, we use the default in states A and B, but not the one in state N.

Formally, if we already have a candidate **new-config** for the new configuration, then the following recursive predicate indicates whether or not the default appearing in a state x is taken:

$$\begin{aligned}&\mathbf{default\text{-}used}(x, \mathbf{new\text{-}config})\\ &\quad \equiv \forall y \in (\mathbf{superstate}^+(x) \cap \mathbf{new\text{-}config}).\\ &\qquad\qquad (\mathbf{default\text{-}used}(y, \mathbf{new\text{-}config}) \rightarrow (y \text{ does not determine in } x)).\end{aligned}$$

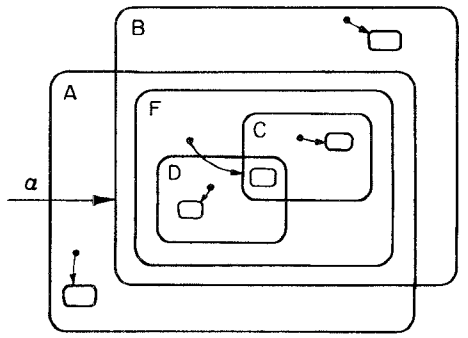
For the moment, "y does not determine in x" is taken to stand for $\rho^+(x) \cap \delta'(y) = \varnothing$, where

$$\delta'(y) = \begin{cases} \delta(y) & y \neq root, \\ \mathbf{mandatory} & y = root. \end{cases}$$

Note that, since $\mathbf{default\text{-}used}(root, \mathbf{new\text{-}config})$ is trivially true, we ensure that **mandatory**, that is, the states in C that were not left, together with those entered, are all in the next configuration.

Now, when overlapping is allowed, a transition can determine which of the descendants of a state it will enter, not only by entering one or more of them directly but also by entering states with which they overlap. Of course, this kind of choice may be made either by the transition itself or by a default arrow taken in a superstate.

Figure 25



For example, when taking transition α in Figure 25, the only default arrow taken is that of state F, since α prescribes entrance to both A and B, and taking any other default would contradict that. Hence, "y does not determine in x" now becomes

$$\rho^+(x) \cap \delta'(y) = \varnothing \wedge \neg \textbf{overlap}(x, \delta'(y)).$$

If **default-used** is true, meaning that the default arrow should be taken, then **new-config** must contain $\delta(x)$ as a subset, the states whose entrance is prescribed by x's default. In fact, it should contain $\delta'(x)$, and we occasionally need something even stronger. For a set of states X, define $\textbf{lub}(X)$ as the intersection of all configurations that include X if that intersection is not empty, and X if it is. Now, $\delta''(x)$ is defined as $\textbf{lub}(\delta'(x))$. Our final definition of "y does not determine in x" is thus

$$\rho^+(x) \cap \delta''(y) = \varnothing \wedge \neg \textbf{overlap}(x, \delta''(y)).$$

The set of configurations that are preliminary candidates for being the next configuration can now be defined as follows:

$$\begin{aligned}
&\textbf{candidates}(C, \langle S, E, L, T \rangle) \\
&\quad = \{\textbf{new-config} \mid \textbf{new-config} \text{ is a legal configuration} \\
&\quad \wedge \forall x \in \textbf{new-config}.(\textbf{default-used}(x, \textbf{new-config}) \rightarrow \\
&\qquad \delta''(x) \subseteq \textbf{new-config})\}.
\end{aligned}$$

(Note that any **new-config** that finds its way into **candidates** will contain **mandatory**, by the definitions of δ' and **default-used**.) The result of taking the transition is now given as follows: If the set **candidates** (whose parameters we omit for brevity) is empty, the transition is undefined, since it contains an internal contradiction. See, for example, arrow α in Figure 26. If it is a singleton, then its single member is taken as the next configuration. If, on the other hand, it contains more than one candidate configuration, we form the intersection

$$\textbf{min-candidate} = \bigcap_{N \in \textbf{candidates}} N.$$

Now, if **min-candidate** is a member of **candidates**, we take it to be the new configuration, and it will actually be the *minimal* candidate solution. If not,

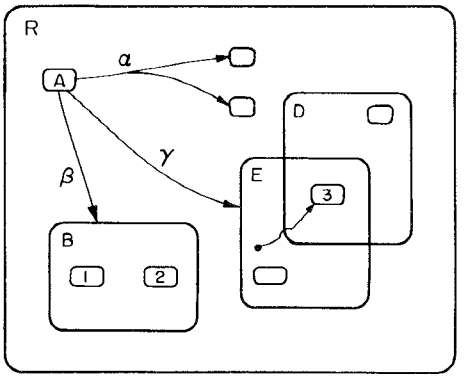


Figure 26

then the transition is undefined (or defined as nondeterminism) because there is, in fact, no minimal configuration.

For example, arrow β of Figure 26 allows $\{R, B, 1\}$ or $\{R, B, 2\}$ as **new-config**'s, since $\delta(\beta) = \emptyset$. Hence, there is no minimum and no next configuration. On the other hand, arrow γ allows $\{R, E, 3\}$ or $\{R, E, 3, D\}$, and hence, the next configuration is $\{R, E, 3\}$.

We should mention that to start off a statechart we initialize it to the start configuration consisting of applying our definitions for defaults to **mandatory** = $\{root\}$.

4.4 Parallel Transitions

In a nonoverlapping statechart, a set of transitions may be performed simultaneously if it is structurally consistent, meaning that the **lca**s of any two transitions in the set are orthogonal. Since **lca**s cannot be exclusive for transitions relevant to a common state configuration, this requirement is similar to the following one: For any two such transitions, the **left** ∪ **added** sets[7] are pairwise disjoint, so that transitions do not interfere with each other. We adopt this requirement. For example, in Figure 27, α and β are consistent if B is in the configuration and A is not. On the other hand, if A is in the configuration, but B is not, then operating each arrow separately will cause the other component of B to enter its default, and the two transitions will thus no longer be consistent. Hence, as in the definition of **left**, in the overlapping states case consistency between transitions is not a structural property but a dynamic, temporal one. For each state configuration, the consistency question arises anew.

We are not quite done yet. Consider Figure 28. After taking two transitions as if they were taken separately,[8] the result may not be a legal configuration.

[7] The set **added** is the set of states added to the original configuration C after removing **left**, which is discussed in Section 4.2, to obtain the new configuration. Thus, it is simply **min-candidate** − (C − **left**).

[8] As the reader may verify, a consequence of our requirement that the **left** ∪ **added** sets are disjoint for every pair of transitions that are to be taken simultaneously is that the transitions can be taken in any order.

Figure 27

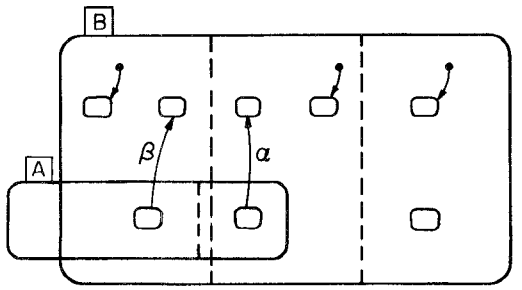
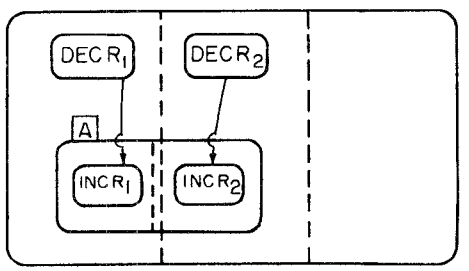


Figure 28



In this example, state A must be in the next configuration whenever its descendants are in it (see Section 3.2). Hence, we must complete the result to a unique minimal legal configuration.

Formally, let t_i, for $1 \leq i \leq n$, be a set of transitions whose respective sets in a given configuration C are $\mathbf{left}_i$ and $\mathbf{added}_i$, respectively. Denote

$$\textbf{new-full-config} = \textbf{lub}\left(\left(C - \bigcup_{1 \leq i \leq n} \textbf{left}_i\right) \cup \left(\bigcup_{1 \leq i \leq n} \textbf{added}_i\right)\right).$$

The set $\{t_i\}$ is *consistent* in C if

(1) $\forall i \neq j$, $(\mathbf{left}_i \cup \mathbf{added}_i) \cap (\mathbf{left}_j \cup \mathbf{added}_j) = \varnothing$; and
(2) **new-full-config** is a legal configuration.

Now, if the set $\{t_i\}$ is indeed consistent in C, its transitions are taken together, and **new-full-config** is the next configuration. The sets **left** and **added** associated with taking this parallel transition are given by

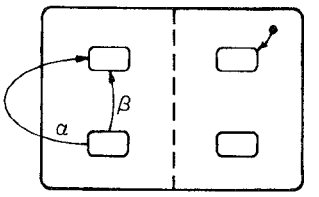
$$\textbf{left} = \bigcup_{1 \leq i \leq n} \textbf{left}_i,$$

$$\textbf{added} = \textbf{new-full-config} - (C - \textbf{left}).$$

Note that, as in the nonoverlapping states case, there may be several maximal consistent sets of transitions.

The semantics of transitions, as presented in the previous sections, is consistent with the semantics for nonoverlapping statecharts (see, e.g., [6]), in the following sense: If our semantics is applied to a conventional state-

Figure 29



chart, in which transitions are not allowed to exit and then reenter states, the two semantics coincide. (The two transitions in Figure 29, for example, have the same semantics in the conventional case, but differ in ours.) As explained earlier, the usual semantics for statecharts do not deal adequately with transitions that are allowed to contain more than just a source and target, whereas ours .does.

5. DISCUSSION

The addition of overlapping to statecharts enriches the formalism. Although it does not provide any new expressive power (the extended statecharts can still express only regular sequences of events), it enables more compact and natural specifications. However, as explained in the introduction, the extension also complicates things considerably. Thus, the question of whether the benefits are worth the cost one has to pay does not have a clear answer.

An interesting direction for further work involves the efficient implementation of overlapping, providing simulation capabilities. For example, it appears that extending the implementation of statecharts in the STATEMATE system [7] would not be straightforward. This is due to the dynamic nature of the implications of a transition in the overlapping states case and to the need for more complicated graphics. It is also of interest to extend other languages that are based on higraphs to include overlapping. The activity charts and module charts of STATEMATE [7] come to mind, as do the Miró diagrams for security constraints of [8].

Other extensions of statecharts, such as parameterized states, recursion, and transitions endowed with probabilities [4], would also become more difficult when coupled with overlapping.

ACKNOWLEDGMENTS

We would like to thank Doron Drusinsky for many helpful discussions, and Yehuda Barbut for a superb job with the figures. The referees of previous versions of the paper made several very useful suggestions.

REFERENCES

1. Berry, G., and Cosserat, L. The synchronous programming language ESTEREL and its mathematical semantics. In *Proceedings of the CMU Seminar on Concurrency. Lecture Notes in Computer Science, vol. 197.* Springer-Verlag, New York, 1987, pp. 389–449.
2. Berry, G., and Gonthier, G. The Esterel synchronous programming language. Design, semantics, implementation. *Sci. Comput. Prog.* To be published. (Also, INRIA Res. Rep. 842, 1988)

3. DRUSINSKY, D., AND HAREL, D. Using statecharts for hardware description. Tech. Rep. CS85-06, Dept. of Applied Mathematics, The Weizmann Institute of Science, Rehovot, Israel, 1985.
4. HAREL, D. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program. 8* (July 1987), 231–274.
5. HAREL, D. On visual formalisms. *Commun. ACM 31* (1988), 514–530.
6. HAREL, D., PNUELI, A., SCHMIDT, J. P., AND SHERMAN, R. On the formal semantics of statecharts. In *Proceedings of the 2nd IEEE Symposium on Logic in Computer Science* (Ithaca, N.Y., July 1987) IEEE Press, New York, 1987, pp. 54–64.
7. HAREL, D., LACHOVER, H., NAAMAD, A., PNUELI, A., POLITI, M., SHERMAN, R., SHTULL-TRAURING, A., AND TRAKHTENBROT, M. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Trans. Softw. Eng. 16*, 4 (Apr. 1990), 403–414.
8. HEYDON, A., MAIMONE, M. W., TYGAR, J. D., WING, J. M., AND ZAREMSKI, A. M. Miró: Visual specification of security. *IEEE Trans. Softw. Eng. 16*, 10 (Oct. 1990), 1185–1197.
9. HUIZING, C., AND DEROEVER, W. P. Introduction to design choices in the semantics of statecharts, *Inf. Process. Lett. 37* (1991), 205–213.
10. HOOMAN, J. J. M., RAMESH, S. AND DE ROEVER, W. P. A compositional axiomatization of statecharts. *Theor. Comput. Sci. 101* (1992), 289–335.
11. HUIZING, C., GERTH, R., AND DEROEVER, W. P. Modeling statecharts behavior in a fully abstract way. In *Proceedings on the Colloquium on Trees in Algebra and Programming*. Lecture Notes in Computer Science, vol. 299. Springer-Verlag, New York, 1988, pp. 271–294.
12. KAHANA, C.-A. State overlapping in statecharts. M.S. thesis, Dept. of Mathematics and Computer Science, Bar-Ilan Univ., Ramat Gan, Israel, 1986 (in Hebrew).
13. PNUELI, A., AND SHALEV, M. What is in a step: On the semantics of statecharts. In *Proceedings of the Symposium on Theoretical Aspects of Computer Software*. Lecture Notes in Computer Science, vol. 526. Springer-Verlag, Berlin, 1991, pp. 244–264.

Received March 1990; revised December 1991 and July 1992; accepted July 1992