[44] Ramadge, P.J., "The complexity of some basic problems in the supervisory control of discrete event systems", in *Advanced Computing Concepts and Techniques in Control Engineering*, M.J. Denham and A.J. Laub, Eds., pp. 171-190, Springer-Verlag, New York, 1988.

[45] Ramadge, P.J., and Wonham, W.M., "Supervisory control of a class of discrete-event processes, *SIAM J. on Control and Optimization, 25*, pp. 206-230, 1987.

[46] Smedinga, R., "Control of Discrete Systems", Doctoral thesis, University of Groningen, 1989.

[47] Thistle, G., and Wonham W.M., "Control problems in temporal-logic framework", *Int. J Control, 44*, pp. 943-976, 1986.

[48] Tsitsiklis, "On the control of discrete event dynamical systems", *Math. of Control, Signals and Systems, 2*, pp. 95-107, 1989.

[49] Wonham, W.M., and Ramadge, P.J., "On the supremal controllable sublanguage of a given language", *SIAM J. on Control and Optimization, 25*, pp. 637-659, 1987.

[50] Wonham, W.M., and Ramadge, P.J., "Modular supervisory control of discrete event systems", *Math. of Control, Signals and Systems, 1*, pp. 13-30, 1988.

[51] Zhong, H., and Wonham W.M., "On hierarchical control of discrete event systems", *22nd Ann. Conf. on Information Sci. and Systems*, Princeton, N.J., pp. 64-70, 1988

[29] Lin, F. and Wonham, W.M., "Decentralized supervisory control of discrete event systems", *Information Sciences, 44*, pp. 199-224, 1988.

[30] Loeckx, J., and Seiber, K., *The foundations of program verification*, 2nd ed., John Wiley & sons, New York, 1987.

[31] Manna, Z. and Pnueli A., "The anchored version of the temporal framework", *Springer LNCS 354*, pp. 201-284, 1989.

[32] Milne, G.J., "CIRCAL and the representation of communication, concurrency, and time", /em ACM TOPLAS, 7, pp.270-298, 1985.

[33] Milner, R., "A calculus for communicating systems", *Springer LNCS 92*, 1980.

[34] Olderog, E.-R. "Process theory: Semantics, Specification and Verification", in /em Springer LNCS 224, pp.442-509, 1986.

[35] Olderog E.-R., and Hoare C.A.R., "Specification-oriented semantics for communicating processes", *Acta Informatica, 23*, pp. 9-66, 1986.

[36] Ozveren, C.M., Willsky, A.S., and Antsaklis, P.J., "Stability and stabilizability of discrete event dynamic systems", preprint, 1989.

[37] Ozveren, C.M., and Willsky, A.S., "Observability of discrete event dynamic systems", preprint, 1989.

[38] Ozveren, C.M., and Willsky, A.S., "Aggregation and multi-level control in discrete event dynamic systems", preprint, 1989.

[39] Peterson, J.L., *Petry nets and modeling of systems*, Prentice-Hall, Englewood Cliffs, 1981.

[40] Pnueli, A., "Application of temporal logic to the specification and verification of reactive systems: a survey of current trends", *Springer LNCS 224*, pp.510-584, 1986.

[41] Pnueli, A., "Linear and branching structures in the semantics and logics of reactive systems, *Springer LNCS 194*, pp.15-32, 1985.

[42] Ramadge, P.J., "Observability of discrete event systems", *Proc. 25th IEEE Conf. Decision and Control*, pp. 1108-1112, 1986.

[43] Ramadge, P.J., "Some tractable problems in the supervisory control of discrete event systems described by Buchi automata", *IEEE Trans. Aut. Control, AC-34*, pp. 10-19, 1989.

[14] Golaszewski, C.H., and Ramadge, P.J., "Control of discrete event processes with forced events", *Proc. 26th IEEE Conf. on Decision and Control*, pp. 247-251, Los Angeles, December 1987.

[15] Golaszewski, C.H., and Ramadge, P.J., "Mutual Exclusion problems for discrete event systems with shared events", *Proc. 27th IEEE Conf. Decision and Control*, pp. 234-239, Dec. 1988.

[16] Golaszewski, C.H., and Ramadge, P.J., "Discrete event processes with arbitrary controls", in *Advanced Computing Concepts and Techniques in Control Engineering*, M.J. Denham and A.J. Laub, Eds., pp. 459-469, Springer-Verlag, New York, 1988.

[17] Harel, D., "Statecharts: A visual formalism for complex systems", *Science of Computer Programming, 8*, pp. 231-274, 1987.

[18] Hennessy, M. *Algebraic Theory of Processes*, MIT Press, Cambridge, Ma., 1988.

[19] Heymann, M., and Meyer, G., "An algebra of discrete event processes", NASA TM, 1989.

[20] Hoare C.A.R., *Communicating sequential processes*, Prentice-Hall International, 1985.

[21] Hopcroft, J.E., and Ullman, J.D., *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading MA.,1979.

[22] Inan, K., and Varaya, P., "Finitely recursive process models for discrete event systems", *IEEE Trans. on Automatic Control, AC-33*, pp. 626-639, 1988.

[23] Inan, K., and Varaya, P., "Algebras of Discrete Event Models", *Proc. IEEE, 77*, pp. 24-38, 1989.

[24] Kroger, F., *Temporal logic of programs*, Springer Verlag, New York, 1987.

[25] Lafortune, S., "Modeling and analysis of transition execution in database systems", *IEEE Trans on Aut. Control, AC-33*, pp. 439-447, 1988.

[26] Lafortune, S., and Wong, E., "Astate model for the concurrency control problem in database management systems", *Proc 24th IEEE Conf. on Decision and Control*, 1985.

[27] Li, Y., and Wonham W.M., "On supervisory control of real-time Discrete-event systems", *Information sciences, 46*, pp. 159-183, 1988.

[28] Lin, F. and Wonham, W.M., "On observability of discrete event systems", *Information Sciences, 44*, pp. 173-198, 1988.

[2] de Bakker, J.W., de Roever,W.-P., and Rozenberg, G., eds., "Current trends in concurrency", *springer LNCS 224*, 1986.

[3] de Bakker, J.W., de Roever,W.-P., and Rozenberg, G., eds., "Linear time, branching time and partial order in logics and models for Concurrency", *springer LNCS 354*, 1989.

[4] Bergstra, J.A. and Klop, J.W., "Process theory based on bisimulation semantics", *Springer LNCS 354*, pp. 50-122, 1989.

[5] Bergstra, J.A., Klop, J.W., and Olderog E.-R., "Readies and failures in the algebra of communicating processes", *SIAM J. on Computing, 17*, pp. 1134-1177, 1988.

[6] Berry, G., and Cosserat, I., "The ESTEREL synchronous programming language and its mathematical semantics", *Springer LNCS 197*, 1985.

[7] Brave, Y., and Heymann, M., "Formulation and control of a class of real-time discrete-event processes", EE Pub 714, Department of Electrical Engineering, Technion, Haifa, February, 1989. (see also *Proc. 27th IEEE Conf. Decision and Control*, pp. 1131-1132, Dec. 1988.)

[8] Brave, Y., and Heymann, M., "On Stabilization of Discrete-Event Processes", EE Pub 724, Department of Electrical Engineering, Technion, Haifa, April, 1989. (to appear in *Int. J. Control*).

[9] Brooks, S.D., and Roscoe, A.W., "An improved failures model for communicating processes" *Springer LNCS 197*, pp. 281-305, 1985.

[10] Brooks, S.D., Hoare C.A.R., and Roscoe, A.W., "A theory for communicating sequential processes", *J. ACM, 31*, pp. 560-599, 1984.

[11] Cho, H., and Marcus, S.I., "on supremal languages of classes of sublanguages that arise in supervisor synthesis problems with partial observation", *Math Control Signals Systems, 2*, pp. 47-69, 1989.

[12] Cieslak, R., Desclaux, C., Fawaz, A.,and Varaya, P., "Supervisory control of discrete event processes with partial observations", *IEEE Trans on Aut. Control, AC-33*, pp. 249-260, 1988.

[13] Emerson, E,-A., and Srinivasan, J., "Branching time temporal logic", *Springer LNCS 354*, pp. 123-172, 1989.

Proposition 14.1 *Let P be a process. The class of controllable sublanguages of $\mathcal{L}(P)$ is closed under set union.*

Consider now a process P, and let Δ, the deadlock process, serve as supervisor. The controlled process is then given as

$$(\Delta/P) = P_{\Sigma}||_{\Sigma_c}\Delta, \tag{180}$$

and it is clear from Proposition 156 that if S is any supervisor for P, then

$$\mathcal{L}(\Delta/P) \subseteq (S/P). \tag{181}$$

Thus, the language (Δ/P) is the smallest controllable sublanguage of P. We denote this sublanguage by $\mathcal{S}_P$ and call it the *spontaneous* language of P.

Theorem 14.3 *Let P be a process. Let $\mathcal{K} \subseteq \mathcal{L}(P)$ be a nonempty closed sublanguale. If $\mathcal{S}_P \subseteq \mathcal{K}$, then $\mathcal{K}$ contains a unique nonempty supremal controllable sublanguage.*

Let $\mathcal{K} \subseteq \Sigma^*$ be a closed language. Two traces $s_1, s_1 \in \mathcal{K}$ are called *(Nerode) equivalent*, if for all $t \in \mathcal{K}$, $s_1\hat{}t \in \mathcal{K} \;\Leftrightarrow\; s_2\hat{}t \in \mathcal{K}$. Thus, two traces of $\mathcal{K}$ are equivalent if they have the same continuations in $\mathcal{K}$. We denote the (Nerode) equivalence-class of a trace $s \in \mathcal{K}$ by $[s]_{\mathcal{K}}$ or, when no confusion can arise, simply by $[s]$. With the aid of the Nerode-equivalence relation, it is easy to construct a state-transition graph (actually state-transition tree) as follows. Let the state set Q be identified with the set of all equivalence classes of $\mathcal{K}$. If $q, q' \in Q$ are two states such that $q = [s]$ and $q' = [t]$, then there is an edge or transition labeled σ from q to q' if $s\sigma \in [t]$. We shall call this transition graph associated with $\mathcal{K}$ *canonical.*

Definition 14.2 Let $\Theta \subseteq \Sigma$ be a given subset. A closed language $\mathcal{K} \subseteq \Sigma^*$ is called Θ-*flat* if for any $s \in \mathcal{K}$ and $\sigma_1, \sigma_2 \in \mathcal{K}$

$$s\sigma_1, s\sigma_2 \in \mathcal{K} \;\Rightarrow\; [s\sigma_1] = [s\sigma_2]. \tag{182}$$

It is clear that if $\mathcal{K}$ is a Θ-flat closed language then the canonical state graph G of $\mathcal{K}$ has the property that, given any state q of G, then all state-transitions labeled with events from Θ, lead to the same target state q' (if the set of such transitions at q is nonempty).

References

[1] Benveniste, A., Le Goff, B., and Le Guernic, "Hybrid dynamical systems theory and the language "SIGNAL"", Internal Publication No. 403, IRISA/INRIA, Rennes Cedex, France, April 1988.

supervisor for a process P, then

$$\mathcal{L}(S/P) = \mathcal{L}(P_\Sigma||_{\Sigma_c}S) \subseteq \mathcal{L}(P). \tag{176}$$

We can now introduce the concept of controllable languages.

Definition 14.1 Let $\mathcal{K}$ be a closed sublanguage of $\mathcal{L}(P)$. $\mathcal{K}$ is said to be *controllable* if and only if there exists a supervisor S such that

$$\mathcal{K} = \mathcal{L}(P_\Sigma||_{\Sigma_c}S). \tag{177}$$

The following theorem is an abstract characterization of controllable languages. (Recall that for a closed language $\mathcal{K}$, $det(\mathcal{K})$ is the deterministic process whose language is $\mathcal{K}$ as defined in Section 12.6.)

Theorem 14.1 *A closed language $\mathcal{K}$ is controllable if and only if*

$$\mathcal{L}(P_\Sigma||_{\Sigma_c}det(\mathcal{K})) = \mathcal{K}. \tag{178}$$

Proof. If (178) holds, then $det(\mathcal{K})$ can serve as supervisor and there is nothing to prove. Conversely, suppose that $\mathcal{K}$ is controllable. We need to show that $det(\mathcal{K})$ satisfies (178), i.e., that $det(\mathcal{K})$ can be used as supervisor. Suppose S is a supervisor such that $R = P_\Sigma||_{\Sigma_c}S$ and $\mathcal{L}(R) = \mathcal{K}$. Then (178) follows easily from Proposition 13.13, which implies that $\mathcal{L}(P_\Sigma||_{\Sigma_c}R) = \mathcal{L}(R)$ and Corollary 13.3, which implies that $\mathcal{L}(P_\Sigma||_{\Sigma_c}det(\mathcal{K})) = \mathcal{L}(P_\Sigma||_{\Sigma_c}R)$.

∎

A more concrete characterization of controllability (which was actually used as definition of controllability by Wonham and Ramadge in [45]) is the following corollary to the above theorem:

Theorem 14.2 *A closed sublanguage $\mathcal{K} \subseteq \mathcal{L}(P)$ is controllable if and only if for all traces $t\sigma \in \mathcal{L}(P)$ such that $t \in \mathcal{K}$ and $\sigma \in \Sigma$,*

$$t\sigma \notin \mathcal{K} \ \Rightarrow \ \sigma \in \Sigma_c. \tag{179}$$

As an immediate consequence of Theorem 13.6 we have the following of P but their occurrence is possible only if they are not disabled by S. We model this participation of S in events of Σ_c by allowing them also to take place in S. Thus events of Σ_c will take place if and only if both processes P and S execute them concurrently. Finally, events in Σ_d cannot occur without being triggered by S. Thus, they must be in the priority set B of S. If an event $\sigma \in \Sigma_d$ is also in A, then we interpret it as a *closed-loop* a closed-loop driven event, meaning that the controller waits for the occurrence of the event in P before it proceeds with its own transitions. Thus, closed-loop driven events are, in so far as concurrancy is concerned, indistinguishable from events of Σ_u (certain other restrictions must be imposed for physical realizability (see e.g. [7, 14]) but this will be of no concern to us here). However, events of Σ_d can be triggered also open-loop, and hence will occur in the concurrent process whenever they are triggered by S (whether or not they also occur in P) and will occur in P (concurrently) whenever they are possible in P at the time. Open-loop driven events thus differ from events in Σ_c in that they can take place in the concurrent process even if the process P fails to participate. It is easily seen that open-loop driven events play the same formal role in the supervisor as uncontrollable events in the process.

The controlled, or *closed-loop*, process is then given by

$$R = (S/P) := P_A||_B S. \tag{174}$$

The enablement-disablement control mechanism was introduced by Wonham and Ramadge in [45] while the (closed-loop) control mechanism with driven events was introduced by Golaszevski and Ramadge in [14] and, (in a real-time control setting) by Brave and Heymann [7] (and was called there *forcing*. The open-loop control mechanism with driven events is new.

We identify process *behavior* with the language that it generates. Thus, a behavioral specification is, typically, a statement about languages. If $\Sigma_s \subseteq \Sigma$ is some event subset, then a *local* specification might consist of a pair of languages $\underline{\mathcal{K}_s}$, $\overline{\mathcal{K}_s} \subseteq {\Sigma_s}^*$ such that $\mathcal{L}(P\backslash_{\Sigma-\Sigma_s})$, the localization of the process language to Σ_s satisfies the constraint

$$\underline{\mathcal{K}_s} \subseteq \mathcal{L}(P\backslash_{\Sigma-\Sigma_s}) \subseteq \overline{\mathcal{K}_s}. \tag{175}$$

Frequently, $\underline{\mathcal{K}_s} = \emptyset$ and the specification might consist of the upper bound constraint only. If $\Sigma_s = \Sigma$, we call the corresponding specification *global*.

In the remainder of the present section we confine our attention to the Wonham-Ramdage framework of control [45]. Thus we shall assume henceforth that $\Sigma = \Sigma_u \cup \Sigma_c$, $A := \Sigma_u \cup \Sigma_c = \Sigma$ and $B = \Sigma_c$. In view of Proposition 13.18 it is then clear that if S is a

The process $p(P)$ is called the *image* of P under p. We can also define the *inverse image* of a process Q under a projection operator p as the union of all processes P such that $p(P) = Q$, that is

$$p^{-1}(Q) := \bigcup (P \mid p(P) = Q),$$

or, alternatively,

$$p^{-1}(Q) := \{e \in \mathcal{O}_\Sigma \mid p(e) \in Q\}. \tag{172}$$

The proof that the set $p^{-1}(Q)$ as defined above satisfies the conditions of Definition 12.1 is straightforward and is left to the reader.

14 Control

The control of a discrete event process is accomplished through its interaction with the environment. Thus, we think of the environment as being capable of influencing the occurrence of certain events in the process under consideration. In particular, if the environment is itself a process, called a *supervisor*, control can be achieved by the (prioritized) parallel composition of the process and the supervisor. To make this idea precise, something must be said about the events that participate in the control process. Thus, the event set Σ is partitioned into three disjoint subsets

$$\Sigma = \Sigma_u \cup \Sigma_c \cup \Sigma_d, \tag{173}$$

where Σ_u is the subset of *uncontrollable* events that occur spontaneously in the process and cannot be disabled by the environment, Σ_c is the set of *controllable* events that occur spontaneously in the process but can be diasabled by the environment (and hence can be thought of as requiring the participation of the environment), and finally Σ_d is the set of *driven* events that in order to take place must be triggered, or forced, by the environment. To make this event classification mathematically more precise, we require that the priority sets A and B of the process P and the supervisor S, respectively, satisfy the conditions that $\Sigma_u \cup \Sigma_c \subseteq A$ and $\Sigma_c \cup \Sigma_d = B$.

Our interpretation of these priority sets is then as follows: Events in Σ_u are always initiated by and occur spontaneously in P. If they can also occur in the supervisor S, this occurrence will be interpreted as being triggered by P and hence their occurrence in S will be assumed to coincide with that in P. But if the supervisor cannot execute the corresponding event in Σ_u, it will still take place in P disregarding S. Events in Σ_c are spontaneous events

To this end define for a trajectory $e \in P$

$$e_p := (p^{-1}p(X_0), (\sigma_1, p^{-1}p(X_1)) \dots (\sigma_k, p^{-1}p(X_k))), \tag{170}$$

where $p^{-1}(X)$ denotes the inverse image of X under p.The definition of $p(P)$ is then as follows:

$$p(P) := \{p(e) \in \mathcal{O}_\Sigma \mid e_p \in P\}. \tag{171}$$

Notice that $e_p \in P$ implies that $e \in P$ in view of the fact that the inclusion $X \subseteq p^{-1}p(X)$ always holds. (The converse implication does not hold in general.)

Proposition 13.19 *The set $p(P)$ is a process.*

Proof. As usual, we need to show that conditions C1-C7 of Definition 12.1 are all satisfied. Thus, note first that for the trajectory $e = (\emptyset, \varepsilon)$, $p(e) = (\emptyset, \varepsilon)$. Also, $p^{-1}(\emptyset, \varepsilon) = (\emptyset, \varepsilon)$, whence $(\emptyset, \varepsilon) \in p(P)$ and condition C1 holds. To see that condition C2 holds, suppose that for some trajectory $p(e) \in p(P)$ $(e \in P)$ and some i, $p(\sigma_i) \in p(X_{i-1})$. Then $\sigma_i \in p(X_{i-1}$ which is impossible since by definition of $p(P)$, $p(e) \in p(P)$ only if $e_p \in P$. Hence condition C2 holds. We turn now to condition C6 and proceed as follows. Assume that

$$\begin{aligned} p(e) &= (p(X_0), (p(\sigma_1), p(X_1)) \dots (p(\sigma_k), p(X_k))) \in p(P) \\ p(h) &= (p(X_0), (p(\sigma_1), p(X_1)) \dots (p(\sigma_j), p(X_j))(p(\sigma), \emptyset)) \notin p(P). \end{aligned}$$

Then by (171)

$$\begin{aligned} e_p &= (p^{-1}p(X_0), (\sigma_1, p^{-1}p(X_1)) \dots (\sigma_k, p^{-1}p(X_k))) \in P \\ h_p &= (p^{-1}p(X_0), (\sigma_1, p^{-1}p(X_1)) \dots (\sigma_j, p^{-1}p(X_j))(\sigma, \emptyset)) \notin P. \end{aligned}$$

Upon applying condition C6 to P, we then conclude that

$$g_p := (p^{-1}p(X_0), (\sigma_1, p^{-1}p(X_1)) \dots (\sigma_j, p^{-1}p(X_j \cup \{\sigma\})) \dots (\sigma_k, p^{-1}p(X_k))) \in P.$$

Thus

$$g := (p(X_0), (p(\sigma_1), p(X_1)) \dots (p(\sigma_j), p(X_j \cup \{\sigma\})) \dots (p(\sigma_k), p(X_k))) \in p(P)$$

and condition C6 holds. The remaining conditions are straightforward.

∎

Finally, the following relation between parallel composition and internalization will be important to our study of control:

Proposition 13.18 $\mathcal{L}((P_A||_B Q)\backslash_{\Sigma-A}) \subseteq \mathcal{L}(P\backslash_{\Sigma-A})$.

Proof. Since for a process P, we can identify $\mathcal{L}(P)$ with $P_{\mathcal{T}}$, its set of free trajectories (see Section 12.6), we need to consider only the free trajectories of P and Q. We proceed by induction on trajectory length. The trajectory $(\emptyset, \varepsilon)$ is in every process. So assume that the proposition holds for all trajectories of length up to and including k. We shall show that the proposition holds also for trajectories of length $k+1$. Let $(\emptyset, w) \in (P_A||_B Q)\backslash_{\Sigma-A})$ be a free trajectory of length k and, for $\sigma \in A$, assume that $(\emptyset, w\hat{\ }(\sigma, \emptyset)) \in (P_A||_B Q)\backslash_{\Sigma-A}$. We wish to show that this implies that $(\emptyset, w\hat{\ }(\sigma, \emptyset)) \in P\backslash_{\Sigma-A}$ as well. Let E be the subset of all free trajectories $e \in P$ for which there exist $f \in Q$ satisfying

$$(\emptyset, w) = e\backslash_{\Sigma-A} \in (e_A||_B f)\backslash_{\Sigma-A}.$$

(Notice that free trajectories are always valid.) By assumption, the set E is nonempty and for some $e \in E$ there exists $f \in Q$, such that $(\emptyset, w\hat{\ }(\sigma, \emptyset)) \in (e_A||_B f)\hat{\ }(\sigma, \emptyset)\backslash_{\Sigma-A}$. Since $\sigma \in A$ it then follows from Definition 13.1, that $e\hat{\ }(\sigma, \emptyset) \in P$, concluding the proof.

∎

13.5 Projection Operator

Let $p : \Sigma \to \Sigma$ be a map, and extend p to a map $\hat{\Sigma} \to \hat{\Sigma}$ by defining $p(\Uparrow) = \Uparrow$ and $p(\Downarrow) = \Downarrow$. Let $P \in \mathcal{P}_\Sigma$ be a process and let

$$e = (X_0, (\sigma_1, X_1) \ldots (\sigma_k, X_k)) \in P$$

be a trajectory. Then the map p is extended to trajectories by letting

$$p(e) = (p(X_0), (p(\sigma_1), p(X_1)) \ldots (p(\sigma_k), p(X_k))), \tag{168}$$

where for a subset $X \subseteq \Sigma_1$, $p(X) := \{p(x) | x \in X\}$. The map p is applied to execution strings in a similar way. We call this operator p a *projection operator*. We wish to define $p(P)$ to be the process whose execution strings are $p(w)$ whenever the execution strings of P are w, that is,

$$P \overset{w}{\to} Q \;\Rightarrow\; p(P) \overset{p(w)}{\to} p(Q). \tag{169}$$

Lemma 13.2 *Let $e := (X, \varepsilon) \in (a \to P)$. If $e\backslash_{\{a\}}$ $(= e)$ $\in (a \to P)\backslash_{\{a\}}$, then $e \in P$ and $e\backslash_{\{a\}} \in P\backslash_{\{a\}}$.*

Proof. Let $(X, \varepsilon) \in (a \to P)$. Then $a \notin X$ and, by condition (i) of Definition 13.2, $(X, \varepsilon)\backslash_{\{a\}}$ $(= (X, \varepsilon))$ $\in (a \to P)\backslash_{\{a\}}$ if and only if for every $\{a\}$–stabilizing (right) execution string w such that $(X, w) \in (a \to P)$, $(X, w^{\{a\}})$ is valid. But each such execution string w can be written as $w = ((a, Z)\hat{\ } v)$, where (Z, v) is a trajectory of P. Since v is clearly also an $\{a\}$–stabilizing (right) execution string, and $w^{\{a\}} = v^{\{a\}}$, it follows from condition (i) of Definition 13.2 (applied now to P) that if $(X, \varepsilon) \in P$ then $(X, \varepsilon)\backslash_{\{a\}} \in P\backslash_{\{a\}}$. It remains to be shown that $(X, \varepsilon) \in P$. If $(X, \varepsilon) \notin P$, then there exists $Y \subseteq X$ and $x \in X - Y$ such that $(Y, \varepsilon) \in P$ and $(Y \cup \{x\}, \varepsilon) \notin P$. By Condition C6 of Definition 12.1 this implies that $(Y, (x, \emptyset)) \in P$ which in turn implies that $(X, (a, Y)(x, \emptyset)) \in (a \to P)$. Since $x \in X$, it then follows that $(X, (x, \emptyset))$ is not valid. But this is impossible since $(X, \varepsilon)\backslash_{\{a\}} \in (a \to P)\backslash_{\{a\}}$, concluding the proof.

∎

Proof of Proposition 13.15. We shall prove the case when $b = a$. The case when $b \neq a$ is straightforward. First note that as an immediate consequence of Lemma 13.1 we have that $P\backslash_{\{a\}} \subseteq (a \to P)\backslash_{\{a\}}$. Thus, we only need to prove the reverse inclusion, i.e., that $(a \to P)\backslash_{\{a\}} \subseteq P\backslash_{\{a\}}$. By Lemma 13.2, this is true for all trajectories of the form $e = (X, \varepsilon) \in (a \to P)$. As a further consequence of Lemma 13.2 it follows that for every trajectory $e = ((X, a)(Y_1, \sigma_2) \ldots (Y_{k-1}, \sigma_k), Y_k) \in (a \to P)$, such that $e\backslash_{\{a\}} \in (a \to P)\backslash_{\{a\}}$, there exists $\widehat{Y_1} \subseteq X$ such that $f = ((\widehat{Y_1}, \sigma_2) \ldots (Y_{k-1}, \sigma_k), Y_k) \in P$, $f\backslash_{\{a\}} \in P\backslash_{\{a\}}$ and $f\backslash_{\{a\}} = e\backslash_{\{a\}}$. This concludes the proof.

∎

The internalization operator distributes over the internal choice:

Proposition 13.16 $(P \oplus Q)\backslash_{\{a\}} = P\backslash_{\{a\}} \oplus Q\backslash_{\{a\}}$.

The internalization operator does not distribute over the external choice operator but the following simplifying equation holds.

Proposition 13.17 $((a \to P) + Q)\backslash_{\{a\}} = P\backslash_{\{a\}} \oplus (P + Q)\backslash_{\{a\}}$.

with $w'\backslash_\Theta$ as defined below. We can rewrite w' as

$$w' = ((X_0, (\sigma_1, X_1) \ldots (\sigma_{l-1}, X_{l-1})), \sigma_l) = (e', \sigma_l)$$

where

$$e' = (X_0, (\sigma_1, X_1) \ldots (\sigma_{l-1}, X_{l-1})).$$

We now have

$$w'\backslash_\Theta := (e'\backslash_\Theta, \sigma_l), \tag{167}$$

where $e'\backslash_\Theta$ is the $\Theta-$internalization of e' as defined above.

Example 13.7 Let $\Sigma = \{a, b, c, d\}$, $\Theta = \{a\}$,and

$$w = (\{c, d\}, a)(\{b, d\}, a)(\{a, b, c\}, d)(\{b, c, d\}, a).$$

Then,

$$w\backslash_\Theta = (\{b, c, d\}, d).$$

Proposition 13.14 $P \overset{w}{\to} Q \;\Rightarrow\; (P\backslash_\Theta) \overset{w\backslash_\Theta}{\to} (Q\backslash_\Theta)$.

The above proposition implies, just as we might expect, that events in Θ occur spontaneously whenever they can and the process $P\backslash_\Theta$ undergoes corresponding unobserved (silent) transitions.

Proposition 13.15

$$(b \to P)\backslash_{\{a\}} = \begin{cases} P\backslash_{\{a\}} & \textit{if } b = a \\ (b \to P\backslash_{\{a\}}) & \textit{if } b \neq a \end{cases}$$

For the proof of Proposition 13.15 we shall make use of the following lemmas.

Lemma 13.1

(i) $((\emptyset, a)\hat{\ }w, X)\backslash_{\{a\}} = (w, X)\backslash_{\{a\}}$.
(ii) $((\emptyset, a)\hat{\ }w, X)\backslash_{\{a\}} \in (a \to P)\backslash_{\{a\}} \;\Leftrightarrow\; (w, X)\backslash_{\{a\}} \in P\backslash_{\{a\}}$.

Theorem 13.8 *The internalization operator is well defined.*

Proof. We need to show that for a process P, the set $P\backslash_\Theta$ is a process, and thus satisfies the conditions of Definition 12.1. Conditions C1-C5 as well as C7 are straightforward and we shall prove condition C6.

Let $e\backslash_\Theta = (Y_0, (\sigma_1, Y_1) \dots (\sigma_k, Y_k)) \in P\backslash_\Theta$ and assume that for some $j : 0 \leq j \leq k$ and some $\sigma \in \Sigma - \Theta - Y_j$, $(Y_0, (\sigma_1, Y_1) \dots (\sigma_j, Y_j)(\sigma, \emptyset)) \notin P\backslash_\Theta$. We need to show that this implies that $(Y_0, (\sigma_1, Y_1) \dots (\sigma_j, Y_j \cup \{\sigma\}) \dots (\sigma_k, Y_k)) \in P\backslash_\Theta$.

Write $e = (X_0, u\hat{\ }w)$, where $u = (\rho_1, X_1) \dots (\rho_l, X_l)$, $w = (\rho_{l+1}, X_{l+1}) \dots (\rho_m, X_m)$, and $(X_0, u)\backslash_\Theta = (Y_0, (\sigma_1, Y_1) \dots (\sigma_j, Y_j))$. From the assumption that $(Y_0, (\sigma_1, Y_1) \dots (\sigma_j, Y_j)(\sigma, \emptyset)) \notin P\backslash_\Theta$, we can conclude, upon making use of Definition 13.2 (iii), that $(X_0, u\hat{\ }(\sigma, \emptyset)) \notin P$. But, upon applying condition C6 of Definition 12.1 to P, this implies that $\widehat{e} = (X_0, (\rho_1, X_1) \dots (\rho_l, X_l \cup \{\sigma\}) \dots (\rho_m, X_m) \in P$. Consequently, $\widehat{e}\backslash_\Theta = (Y_0, (\sigma_1, Y_1) \dots (\sigma_j, X_j \cup \{\sigma\}) \dots (\sigma_k, X_k)) \in P\backslash_\Theta$ and the proof is complete.

∎

The following property is also easily proved

Theorem 13.9 *For subsets* $\Theta_1, \Theta_2 \subseteq \Sigma$, $(P\backslash_{\Theta_1})\backslash_{\Theta_2} = (P\backslash_{\Theta_2})\backslash_{\Theta_1} = P\backslash_{\Theta_1 \cup \Theta_2}$.

We turn now to the operational (transition) behavior of the internalization operator. First we need to define the internalization of (left) execution strings. If w is an execution string of the process P, we shall denote by $w\backslash_\Theta$ the corresponding execution string of $P\backslash_\Theta$ (after internalization). Let

$$w = (X_0, \sigma_1) \dots (X_{k-1}, \sigma_k).$$

If $\sigma_i \in \Theta$ for all $i = 1, \dots, k$, we define $w\backslash_\Theta := \varepsilon$, the empty string. In the general case, write $w = w'\hat{\ }w''$, where

$$w' = (X_0, \sigma_1) \dots (X_{l-1}, \sigma_l),$$
$$w'' = (X_l, \sigma_{l+1}) \dots (X_{k-1}, \sigma_k),$$

with $\sigma_l \in \Sigma - \Theta$, and $\sigma_i \in \Theta$ for all $i = l+1, \dots, k$. We define $w\backslash_\Theta$ as follows:

$$w\backslash_\Theta := (w'\hat{\ }w'')\backslash_\Theta := (w'\backslash_\Theta)\hat{\ }(w'')\backslash_\Theta) = (w'\backslash_\Theta)\hat{\ }\varepsilon = w'\backslash_\Theta,$$

Definition 13.2 For a process P, the process $P\backslash_\Theta$ is given as follows:

(i) $(X_0, \varepsilon)\backslash_\Theta \in P\backslash_\Theta \quad \Leftrightarrow \quad$ either (1) $(X_0 \cup \{\Uparrow\}, \varepsilon) \in P$, or (2) $(X_0, \varepsilon) \in P$ and for every Θ-stabilizing execution string v such that $(X_0, v) \in P$, (X_0, v^Θ) is valid.

(ii) $(X_0 \cup \{\Uparrow\}, \varepsilon)\backslash_\Theta \in P\backslash_\Theta \quad \Leftrightarrow \quad$ either (1) $(X_0 \cup \{\Uparrow\}, \varepsilon) \in P$, or (2) $\exists$ a Θ-divergent sequence of execution strings $\{w^i\}$ such that $(X_0, w^i) \in P$ for all i.

(iii) If $e\backslash_\Theta \in P\backslash_\Theta$, where $e = (X_0, w) \in P$, and if $f = (X_0, w\hat{}(\sigma_{k+1}, X_{k+1})) \in P$, Then $f\backslash_\Theta \in P\backslash_\Theta \Leftrightarrow$ either (1) $f^\Uparrow := (X_0, w\hat{}(\sigma_{k+1}, X_{k+1} \cup \{\Uparrow\})) \in P$, or (2) $f\backslash_\Theta$ is valid, and $(X_0, w\hat{}(\sigma_{k+1}, X_{k+1})\hat{}v^\Theta)$ is valid for every Θ-stabilizing execution string v such that $(X_0, w\hat{}(\sigma_{k+1}, X_{k+1})\hat{}v) \in P$.

(iv) If $e\backslash_\Theta \in P\backslash_\Theta$, where $e = (X_0, w) \in P$, and if $f = (X_0, w\hat{}(\sigma_{k+1}, X_{k+1})) \in P$, Then $f^\Uparrow\backslash_\Theta \in P\backslash_\Theta \Leftrightarrow$ either (1) $f^\Uparrow \in P$, or (2) $\exists$ a Θ-divergent sequence of execution strings $\{w^i\}$ such that $(X_0, w\hat{}(\sigma_{k+1}, X_{k+1})\hat{}w^i) \in P$ for all i.

Example 13.5 Let $\Sigma = \{a, b, c, d\}$, let $\Theta = \{a\}$ and let P be the following process:

$$\begin{aligned} P \quad = \quad & gen\{(\{c,d\}, (b, \{a,b,c,d\})), \\ & (\{c,d\}, (a, \{b,d\})(c, \{a,b,c,d\})) \\ & (\{c,d\}, (a, \{b,d\})(a, \{a,b,c\})(d, \{a,b,c,d\}))\}. \end{aligned}$$

Then

$$\begin{aligned} P\backslash_{\{a\}} \quad = \quad & gen\{(\emptyset, (b, \{b,c,d\})) \\ & (\{b\}, (c, \{b,c,d\})) \\ & (\{b,c\}, (d, \{b,c,d\})). \end{aligned}$$

Example 13.6 Let $\Sigma = \{a, b, c, d\}$, let $\Theta = \{a\}$ and let P be the following process:

$$\begin{aligned} P \quad = \quad & gen\{(\{c,d\}, (b, \{a,b,c,d\})), \\ & (\{c,d\}, (a, \{b,d\})(c, \{a,b,c,d\})) \\ & (\{c,d\}, (a, \{b,d\})(a, \{a,b,c\})(d, \{a,b,c,d\})) \qquad (165) \\ & (\{c,d\}, (a, \{b,d\})(a, \{b,c\})^*)\}. \end{aligned}$$

Then

$$P\backslash_{\{a\}} \quad = \quad \nabla \qquad (166)$$

We have the following

observed process, denoted $P\backslash_\Theta (\in \mathcal{P}_{\overline{\Sigma}})$, where $\Theta := \Sigma - \overline{\Sigma}$, we define below a internalization operator

$$\pi_\Theta \ : \ \mathcal{P}_\Sigma \ \rightarrow \ \mathcal{P}_{\overline{\Sigma}} \ : \ P \ \longmapsto \ P\backslash_\Theta \tag{163}$$

that deletes from P all events of P that belong to Θ.

To define the operator π_Θ we proceed as follows. (We shall use here right representations of trajectories.)

For a trajectory

$$e = (X_0, w) = (X_0, (\sigma_1, X_1) \ldots (\sigma_k, X_k)) \tag{164}$$

let $e\backslash_\Theta$ denote the trajectory obtained from e as follows:

(i) Delete from e all occurrences of event symbols that belong to Θ (both as executed events and refused events). Thus, each refusal set X_i becomes $X_i\backslash_\Theta$.

(ii) Replace all consecutive refusal sets whose associated execution event symbols have been deleted, by their union. That is, if (in e) $\sigma_{i-1} \in \Sigma - \Theta$ and $\sigma_i, \sigma_{i+1}, \ldots, \sigma_l \in \Theta$, then replace $X_{i-1}\backslash_\Theta$ by $(X_{i-1}\backslash_\Theta) \cup (X_i\backslash_\Theta) \cup \ldots \cup (X_l\backslash_\Theta)$.

Example 13.4 Let $\Sigma = \{a, b, c, d\}$, $\Theta = \{a, b\}$ and

$$e = (\{a, d\}, (b, \{c, d\})(a, \{b, c\})(d, \{a, b, d\})(c, \{a, b, c, d\})).$$

Then

$$e\backslash_\Theta = (\{c, d\}, (d, \{d\})(c, \{c, d\}).$$

Let $w = (\sigma_{j+1}, X_{j+1}) \ldots (\sigma_l, X_l)$ be an execution string. We say that w is Θ-*stabilizing* provided $\sigma_l \in \Sigma - \Theta$ and $\sigma_i \in \Theta$ for $i = j+1, \ldots, l-1$. If w is Θ-stabilizing, we define $w^\Theta := (\sigma_l, X_l)$. The execution string w is called Θ-*nonstabilizing* if $\sigma_i \in \Theta$ for all $i = j+1, \ldots, l$. A sequence of execution strings $\{w^i\}_{i=1}^\infty$ is called Θ-*divergent* if for each i, w^i is a proper prefix of w^{i+1} and all w^i are Θ-nonstabilizing.

Recall that a trajectory is *valid* if $\sigma_i \notin X_{i-1}$ for all i. We can now define the operator π_Θ inductively.

Proof. First note that

$$\begin{array}{rcll} R_B||_BQ & = & (P_A||_BQ)_B||_BQ & \text{(by definition)} \\ & = & P_A||_B(Q_B||_BQ) & \text{(by (145))} \\ & = & P_A||_BQ & \text{(by (143))} \\ & = & R. & \text{(by definition)} \end{array}$$

Thus, in view of Proposition 13.10 we have

$$\mathcal{L}(R) \;\; = \;\; \mathcal{L}(R_B||_BQ) \subseteq \mathcal{L}(R_A||_BQ).$$

To complete the proof we need to show that the reverse inclusion holds, i.e., that

$$\mathcal{L}(R_A||_BQ) \subseteq \mathcal{L}(R). \tag{158}$$

First observe that

$$\begin{array}{rcll} R & = & P_A||_BQ & \text{(by definition)} \\ & = & P_A||_B(Q_A|_BQ) & \text{(by (143))} \\ & = & (P_A||_AQ)_A||_BQ. & \text{(by (145))} \end{array} \tag{159}$$

Also, by definition of R,

$$R_A||_BQ = (P_A||_BQ)_A||_BQ. \tag{160}$$

In view of Proposition 13.10, we have

$$\mathcal{L}(P_A||_BQ) \subseteq \mathcal{L}(P_A||_AQ), \tag{161}$$

which, with the aid of (156) yields

$$(P_A||_BQ)_A||_BQ \subseteq (P_A||_AQ)_A||_BQ. \tag{162}$$

Upon combining (159), (160) and (162) we obtain (158), concluding the proof.

∎

13.4 Internalization Operator

Let $P \in \mathcal{P}_\Sigma$ be a process and suppose that the external observer can observe only those events of P that are in a proper subset $\overline{\Sigma} \subseteq \Sigma$. To obtain a suitable model for this partially

(i) There exists $e \in E(w)$ and $f \in F(e)$ such that $e\hat{~}(\sigma, \emptyset) \in P$ and $f\hat{~}(\sigma, \emptyset) \in Q$. Then

$$(\emptyset, w\hat{~}(\sigma, \emptyset)) \in (e\hat{~}(\sigma, \emptyset)_{A_1} ||_B f\hat{~}(\sigma, \emptyset)) \subseteq (P_{A_1} ||_B Q).$$

(ii) For all $e \in E(w)$ there exists no $f \in F(e)$ such that $f\hat{~}(\sigma, \emptyset) \in Q$. Then (153) implies that $\sigma \notin B$, whence

$$(\emptyset, w\hat{~}(\sigma, \emptyset)) \in (e\hat{~}(\sigma, \emptyset)_{A_1} ||_B f) \subseteq (P_{A_1} ||_B Q).$$

(iii) There is no $e \in E(w)$ such that $e\hat{~}(\sigma, \emptyset)) \in P$. In this case (153) implies that $\sigma \notin A_2$ and since $A_1 \subseteq A_2$, it follows that $\sigma \notin A_1$. Thus

$$(\emptyset, w\hat{~}(\sigma, \emptyset)) \in (e_{A_1} ||_B f\hat{~}(\sigma, \emptyset)) \subseteq (P_{A_1} ||_B Q).$$

This completes the proof.

∎

Proposition 13.11 *Let P and Q be processes. If $\mathcal{L}(P) \subseteq \mathcal{L}(Q)$ then, for all $A, B \subseteq \Sigma$,*

$$\mathcal{L}(P) \subseteq \mathcal{L}(P_A ||_B Q). \tag{155}$$

Proof. Elementary.

∎

Proposition 13.12 *Let $A, B \subseteq \Sigma$ be any subsets and let P_1, P_2 and Q be processes. Then*

$$\mathcal{L}(P_1) \subseteq \mathcal{L}(P_2) \quad \Rightarrow \quad \mathcal{L}(P_{1A} ||_B Q) \subseteq \mathcal{L})(P_{2A} ||_B Q). \tag{156}$$

An immediate consequence of the above Proposition is the following

Corollary 13.3 $\mathcal{L}(P_1) = \mathcal{L}(P_2) \quad \Rightarrow \quad \mathcal{L}(P_{1A} ||_B Q) = \mathcal{L}(P_{2A} ||_B Q)$.

Proposition 13.13 *Let P and Q be processes and let $A, B \subseteq \Sigma$ satisfy $A \subseteq B$. Let $R := P_A ||_B Q$. Then*

$$\mathcal{L}(R_A ||_B Q) = \mathcal{L}(R). \tag{157}$$

participation of the other process whenever possible. At the same time a process cannot execute events in the other process' priority set without the other process' participation.

- $B \subseteq A$. Events in $\Sigma - A$ can be executed by each process without interference by the other process but with synchronous participation of the other process whenever possible. Events in $A - B$ require participation of process P for execution while events in B require participation of both processes; otherwise deadlock will occur.

There is one additional special case that is not explicitly parametrized by A and B that deserves attention. Let $\Sigma = \Sigma_P \cup \Sigma_Q$ where Σ_P includes all events that process P can ever execute and similarly for Σ_Q. Then each event in $\overline{\Sigma} := \Sigma - \Sigma_P \cap \Sigma_Q$ can be executed by at most one of the two processes. Thus for events in $\overline{\Sigma}$ the parallel composition operator models event interleaving.

The following results will be useful in discussing certain control problems.

Theorem 13.7 *If P and Q are deterministic processes then for any $A, B \subseteq \Sigma$, the process $P_A||_B Q$ is deterministic as well.*

Proposition 13.10 *Let $A_1, A_2, B \subseteq \Sigma$ be any subsets and consider processes P and Q. If $A_1 \subseteq A_2$, then*

$$\mathcal{L}(P_{A_2}||_B Q) \subseteq \mathcal{L}(P_{A_1}||_B Q) \tag{152}$$

Proof. Since for a process P we can identify $\mathcal{L}(P)$ with $P_{\mathcal{T}}$, its set of free trajectories (see Section 12.6), we need to consider only the free trajectories of P and Q. We will proceed by induction on trajectory length. The trajectory $(\emptyset, \varepsilon)$ is in every process, so assume that the proposition holds for all free trajectories of length up to and including k. We shall show that it holds also for trajectories of length $k+1$. Let

$$(\emptyset, w\widehat{\ }(\sigma, \emptyset)) \in (P_{A_2}||_B Q), \tag{153}$$

where $(\emptyset, w) \in (P_{A_2}||_B Q)$ is a (free) trajectory of length k. Let $E(w)$ denote the set of all free trajectories $e \in P$ for which there exist $f \in Q$ such that

$$(\emptyset, w) \in (e_{A_2}||_B f). \tag{154}$$

For each $e \in E(w)$ let $F(e)$ denote the set of all trajectories f satisfying (154). To show that $(\emptyset, w\widehat{\ }(\sigma, \emptyset)) \in (P_{A_1}||_B Q)$, we must consider three cases:

The operational behavior of the operator ${}_A||_B$ is exhibited in a transparent way by the following transition formulas (we use the right representation):

Let (σ, X) be an execution. Then

$$P \xrightarrow{(\sigma,X)} P' \;\;\&\;\; Q \xrightarrow{(\sigma,X)} Q' \;\;\Rightarrow\;\; P_A||_B Q \xrightarrow{(\sigma,X)} P'_A||_B Q' \tag{149}$$

$$P \xrightarrow{(\sigma,X)} P' \;\;\&\;\; Q \xrightarrow{(\sigma,X)} \backslash \;\;\Rightarrow\;\; P_A||_B Q \xrightarrow{(\sigma,X)} \begin{cases} P'_A||_B Q & \text{if } \sigma \notin B \\ \backslash & \text{if } \sigma \in B \end{cases} \tag{150}$$

$$Q \xrightarrow{(\sigma,X)} Q' \;\;\&\;\; P \xrightarrow{(\sigma,X)} \backslash \;\;\Rightarrow\;\; P_A||_B Q \xrightarrow{(\sigma,X)} \begin{cases} P_A||_B Q' & \text{if } \sigma \notin A \\ \backslash & \text{if } \sigma \in A. \end{cases} \tag{151}$$

where the notation $P \xrightarrow{(\sigma,X)} \backslash$ means that the execution (σ, X) is not possible for P.

Equation (149) states that if a given execution (σ, X) is possible for both processes P and Q, then it will be executed simultaneously (i.e., in synchronization) in both processes. When an execution is initiated by one of the processes (and hence is of course possible in it) but is not possible in the other, the initiating process will execute the event on its own unless the event symbol is in the blocking, or priority, set of the other process, in which case the execution will be blocked leading to a deadlock.

The above definition of concurrency, being parametrized by the two priority sets, models a wide range of behaviors depending on the chosen values of the parameters. Let us consider a number of interesting special cases.

- $A = B = \Sigma$. Strict synchronization is obtained; events are executed if and only if they are possible in both processes, and deadlock occurs otherwise.
- $A = B = \emptyset$. This is the, so called, *broadcast synchronization* in which case each process can offer, at will, events for execution. If the other process can execute the offered event as well, they will execute it together in synchronization, otherwise the initiating process will execute the offered event by itself. Obviously, broadcast synchronization can never lead to deadlock.
- $A = B = S \neq \Sigma$. This parametrization models strict synchronization for events in S and broadcast synchronization for events in $\Sigma - S$.
- $A \cap B = \emptyset$ and $A \cup B = \Sigma$. In this case each process can execute events in its own priority set without interference of the other process with assurance of synchronous

Parallel composition distributes over the internal choice operator and we have the following:

Theorem 13.5 *For processes P, Q and R with priority sets A and B*

$$P_A||_B(Q \oplus R) = (P_A||_B Q) \oplus (P_A||_B R). \tag{146}$$

In fact, the above theorem generalizes to arbitrary sets of processes as follows

Theorem 13.6 *Let $\mathcal{D}$ ba an arbitrary set of processes. Then*

$$\bigcup_{P_\alpha \in \mathcal{D}} (P_{\alpha A}||_B Q) = (\bigcup_{P_\alpha \in \mathcal{D}} P_\alpha)_A||_B Q) \tag{147}$$

The fact that parallel composition does not, in general, distribute over the external choice operator is exhibited by the following simple

Example 13.3 Let $\Sigma = \{a, b, c\}$, let

$$\begin{aligned} P &= gen\{(\{b\},(a,\{a,b,c\})),(\{b\},(c,\{a,b,c\}))\}, \\ Q &= gen\{(\{b,c\},(a,\{a,b,c\})), \\ R &= gen\{(\{a,c\},(b,\{a,b,c\})), \end{aligned} \tag{148}$$

and $A = \{a, c\}, \quad B = \{a, b\}$.

Then

$$S := P_A||_B(Q + R) = gen \begin{cases} (\{\emptyset\},(a,\{a,b,c\})), \\ (\{\emptyset\},(b,\{a,b\})(c,\{a,b,c\})), \\ (\{\emptyset\},(c,\{a,c\})(b,\{a,b,c\})). \end{cases}$$

$$T := P_A||_B Q + P_A||_B R = gen \begin{cases} (\{\emptyset\},(a,\{a,b,c\})), \\ (\{\emptyset\},(c,\{a,b,c\})), \\ (\{\emptyset\},(b,\{a,b\})(c,\{a,b,c\})), \\ (\{\emptyset\},(c,\{a,c\})(b,\{a,b,c\})). \end{cases}$$

Thus, the process T can deadlock after an initial execution of c while the process S cannot.

¿From condition C3 we know that

$$\widehat{g} = (Z_0, (\alpha_1, Z_1) \ldots (\alpha_m, Z_m)) \in (\widehat{e}_A ||_B \widehat{f}) \subseteq P_A ||_B Q$$

for suitable prefixes

$$\begin{aligned} \widehat{e} &= (X_0, (\sigma_1, X_1) \ldots (\sigma_i, X_i)) = (X_0, \widehat{w}) \\ \widehat{f} &= (Y_0, (\mu_1, Y_1) \ldots (\mu_j, Y_j)) = (Y_0, \widehat{v}) \end{aligned}$$

of e and f, respectively. We must consider three cases in which (141) can hold.

(i) $(X_0, \widehat{w}\hat{}(\alpha, \emptyset)) \in P$, $(Y_0, \widehat{v}\hat{}(\alpha, \emptyset)) \notin Q$, and $\alpha \in B$. By applying condition C6 to Q, it follows that

$$f' = (Y_0, (\mu_1, Y_1) \ldots (\mu_j, Y_j \cup \{\alpha\})) \ldots (\mu_l, Y_l)) \in Q.$$

Furthermore, $\alpha \in B$ implies that $S(X_i, Y_j \cup \{\alpha\}) = S(X_i, Y_j) \cup \{\alpha\}$, and it follows that $Z_m \cup \{\alpha\} \subseteq S(X_i, Y_j \cup \{\alpha\})$. It now easily follows that

$$g' \in (e_A ||_B f') \ (\subseteq P_A ||_B Q). \tag{142}$$

(ii) $(X_0, \widehat{w}\hat{}(\alpha, \emptyset)) \notin P$, $(Y_0, \widehat{v}\hat{}(\alpha, \emptyset)) \in Q$, and $\alpha \in A$. This case is similar to case (i).

(iii) $(X_0, \widehat{w}\hat{}(\alpha, \emptyset)) \notin P$, $(Y_0, \widehat{v}\hat{}(\alpha, \emptyset)) \notin Q$. By applying condition C6 to P and to Q it follows that

$$\begin{aligned} e' &= (X_0, (\sigma_1, X_1) \ldots (\sigma_i, X_i \cup \{\alpha\}) \ldots (\alpha_k, X_k)) \in P \\ f' &= (Y_0, (\mu_1, Y_1) \ldots (\mu_j, Y_j \cup \{\alpha\}) \ldots (\mu_l, Y_l)) \in Q. \end{aligned}$$

Also, $S(X_i \cup \{\alpha\}, Y_j \cup \{\alpha\}) = S(X_i, Y_j) \cup \{\alpha\}$. It is now not difficult to show that

$$g' \in (e'_A ||_B f') \ (\subseteq P_A ||_B Q).$$

This completes the proof.

∎

The following properties of the parallel composition operator are not hard to prove.

Theorem 13.4 *For processes P, Q and R with priority sets A, B and C, respectively,*

$$\begin{aligned} P_A ||_B P &= P & (143) \\ P_A ||_B Q &= Q_B ||_A P & (144) \\ (P_A ||_B Q)_{A \cup B} ||_C R &= P_A ||_{B \cup C} (Q_B ||_C R). & (145) \end{aligned}$$

then by application of (iic) above it follows immediately (since $\Uparrow \in \Sigma_{td}$) that

$$pref_{j+1}(f) \in (e_A||_B pref_{j+1}(f^*)).$$

∎

We now have the following Theorem:

Theorem 13.3 *The process $P_A||_B Q$ is well defined.*

Proof. We need to show that the conditions of Definition 12.1 are satisfied. First note that $S(\emptyset, \emptyset) = \emptyset$, whence $(\emptyset, \varepsilon) \in P_A||_B Q$ and C1 holds. To prove C2, we proceed inductively. The condition holds trivially for $(e_A||_B f)$, where $e = (X, \varepsilon)$ and $f = (Y, \varepsilon)$. So assume that it holds for $(e_A||_B f)$ where e and f are given by (134) and (135), respectively. If for some $i: \ 0 \leq i \leq k$

$$(X_0, (\sigma_1, X_1) \ldots (\sigma_i, X_i \cup \{\Uparrow\})) \in P, \tag{139}$$

or if for some $j: \ 0 \leq j \leq l$

$$(Y_0, (\mu_1, Y_1) \ldots (\mu_j, Y_j \cup \{\Uparrow\})) \in Q, \tag{140}$$

then there is nothing more to show in view of condition C7 of Definition 12.1 (see also Proposition 13.9). So assume that (139) and (140) are false for all i and j, respectively, and let $e' \in P$ and $f' \in Q$ be given, respectively, by (136) and (137). Then $\sigma \notin X_k$ and $\mu \notin Y_l$ and we need to consider three cases. First, if $\sigma = \mu$, then $\sigma \notin X_k \cup Y_l$ and it follows immediately that $\sigma \notin S(X_k, Y_l)$. Thus, C2 holds for $(e'_A||_B f')$. Next, if $\sigma \notin B$ it again follows that $\sigma \notin S(X_k, Y_l)$ and C2 holds for $(e'_A||_B f)$. Finally, if $\mu \notin A$, then $\mu \notin S(X_k, Y_l)$ and C2 holds for $(e_A||_B f')$. This proves condition C2. Conditions C3, C4 and C5 are straightforward and condition C7 is proved in a manner similar to the proof of Proposition 13.9. We turn to condition C6. Assume that

$$g = (Z_0, (\alpha_1, Z_1) \ldots (\alpha_n, Z_n)) \in (e_A||_B f) \subseteq P_A||_B Q$$

for trajectories $e \in P$ and $f \in Q$ where e and f are given, respectively, by (134) and (135), and assume further that for some $m: \ 0 \leq m \leq n$ and some $\alpha \in \Sigma - Z_m$, the trajectory

$$(Z_0, (\alpha_1, Z_1) \ldots (\alpha_m, Z_m)(\alpha, \emptyset)) \notin P_A||_B Q. \tag{141}$$

We need to show that the above implies that

$$g' = (Z_0, (\alpha_1, Z_1) \ldots (\alpha_m, Z_m \cup \{\alpha\}) \ldots (\alpha_n, Z_n)) \in P_A||_B Q.$$

and let

$$e' = e\hat{}(\sigma, X_{k+1}) \in P, \tag{136}$$
$$f' = f\hat{}(\mu, Y_{l+1}) \in Q. \tag{137}$$

Then

(a) $(e'_A||_B f') :=$

$$\begin{cases} \{g = h\hat{}(\sigma, Z) \mid h \in (e_A||_B f) \ \ \& \ \ Z \subseteq S(X_{k+1}, Y_{l+1})\} & \text{if } \sigma = \mu \\ \text{undefined} & \text{otherwise} \end{cases}$$

(b) $(e'_A||_B f) :=$

$$\begin{cases} \{g = h\hat{}(\sigma, Z) \mid h \in (e_A||_B f) \ \ \& \ \ Z \subseteq S(X_{k+1}, Y_l)\} & \text{if } f\hat{}(\sigma, \emptyset) \notin Q \text{ and } \sigma \notin B, \text{ or} \\ & \text{if } \Uparrow \in X_k \\ \text{undefined} & \text{otherwise} \end{cases}$$

(c) $(e_A||_B f') :=$

$$\begin{cases} \{g = h\hat{}(\mu, Z) \mid h \in (e_A||_B f) \ \ \& \ \ Z \subseteq S(X_k, Y_{l+1}))\} & \text{if } e\hat{}(\mu, \emptyset) \notin P \text{ and } \mu \notin A, \text{ or} \\ & \text{if } \Uparrow \in Y_l \\ \text{undefined} & \text{otherwise} \end{cases}$$

Before proceeding with our discussion, let us note the following

Proposition 13.9 *For any process P and subsets $A, B \in \Sigma$*

$$P_A||_B \nabla = \nabla \tag{138}$$

Proof. Since, obviously, $P_A||_B\nabla \subseteq \nabla$, we only need to show that the reverse inclusion holds for an arbitrary process P. Let $f = (X_0, (\sigma_1, X_1) \dots (\sigma_k, X_k)) \in \nabla$ be any trajectory. Then $f \in comp(f^*)$ where $f^* = (\Sigma_{td}, (\sigma_1, \Sigma_{td}) \dots (\sigma_k, \Sigma_{td})) \in \nabla$. We will be done by showing that $f \in (e_A||_B f^*)$, where $e = (\emptyset, \varepsilon) \in P$ (arbitrary P). Indeed, by employing (i) above we obtain that

$$pref_0(f) = (X_0, \varepsilon) \in (e_A||_B pref_0(f^*)) = \{(Z, \varepsilon) : \ Z \subseteq \Sigma_{td}\}.$$

Proceeding inductively, it is not difficult to see that if we assume that

$$pref_j(f) = (X_0, (\sigma_1, X_1) \dots (\sigma_j, X_j)) \in (e_A||_B pref_j(f^*)),$$

Corollary 13.2

$$P + Q \subseteq (P + Q) \oplus P = (P \oplus Q) + P \subseteq P \oplus Q.$$
$$P + Q \subseteq (P + Q) \oplus Q = (P \oplus Q) + Q \subseteq P \oplus Q.$$

13.3 Parallel Composition

In this section we introduce the operation of *parallel composition.* The operator that we present here extends significantly the scope of synchronization operators previously proposed in the literature. Specifically, our operator can model synchronizations ranging from rigid concurrency (with deadlock) to broadcast synchronization [41] that is deadlock free. It is this operation that requires the extended modeling framework developed in the present paper.

Let P and Q be processes in $\mathcal{P}_\Sigma$ and let A and B be subsets of Σ, that we call the *blocking* or *priority* sets of P and Q, respectively. We define now the *prioritized synchronous composition* of P and Q with priority sets A and B, denoted $P_A||_BQ$, as follows:

$$P_A||_BQ := \{g \mid \exists e \in P,\ f \in Q\ :\ g \in (e_A||_Bf)\}, \tag{132}$$

where $(e_A||_Bf)$ denotes the set of all successful synchronized interleavings of e and f as defined below. We shall also need the following notation: For subsets $X, Y \in \Sigma$ define

$$S(X,Y) = S_{A,B}(X,Y) := \begin{cases} (X \cap Y) \cup (X \cap A) \cup (Y \cap B) & \text{if } \Uparrow \notin X \cup Y \\ \Sigma_{td} & \text{otherwise} \end{cases} \tag{133}$$

The set $S(X,Y)$ is a composite refusal set and represents the idea that an event is refused if it is either refused by both processes or if it is refused by one process that can block it (i.e., the event is in the refusing process' priority set).

The definition of $(e_A||_Bf)$ is given inductively as follows: (It will be convenient here to use the right representation of trajectories.)

Definition 13.1 (i) For $e = (X_0, \varepsilon) \in P$ and $f = (Y_0, \varepsilon) \in Q$,

$$(e_A||_Bf) := \{(Z,\varepsilon) | Z \subseteq S(X_0, Y_0)\}.$$

(ii) Assume that $(e_A||_Bf)$ is defined for trajectories

$$e = (X_0, (\sigma_1, X_1) \dots (\sigma_k, X_k)) \in P, \tag{134}$$
$$f = (Y_0, (\mu_1, Y_1) \dots (\mu_l, Y_l)) \in Q, \tag{135}$$

where

$$\begin{aligned} R(c) = & \quad P(c) & if \quad & c \in A - B \\ = & \quad Q(c) & if \quad & c \in B - A \\ = & \quad P(c) \oplus Q(c) & if \quad & c \in A \cap B. \end{aligned}$$

Also,

$$(a : A \to P(a)) \oplus (a : A \to Q(a)) = (a : A \to P(c) \oplus Q(a)). \tag{131}$$

As a further illustration of how the operators $+$ and $\oplus$ differ from each other, consider the following example.

Example 13.2 Lel $\Sigma = \{a, b\}$ and define the processes $P = (a \to \Delta) = gen((\{b\}, a), \{a, b\})$ and $Q = (b \to \Delta) = gen((\{a\}, b), \{a, b\})$. We can obtain a variety of composite porocesss from P and Q by using the external and internal choice operators as follows:

$$\begin{aligned} P + Q \quad & = gen\{((\emptyset, a), \{a, b\}),\ ((\emptyset, b), \{a, b\})\}. \\ P \oplus Q \quad & = gen\{((\{b\}, a), \{a, b\}),\ ((\{a\}, b), \{a, b\})\}. \\ (P + Q) \oplus P \quad & = gen\{((\emptyset, a), \{a, b\}),\ ((\emptyset, b), \{a, b\}),\ ((\{b\}, a), \{a, b\})\}. \\ (P + Q) \oplus Q \quad & = gen\{((\emptyset, a), \{a, b\}),\ ((\emptyset, b), \{a, b\}),\ ((\{a\}, b), \{a, b\})\}. \end{aligned}$$

The above example is a special case of the following summary of the possible ways choice between two processes can be exercised. We express the different choice operators in a symmetric fashion to emphasize the distinctions.

Proposition 13.8

$$P \oplus Q = \begin{cases} \{e \in (P \cup Q) \mid pref_0(e) \in P \cup Q\} & \textit{if } P \cup Q \neq \nabla \\ \nabla & \textit{otherwise} \end{cases}$$

$$(P + Q) \oplus P = (P \oplus Q) + P = \begin{cases} \{e \in (P \cup Q) \mid pref_0(e) \in P\} & \textit{if } P \cup Q \neq \nabla \\ \nabla & \textit{otherwise} \end{cases}$$

$$(P + Q) \oplus Q = (P \oplus Q) + Q = \begin{cases} \{e \in (P \cup Q) \mid pref_0(e) \in Q\} & \textit{if } P \cup Q \neq \nabla \\ \nabla & \textit{otherwise} \end{cases}$$

$$P + Q = \begin{cases} \{e \in (P \cup Q) \mid pref_0(e) \in P \cap Q\} & \textit{if } P \cup Q \neq \nabla \\ \nabla & \textit{otherwise} \end{cases}$$

An immediate consequence of the above proposition is the following

Corollary 13.1 *Let $P, Q \in \mathcal{P}_{\Sigma_{td}}$ be processes. Then*

$$\begin{array}{rcl}(a \to P) + (a \to Q) & = & (a \to P) \oplus (a \to Q), \\ (a \to P) + (a \to Q) & = & (a \to P \oplus Q).\end{array}$$

We also have the following distributivity laws for $+$ and $\oplus$.

Proposition 13.7 *Let P, Q and R be processes in $\mathcal{P}_{\Sigma_{td}}$. Then*

$$\begin{array}{c}(P + Q) \oplus R = (P \oplus R) + (Q \oplus R), \\ (P \oplus Q) + R = (P + R) \oplus (Q + R).\end{array} \tag{130}$$

Proof. We shall prove the first distributivity law. The second one follows similarly.

$$(P + Q) \oplus R =$$

$$= (P + Q) \cup R$$

$$= \begin{cases} \{e \in P \cup Q \cup R \mid pref_0(e) \in ((P \cap Q) \cup R)\} & \text{if } P \cup Q \neq \nabla \\ \nabla \cup R & \text{otherwise} \end{cases}$$

$$= \begin{cases} \{e \in (P \cup R) \cup (Q \cup R) \mid pref_0(e) \in ((P \cup R) \cap (Q \cup R))\} & \text{if } P \cup Q \neq \nabla \\ \nabla & \text{otherwise} \end{cases}$$

$$= \begin{cases} \{e \in (P \cup R) \cup (Q \cup R) \mid pref_0(e) \in ((P \cup R) \cap (Q \cup R))\} & \text{if } (P \cup R) \cup (Q \cup R) \neq \nabla \\ \nabla & \text{otherwise} \end{cases}$$

$$= (P \oplus R) + (Q \oplus R).$$

The third equality above follows from the fact that if $R = \nabla$, the condition $pref_0(e) \in ((P \cup R) \cap (Q \cup R))\}$ is obviously always satisfied.

∎

An immediate consequence of the above discussion is the following. Let $A, B \in \Sigma$. Then

$$(a : A \to P(a)) + (b : B \to Q(b)) = (c : A \cup B \to R(c))$$

Proposition 13.4 *The external choice operator is well defined and continuous.*

The external choice operator is easily seen to be idempotent, commutative and associative, with the process Δ serving as unit and the process ∇ serving as a zero, that is, for processes P, Q and R we have,

$$\begin{aligned} P + \Delta &= P \\ P + \nabla &= \nabla \qquad (126) \\ P + P &= P \\ P + Q &= Q + P \qquad (127) \\ (P + Q) + R &= P + (Q + R). \end{aligned}$$

By Theorem 12.1, the union R of two processes P and Q is a process. The process R can initially either refuse events that can be refused by P or events that can be refused by Q but the choice cannot be influenced by the environment. We interpret this choice as being determined internally in the process by a completely nondeterministic mechanism. We call this choice the *internal choice operator*, denoted $\oplus$. Thus, for processes P and Q, we define

$$P \oplus Q := P \cup Q. \qquad (128)$$

Obviously, just as the operator $+$, the operator $\oplus$ is also idempotent, associative and commutative, and has ∇ as the zero. However, it does not have Δ as its unity. It is interesting to observe that, in view of Equation (87), $R = P \oplus Q$ implies that $R \overset{\varepsilon}{\to} P$ and $R \overset{\varepsilon}{\to} Q$, namely, both P and Q are ε-postprocesses of R. Indeed, operationally, R is precisely the process that either makes a silent transition to become P or it makes a silent transition to become Q. As a special case of the above, we have

$$P = P \oplus Q \;\Leftrightarrow\; P \overset{\varepsilon}{\to} Q \;\Leftrightarrow\; P \sqsubseteq Q \qquad (129)$$

Proposition 13.5 *The internal choice operator is continuous.*

Consider now the case when we have two prefix processes $P = (a \to \overline{P}))$ and $Q = (a \to \overline{Q})$ and let $R = P \oplus Q$. The first event of the process R is the event a, after which it evolves either to P or to Q but the choice of whether, after the execution of a, the process R behaves like P or like Q is not specified by the process and, thus, cannot be influenced by the environment. The above is a special case of the following proposition which is an immediate consequence of the definitions of the operators $+$ and $\oplus$.

Proposition 13.6 *Let $P, Q \in \mathcal{P}_{\Sigma_{td}}$ be processes such that $(X, \varepsilon) \in P$ if and only if $(X, \varepsilon) \in Q$. Then $P + Q = P \oplus Q$.*

(Notice that the event a is a bound variable in the above expression.) As before, if $Q(a)$ makes silent transitions, so does the process (120), that is,

$$Q(a) \xrightarrow{\varepsilon} Q'(a) \ \Rightarrow\ (a : A \to Q(a)) \ \xrightarrow{\varepsilon}\ (a : A \to Q'(a)). \tag{122}$$

In the process (121), the selection of the first event $a \in A$ is deterministic, that is, completely controlled by the environment. Furthermore, upon selection of the event $a \in A$, the specific process $(a \to Q(a))$ is completely determined. This leads us to an important interpretation of the parameterization of the prefix construction. Consider, for simplicity, the case where $A = \{\sigma_1, \sigma_2\}$. For each σ_i, the process $(\sigma_i \to Q(\sigma_i))$ is given by (115), and the parameterized prefix construction is given by (121). It is then easy to check that the following holds

$$\begin{aligned}&(\sigma_i : \{\sigma_1, \sigma_2\} \to Q(\sigma_i)) = \\ &\{e \in (\sigma_1 \to Q(\sigma_1)) \cup (\sigma_2 \to Q(\sigma_2)) \mid \\ &\mathit{pref}_0(e) \in (\sigma_1 \to Q(\sigma_1)) \cap (\sigma_2 \to Q(\sigma_2))\}.\end{aligned} \tag{123}$$

Thus, the parameterization of the prefix construction can be interpreted as an operator on processes that deterministically selects a process from a specified class of processes through the execution of the first event. We call this operator the *external choice operator* and denote it by the addition symbol $+$. Thus for two processes $(\sigma_1 \to Q(\sigma_1))$ and $(\sigma_2 \to Q(\sigma_2))$, we can write

$$(\sigma_1 \to Q(\sigma_1)) + (\sigma_2 \to Q(\sigma_2)) \ :=\ (\sigma_i : \{\sigma_1, \sigma_2\} \to Q(\sigma_i)). \tag{124}$$

The external choice operator can be extended to processes that do not necessarily have disjoint initial events. For two arbitrary processes P and Q, the external choice operator is defined as,

$$P + Q := \begin{cases} \{e \in P \cup Q \mid \mathit{pref}_0(e) \in (P \cap Q)\} & \text{if} \quad P \cup Q \neq \nabla \\ \nabla & \text{otherwise} \end{cases} \tag{125}$$

Thus, the process $P + Q$ can initially refuse only events that can be refused by both P and Q and afterwards evolves either to P or to Q, unless one (or both) of the processes diverges from the start, in which case so does also the process $P + Q$. The important point is that at the start, the composite process has externally available all events that are externally available to either of the individual components unless one or the other diverges. The choice of initial event can thus be decided by the environment whenever it can be decided by the environment in the component processes.

We interpret the process $(\sigma \to Q)$ as the process that first executes the event σ and then proceeds like Q. Thus, we can define the process P as

$$P = (\sigma \to P/\sigma) \tag{117}$$

and for an execution string $w \in Q$ we can write

$$Q/w = (\sigma \to Q)/\sigma\char`\^w, \tag{118}$$

where $\sigma\char`\^w \equiv (\emptyset, \sigma)\char`\^w$. If the process Q makes silent transitions, then so does also the process $(\sigma \to Q)$ and it follows that

$$Q \xrightarrow{\varepsilon} Q' \;\Rightarrow\; (\sigma \to Q) \;\xrightarrow{\varepsilon}\; (\sigma \to Q'). \tag{119}$$

By Theorem 13.1, the prefix operator is continuous if it is monotonic and pre-image finite. Both of these properties are obvious from the definition, whence we have

Proposition 13.3 *The prefix operator* $(\sigma \to \cdot)$ *is continuous.*

An example of a process built with the prefix construction is

Example 13.1 (The process $(\sigma \to \Delta)$) . This process is (see Example 12.2 and the defining expression (115) above)

$$\begin{aligned}(\sigma \to \Delta) &= \{(\varepsilon, X) \mid X \subseteq \Sigma - \{\sigma\}\} \cup \{((Y, \sigma)\char`\^\varepsilon, X) \mid Y \subseteq \Sigma - \{\sigma\}\,,\ X \subseteq \Sigma\} \\ &= gen((\Sigma - \{\sigma\}, \sigma), \Sigma).\end{aligned}$$

The prefix construction can be parameterized as follows. Let $A \subseteq \Sigma$ be a subset of event symbols and for each $a \in A$, let $Q(a)$ be a process. Then we define the parameterized prefix process

$$(a : A \to Q(a)) \tag{120}$$

as the process that for any event $a \in A$ first executes the event a and then proceeds to execute $Q(a)$. It is given by

$$\begin{aligned}(a : A \to Q(a)) = \quad & \{(\varepsilon, X) \mid X \subseteq \Sigma - A\} \cup \\ & \{((Y, a)\char`\^w, X) \mid a \in A,\ Y \subseteq \Sigma - A,\ (w, X) \in Q(a)\}.\end{aligned} \tag{121}$$

Finally, we remark that a construction similar to the above can be applied to processes defined by *mutually recursive* equations such as

$$P_i = f_i(P_1, \ldots, P_N) \qquad i = 1, \ldots, N. \tag{113}$$

13.2 Prefix Construction and Choice Operators

Let $Q \in \mathcal{P}_{\Sigma_{td}}$ be a process and let $\sigma \in \Sigma$ be an event symbol. We define the process

$$P := (\sigma \to Q) \tag{114}$$

by

$$\begin{aligned} P = (\sigma \to Q) &= \{(\varepsilon, X) \mid X \subseteq \Sigma - \{\sigma\}\} \\ &\cup \{((Y,\sigma)\hat{\ }w, X) \mid Y \subseteq \Sigma - \{\sigma\},\ (w,X) \in Q\}. \end{aligned} \tag{115}$$

To see that P as given by (115) is a well defined process, we must verify that it satisfies all the conditions of Definition 12.1. Let us focus attention on conditions C2 and C6 (the other conditions are straightforward). To see C2, note that if

$$e = ((X_0, \sigma_1) \ldots (X_{k-1}, \sigma_k), X_k) \in P,$$

then by (115), $\sigma_1 = \sigma$ and $X_0 \subseteq \Sigma - \sigma$ whence $\sigma_1 \notin X_0$. For $j > 0$ condition C2 holds by virtue of the fact that it holds for Q. To see that condition C6 holds, it is again enough to consider the case $j = 0$. (We shall use here right representation of trajectories). Let $(X_0, w) \in P$ and let $e = (X_0, (\mu, \emptyset))$, $\mu \notin X_0$. Then $e \notin P$ implies that that $\mu \neq \sigma$ and hence, by (115), $(X_0 \cup \{\mu\}, w) \in P$. Thus the process P is indeed well defined as claimed.

The construction (114) is called the *prefix* construction or the *prefix operator* and we have the following

Proposition 13.2 *The prefix operator is well defined.*

For any $Y \subseteq \Sigma - \{\sigma\}$, it then follows that $Q = P/(Y, \sigma)$. If we choose $Y = \emptyset$, we can, in particular, write $Q = P/(\emptyset, \sigma)$, which, upon identification of the execution $(\emptyset, \sigma)$ with the event symbol σ, becomes

$$Q = P/\sigma. \tag{116}$$

The above proposition provides us with a fairly manageable definition of continuity. A condition which is frequently even easier to verify is provided next. A function f on processes is said to be *pre-image finite* if for every trajectory e, $f^{-1}(e)$ is finite (i.e., consists of a finite number of trajectories). We then have the following

Theorem 13.1 *The function f is continuous if it is monotonic and pre-image finite.*

Proof. Since by monotonicity $f(\sqcup\mathcal{D}) \subseteq \sqcup f(\mathcal{D})$, we must show that the pre-image finiteness property implies that for every directed set $\mathcal{D}$,

$$\sqcup f(\mathcal{D}) \subseteq f(\sqcup\mathcal{D})$$

holds. Consider some $e \notin f(\sqcup\mathcal{D})$. Then $f^{-1}(e) \cap (\sqcup\mathcal{D}) = \emptyset$. Since $f^{-1}(e)$ is finite and $\mathcal{D}$ is directed, there exists a process $P \in \mathcal{D}$ such that $f^{-1}(e) \cap P = \emptyset$, whence $e \notin f(P)$. But then, since $\sqcup f(P) \subseteq f(P)$, it follows that $e \notin \sqcup f(P)$, concluding the proof.

∎

We turn now to the main purpose of the present discussion, namely to the existence of fixed points and their computation. Let $f : \mathcal{P}_{\Sigma_{td}} \to \mathcal{P}_{\Sigma_{td}}$ be a function. A fixed point of f is then a process P such that $f(P) = P$. The following Theorem establishes the existence of fixed points:

Theorem 13.2 (Knaster-Tarski). *Let $f : \mathcal{P}_{\Sigma_{td}} \to \mathcal{P}_{\Sigma_{td}}$ be a monotonic function. Then f has a least fixed-point μf (or $\mu(p).f(p)$) in $\mathcal{P}_{\Sigma_{td}}$. If f is also continuous, then $\mu(p).f(p)$ can be represented as*

$$\mu(p).f(p) = \sqcup\{f^n(\nabla) \mid n \geq 0\}, \tag{111}$$

where ∇ is the divergence process (the least element of $(\mathcal{P}_{\Sigma_{td}}, \sqsubseteq)$).

Theorem 13.2 allows us to define *recursive processes*, that is, processes defined as least fixed-point solutions to recursive equations of the form

$$P = f(P), \tag{112}$$

where f is a continuous function on processes. The process P is then given by μf of (111). If Q is any other solution of (112), then $P \sqsubseteq Q$, that is, P is the most nondeterministic solution of (112). (It follows of course immediately that if P is deterministic it is also uniqe.)

13 Operators on Processes

13.1 Continuity and Fixed Points

In this section we summarize some standard results on continuity and fixed-points of functions on cpo's that are necessary for establishing the soundness of recursive processes. Recall that we are concerned here with the partial order $(\mathcal{P}_{\Sigma_{td}}, \sqsubseteq)$. A function $f : \mathcal{P}_{\Sigma_{td}} \to \mathcal{P}_{\Sigma_{td}}$ is said to be *monotonic* if it respects the partial order relation, that is, f is monotonic if and only if for all $P, Q \in \mathcal{P}_{\Sigma_{td}}$

$$P \sqsubseteq Q \quad \Rightarrow \quad f(P) \sqsubseteq f(Q). \tag{109}$$

A function f is said to be *continuous* if it preserves least upper bounds, that is, if for every directed set of processes $\mathcal{D}$,

$$\sqcup f(\mathcal{D}) = f(\sqcup \mathcal{D}). \tag{110}$$

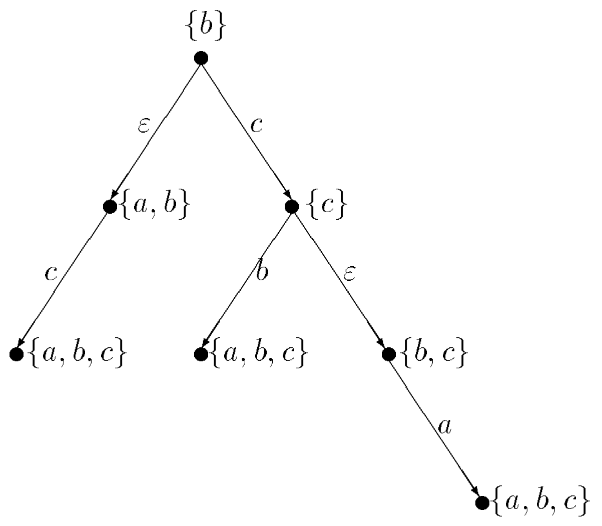
A function $f : \mathcal{P}^n_{\Sigma_{td}} \to \mathcal{P}_{\Sigma_{td}}$ in n arguments is monotonic (respectively, continuous) if it is monotonic (respectively, continuous) in each argument separately when all other arguments are held constant. An immediate consequence of the definitions is that continuity implies monotonicity. In fact, the precise connection between monotonicity and continuity is given by the following

Proposition 13.1 *The function f is continuous if and only if f is monotonic, and for every directed set of processes $\mathcal{D}$, $\sqcup f(\mathcal{D}) \subseteq f(\sqcup \mathcal{D})$.*

Proof. Suppose f is continuous. Then the only claim that is not obvious is that f is monotonic. Let $P \sqsubseteq Q$. Then $\mathcal{D} = \{P, Q\}$ is a directed set and $\sqcup \mathcal{D} = Q$. By continuity, $\sqcup f(\mathcal{D}) = f(\sqcup \mathcal{D}) = f(Q)$, whence $f(P) \sqsubseteq f(Q)$, establishing monotonicity.

Conversely, assume that f is monotonic and that for every directed set $\mathcal{D}$, $\sqcup f(\mathcal{D}) \subseteq f(\sqcup \mathcal{D})$ holds. To prove continuity, it remains to show that monotonicity implies the reverse inclusion, that is, $f(\sqcup \mathcal{D}) \subseteq \sqcup f(\mathcal{D})$. To this end assume that $e \notin \sqcup f(\mathcal{D})$. Then there exists $P \in \mathcal{D}$ such that $e \notin f(P)$. Since $f(\sqcup \mathcal{D}) \subseteq f(P)$ by monotonicity of f, it follows that $e \notin f(\sqcup \mathcal{D})$, concluding the proof.

∎



$gr(P)$

Figure 13:

Algorithm 12.1 always generates a loop-free state-transition *tree* as transition graph of a process, which is generally not finite unless P is finite. We turn now to characterization of *regular* processes (much in the spirit of regular languages) and will show how to construct finite state transition graphs for such processes.

Let P be a process over a finite alphabet Σ and consider again the set $\mathcal{M}(P)$ of $\leq$-maximal trajectories. A trajectory $e = (X_0, (\sigma_1, X_1) \ldots (\sigma_k, X_k)) \in \mathcal{M}(P)$ is called *terminal* if $X_k = \Sigma$. A trajectory $e \in \mathcal{M}(P)$ is said to be *recurrent* if there exists a right execution string $v \in \mathcal{E}^r$ such that

$$e\hat{\ }v^j \in \mathcal{M}(P) \qquad \forall j \geq 0, \tag{106}$$

where $v^j = v\hat{\ } \ldots \hat{\ }v$ (j times). If e is a recurrent trajectory, we call a (right) execution string v a *recurrence* of e if it satisfies (106) above, and there is no proper prefix of v that does so. Further, we let $V(e)$ denote the set of all recurrences of e, we let $V^*(e)$ be the set of all execution strings w of the form

$$w = v_{i_1}^{j_1}\hat{\ } \ldots \hat{\ } v_{i_l}^{j_l}, \tag{107}$$

where $v_{i_1}, \ldots, v_{i_l} \in V(e)$, and $j_1, \ldots, j_l \geq 0$. Finally, if e is recurrent, we define $rec(e)$ as

$$rec(e) := e\hat{\ }V^*(e)* := \bigcup_{w \in V^*(e)} e\hat{\ }w. \tag{108}$$

(v) With each trajectory in $\overline{\mathcal{M}(P)}_{k,e,\sigma}$ associate a distinct node of $gr(P)$.

(vi) For every distinct pair of trajectories f and g in $\overline{\mathcal{M}(P)}_{k,e,\sigma}$ draw an arrow labeled ε (to denote an ε-transition) from f to g provided $f \leq g$. If the $\leq$-minimal set of $\overline{\mathcal{M}(P)}_{k,e,\sigma}$ consists of a single state, draw an arrow labeled σ from e to this state. Otherwise add one more state, draw an arrow labeled σ from e to this new state and an arrow labeled ε from this state to every state in the minimal set.

(vii) Go to step (iii).

(viii) End of algorithm.

Example 12.5 Let the event set be given by $\Sigma = \{a, b, c\}$, and let the process P be given by

$$P = gen \begin{cases} (\{b\}, (c, \{c\})(b, \{a, b, c\})), \\ (\{b\}, (c, \{b, c\})(a, \{a, b, c\})), \\ (\{a, b\}, (c, \{a, b, c\})). \end{cases}$$

The process transition-graph is obtained as state transition graph $gr(P)$. We remark at this point that when P is not a finite set of trajectories, then the present construction will not yield a finite state-transition graph (even when one exists). We shall deal with the finiteness question later in connection with *regular* processes.

Let $\overline{\mathcal{M}(P)} := pref(\mathcal{M}(P))$. We identify the state-set of the process P with the set of all trajectories of $\overline{\mathcal{M}(P)}$ and shall construct its state-transition graph by induction on trajectory length as described below.

Algorithm 12.1 Let $\overline{\mathcal{M}(P)}_0$ be the set of all trajectories of length zero in $\overline{\mathcal{M}(P)}$ (again partially ordered by $\leq$), and with each trajectory of $\overline{\mathcal{M}(P)}_0$ associate a distinct state of P, i.e., a node of $gr(P)$. For every distinct pair of trajectories $e, f \in \overline{\mathcal{M}(P)}_0$ draw an arrow labeled ε (to denote an ε-transition) from e to f provided $e \leq f$. If the *minimal* set of $\overline{\mathcal{M}(P)}_0$, i.e., the set of all $\leq$-minimal elements, consists of a single state, it is called *initial.* (An element $e \in \overline{\mathcal{M}(P)}_0$ is *minimal* if for all $f \in \overline{\mathcal{M}(P)}_0$, $f \leq e \;\Rightarrow\; e = f$.) Otherwise add one more state which we call initial, and draw an arrow labeled ε from the initial state to every state in the minimal set.

Suppose now that the state transition graph has been constructed for all trajectories of length $0, \ldots, k-1$. Let $\overline{\mathcal{M}(P)}_k$ denote the subset of $\overline{\mathcal{M}(P)}$ consisting of all trajectories of length k, and proceed as follows.

(i) Choose a new trajectory $e \in \overline{\mathcal{M}(P)}_{k-1}$. If no new trajectory exists go to step (viii) below.

(ii) Test divergence of e.

  (1) Let X_e denote the final refusal set of e.

  (2) Choose a new symbol $\sigma \in X_e$. If no new symbol exists, go to step (iii).

  (3) Let $\overline{\mathcal{M}(P)}_{k,e,\sigma}$ denote the subset of $\overline{\mathcal{M}(P)}_k$ consisting of all trajectories of the form $e\hat{\ }(\sigma, X_k)$.

  (4) If $\overline{\mathcal{M}(P)}_{k,e,\sigma} = \emptyset$, go to step (2). Otherwise label the state e with a $\Uparrow$ symbol to denote divergence and go to step (i).

(iii) Choose a new symbol $\sigma \in \Sigma - X_e$. If for the given trajectory e no new symbol exists go to step (i).

(iv) If $\overline{\mathcal{M}(P)}_{k,e,\sigma} = \emptyset$ go to step (iii).

Corollary 12.3 $det(S) \overset{\varepsilon}{\to} R \;\Rightarrow\; det(S) = R$

Proof. Since, by (87), $R \subseteq det(S)$, we must show that $det(S) \subseteq R$. By Theorem 12.4, $det(\mathcal{L}(R)) \subseteq R$, whence the proof will be complete if we show that $det(S) \subseteq det(\mathcal{L}(R))$ or, equivalently, that $S \subseteq R_{\mathcal{T}}$.

We proceed by induction on trace length. For ε this is obvious and assume the inclusion holds for all traces of length up to and including n. Let $s = \sigma_1 \ldots \sigma_n \sigma_{n+1} \in S$. Then, by hypothesis, $t = \sigma_1 \ldots \sigma_n \in \mathcal{L}(R)$ and the trajectory $((\emptyset, \sigma_1) \ldots (\emptyset, \sigma_n), \emptyset) \in R$. If $s \notin \mathcal{L}(R)$, so that the trajectory $((\emptyset, \sigma_1) \ldots (\emptyset, \sigma_n)(\emptyset, \sigma_{n+1}), \emptyset) \notin R$, then from condition C6 of Definition 12.1 we conclude that the trajectory $((\emptyset, \sigma_1) \ldots (\emptyset, \sigma_n), \{\sigma_{n+1})$ is in R and hence also in $det(S)$. But this is impossible because it contradicts Equation (93) defining $det(S)$. It follows that $s \in \mathcal{L}(R)$.

∎

The following corollary tells us that deterministic processes always remains deterministic.

Corollary 12.4 $det(S) \overset{w}{\to} R \;\Rightarrow\; R = det(\mathcal{L}(R))$.

Finally, we have the following

Corollary 12.5 $det(S) \overset{w}{\to} R \;\&\; R \overset{\varepsilon}{\to} S \;\Rightarrow\; R = S$.

12.7 State Transition Graphs

In this section we shall show how we can construct for a given process P an associated state-space (or state-set) and a corresponding state-transition graph $gr(P)$.

Let P be a process. We define a relation $\leq$ on P as follows. For trajectories $e, f \in P$ we shall say that $f \leq e$ if $f \in comp(e)$. (Recall that if $e = (X_0, (\sigma_1, X_1) \ldots (\sigma_k, X_k))$ and $f = (Y_0, (\mu_1, Y_1) \ldots (\mu_l, Y_l))$ are trajectories of P, then $f \in comp(e)$ provided $k = l$, $\mu_i = \sigma_i$ for all $i = 1, \ldots, k$, and $Y_i \subseteq X_i$ for all $i = 0, \ldots, k$.) It is readily noted that the relation $\leq$ is a partial order. Let $\mathcal{M}(P)$ be the set of all maximal elements of P with respect to $\leq$. (An element $e \in P$ is *maximal* [30] if for all $f \in P$, $e \leq f \;\Rightarrow\; e = f$.) It is clear that $\mathcal{M}(P)$ is a generating set of P and we shall next see how we can construct from $\mathcal{M}(P)$ a

$$((\emptyset, \sigma_1)(\emptyset, \sigma_2) \ldots (\emptyset, \sigma_k), \emptyset) \in P.$$

Let j, $0 \leq j \leq k$, be the first index such that

$$((X_0, \sigma_1) \ldots (X_j, \sigma_{j+1})(\emptyset, \sigma_{j+2}) \ldots (\emptyset, \sigma_k), \emptyset) \notin P. \tag{100}$$

Then there exists a proper subset $Y \subseteq X_j$ such that

$$((X_0, \sigma_1) \ldots (X_{j-1}, \sigma_j)(Y, \sigma_{j+1})(\emptyset, \sigma_{j+2}) \ldots (\emptyset, \sigma_k), \emptyset) \in P \tag{101}$$

and an event symbol $x \in X_j - Y$ such that

$$((X_0, \sigma_1) \ldots (X_{j-1}, \sigma_j)(Y \cup \{x\}, \sigma_{j+1})(\emptyset, \sigma_{j+2}) \ldots (\emptyset, \sigma_k), \emptyset) \notin P. \tag{102}$$

From condition C6 of Definition 12.1 it follows that

$$((X_0, \sigma_1) \ldots (X_{j-1}, \sigma_j)(Y, x), \emptyset) \in P. \tag{103}$$

This implies that the trace $\sigma_1 \ldots \sigma_j x \in S$ and we must conclude from (93) that

$$((X_0, \sigma_1) \ldots (X_{j-1}, \sigma_j), Y \cup \{x\}) \notin det(S) \tag{104}$$

contradicting our assumption. This concludes the proof.

∎

An important consequence of the above theorem is that the set of all processes whose trace set is S is a directed set. Thus we have the following interesting corollaries to Theorem 12.2:

Corollary 12.1 *Let S be a closed subset of Σ^*. The set of processes $\mathcal{C}(S)$ such that*

$$P \in \mathcal{C}(S) \ \Leftrightarrow\ \mathcal{L}(P) = S \tag{105}$$

is directed.

Corollary 12.2 *Let S be a prefix-closed set of traces and let $\mathcal{D}$ be a set of processes such that $P \in \mathcal{D}$ if and only if $\mathcal{L}(P) = S$. Then $\cap\mathcal{D}$ is a process.*

The following corollary tells us that if a process is deterministic it cannot undergo silent or unobserved changes.

it follows from (93) and (94) that

$$((X_0, \sigma_1)(X_1, \sigma_2) \ldots (X_{j-1}, \sigma_j)(X_j \cup \{\sigma\}, \sigma_{j+1}), \emptyset) \in det(S). \tag{97}$$

It is now not hard to show now with repeated use of (93), (94) and (95) that (97) implies that

$$((X_0, \sigma_1) \ldots (X_{j-1}, \sigma_j) \ldots (X_j \cup \{\sigma\}, \sigma_{j+1}) \ldots (X_{k-1}, \sigma_k), X_k) \in det(S) \tag{98}$$

and condition C6 is establishd. The other conditions of Definition 12.1 are also easily verified.

Next examine conditions (93) and (94) and observe that exactly one of their right-hand-side conditions must always hold. This implies that exactly one of the trajectories in (93) and (94) is in $det(S)$ and that condition (91) is satisfied. Hence, $det(S)$ is deterministic. In fact, have shown the following

Proposition 12.5 *Let S be a prefix-closed set of traces. Then the set of trajectories $det(S)$ is a deterministic process such that $\mathcal{L}(det(S)) = S$.*

An interesting deterministic process is given in the following

Example 12.4 ***(The 'All' Process A).*** This is the deterministic process that at each instant can execute any event in Σ

$$\mathcal{A} = det(\Sigma^*). \tag{99}$$

It is not difficult to verify that $\mathcal{A} = \mathcal{A}_{\mathcal{T}}$, that is, $\mathcal{A}$ consists only of free trajectories.

We also have the following important

Theorem 12.4 *Let S be a prefix-closed set of traces and let P be any process such that $\mathcal{L}(P) = S$. Then $det(S) \subseteq P$.*

Proof. We must show that if

$$e = ((X_0, \sigma_1)(X_1, \sigma_2) \ldots (X_{k-1}, \sigma_k), X_k) \in det(S),$$

then $e \in P$. Now we know that $tr(e) \in S$, whence the trajectory and it is not hard to see that we can identify the set $P_{\mathcal{T}}$ with $\mathcal{L}(P)$, the language generated by P.

We turn now to the inverse question. Let S be a prefix closed subset of Σ^*. Do there exist processes P such that $\mathcal{L}(P) = S$? More specifically, how can such processes be constructed?

Definition 12.4 A process P is called *deterministic* if for every trajectory $(w, X) \in P$ and any $\sigma \in \Sigma$

$$(w\hat{}(X,\sigma), \emptyset) \notin P \;\Leftrightarrow\; (w, X \cup \{\sigma\}) \in P. \tag{91}$$

Thus, a process is deterministic whenever events are refused if and only if they are impossible. It is worthwhile to compare (91) with condition C6 of Definition 12.1 where it was only required that impossible events are refused.

Now let S be a prefix-closed set of traces and define $det(S)$ to be the set of trajectories obtained by the following inductive procedure:

$$(\varepsilon, \emptyset) \in det(S) \tag{92}$$

and if $e = (w, X) \in det(S)$ and $\sigma \in \Sigma$, then

$$(w, X \cup \{\sigma\}) \in det(S) \;\Leftrightarrow\; tr(e)\hat{}\sigma \notin S \tag{93}$$

$$(w\hat{}(X,\sigma), \emptyset) \in det(S) \;\Leftrightarrow\; tr(e)\hat{}\sigma \in S. \tag{94}$$

It is readily verified that $det(S)$ is a process. To this end it is necessary to show that the conditions of Definition 12.1 are satisfied. Again, let us examine condition C6. Consider any trajectory

$$((X_0, \sigma_1)(X_1, \sigma_2)\ldots(X_{k-1}, \sigma_k), X_k) \in det(S) \tag{95}$$

and let j, $0 \le j \le k$, and $\sigma \in \Sigma - X_j$ be such that

$$((X_0, \sigma_1)(X_1, \sigma_2)\ldots(X_{j-1}, \sigma_j)(X_j, \sigma), \emptyset) \notin det(S). \tag{96}$$

Since by our construction

$$((X_0, \sigma_1)(X_1, \sigma_2)\ldots(X_{j-1}, \sigma_j), X_j) \in det(S)$$

Note that $\sqcup\mathcal{D}$ need not be an element of $\mathcal{D}$.

An infinite sequence of processes $\{P_i \mid i \geq 0\}$ is called a *chain* if it satisfies the condition that

$$\forall i \,.\, P_i \sqsubseteq P_{i+1}.$$

Clearly a chain of processes is a directed set and the least upper bound of the chain $\sqcup P_i$ is then called the *limit* of the chain.

Theorems 12.1 and 12.2 imply the following important

Proposition 12.4 *The partial order $(\mathcal{P}_{\Sigma_{td}}, \sqsubseteq)$ is a complete partial order (cpo).*

The following fact is also important.

Theorem 12.3 *Let $\mathcal{D}$ be a directed set of processes. Then*

$$\sqcup\mathcal{D} \overset{w}{\rightarrow} R \;\Leftrightarrow\; \forall Q \in \mathcal{D} \,.\, Q \overset{w}{\rightarrow} R$$

Proof. By (85)

$$\begin{aligned} \sqcup\mathcal{D} \overset{w}{\rightarrow} R \;&\Leftrightarrow\; R \subseteq \{(v,X) | (w^\frown v, X) \in \sqcup\mathcal{D}\} \\ &= \{(v,X) | (w^\frown v, X) \in \cap Q\} \\ &\Leftrightarrow\; \forall Q \in \mathcal{D} \,.\, R \subseteq \{(v,X) | (w^\frown v, X) \in Q\} \\ &\Leftrightarrow\; \forall Q \in \mathcal{D} \,.\, Q \overset{w}{\rightarrow} R. \end{aligned}$$

∎

Consider now a process P and let $P_{\mathcal{T}}$ denote the subset of P consisting of all *free* trajectories of P, that is, all trajectories $(w, \emptyset) \in P$ for which w has the form

$$w = (\emptyset, \sigma_1)(\emptyset, \sigma_2) \ldots (\emptyset, \sigma_k). \tag{90}$$

Thus, each free trajectory of P can be identified with its associated trace s,

$$s = \sigma_1 \ldots \sigma_k \; (\in \Sigma^*),$$

Definition 12.3 . A set of processes $\mathcal{D}$ is called *directed* if for each pair $Q_1, Q_2 \in \mathcal{D}$ there exists a process $R \in \mathcal{D}$ such that $Q_1 \sqsubseteq R \;\&\; Q_2 \sqsubseteq R$.

Theorem 12.2 *The intersection of any directed set of processes is a process.*

Proof. Let $\mathcal{D}$ be a directed set of processes and let $P := \cap\mathcal{D}$. Then

$$(w, X) \in P \;\Leftrightarrow\; \forall\, Q \in \mathcal{D} \,.\, (w, X) \in Q$$

To show that P is a process it is necessary to prove that it satisfies conditions C1-C7 of Definition 12.1. Conditions C1-C5 as well as condition C7 hold for the intersection of any set of processes. We shall prove condition C6 which requires also the directedness property.

Suppose

$$e = ((X_0, \sigma_1)(X_1, \sigma_2) \ldots (X_{k-1}, \sigma_k), X_k) \in P,$$

and for some j, $0 \leq j \leq k$, and some $\sigma \in \Sigma - X_j$,

$$e' = ((X_0, \sigma_1)(X_1, \sigma_2) \ldots (X_{j-1}, \sigma_j)(X_j, \sigma), \emptyset) \notin P.$$

Then $e \in Q$ for all $Q \in \mathcal{D}$ but for some process $Q_1 \in \mathcal{D}$, $e' \notin Q_1$. Now condition C6 will hold for P unless there exists some process $Q_2 \in \mathcal{D}$ such that

$$\overline{e} = ((X_0, \sigma_1) \ldots (X_{j-1}, \sigma_j)(X_j \cup \{\sigma\}, \sigma_{j+1}) \ldots (X_{k-1}, \sigma_k), X_k) \notin Q_2.$$

However, by the directedness of $\mathcal{D}$, there is a process $R \in \mathcal{D}$ such that $Q_1 \overset{\varepsilon}{\rightarrow} R$ and $Q_2 \overset{\varepsilon}{\rightarrow} R$ so that $R \subseteq Q_1 \cap Q_2$. It then follows that $e \in R$, $e' \notin R$ and $\overline{e} \notin R$, contradicting condition C6 for R. This violates the assumption that Q_2 exists and the proof is complete.

∎

Let $\mathcal{D}$ be a directed set of processes.Then for each process $P \in \mathcal{D}$, $P \sqsubseteq \cap\mathcal{D}$. Thus, the process $\cap\mathcal{D}$ is the least upper bound of $\mathcal{D}$ and is denoted $\sqcup\mathcal{D}$. By definition of $\sqcup\mathcal{D}$ we then have that

$$\forall Q \in \mathcal{D} \,.\, Q \sqsubseteq \mathcal{D}. \tag{89}$$

such that $\sigma_j \in \Theta$, it follows that

$$e = ((X_0, \sigma_1) \dots (X_{j-1}, \sigma_j)(X_{j-1}, \sigma_j)^k \dots (X_{k-1}, \sigma_k), X_k) \in P$$

for all $k \geq 0$.

The following Proposition is an easy consequence of the above definition:

Proposition 12.3 *Let P be a Θ-stuttering process and let w be a left execution string of P. If for $\sigma in \Theta$, $w \;\; widehat \; (X, \sigma)$ is also a left execution string of P, then for all $k \geq 0$*

$$P/w\hat{}(X, \sigma)^k = P/w.$$

12.6 Nondeterminism

Intuitively, process nondeterminism occurs when a process undergoes 'silent' or unobservable transitions and its behavior cannot be completely determined from its trace history. Thus, if P is a process, we are interested in the possibility of the process undergoing changes along the empty execution string.

Let Q be an ε-postprocess of a given process P. Then

$$Q \subseteq P/\varepsilon = \{(w, X) | (w, X) \in P\} = P, \tag{86}$$

and it follows that Q is an ε-postprocess of P if and only if $Q \subseteq P$, that is, if and only if Q is a *subprocess* of P. We can then write

$$P \xrightarrow{\varepsilon} Q \;\Leftrightarrow\; P \supseteq Q. \tag{87}$$

It then follows immediately that $\xrightarrow{\varepsilon}$ induces a partial order on processes, that is,

$$\begin{gathered} P \xrightarrow{\varepsilon} P \\ P \xrightarrow{\varepsilon} Q \;\&\; Q \xrightarrow{\varepsilon} P \;\Leftrightarrow P = Q \\ P \xrightarrow{\varepsilon} Q \;\&\; Q \xrightarrow{\varepsilon} R \;\Rightarrow\; P \xrightarrow{\varepsilon} R. \end{gathered}$$

We shall denote the partial order $\xrightarrow{\varepsilon}$ by the more conventional notation $\sqsubseteq$. Thus,

$$P \sqsubseteq Q \;\Leftrightarrow\; P \xrightarrow{\varepsilon} Q \;\Leftrightarrow\; P \supseteq Q, \tag{88}$$

and we say that P is *more nondeterministic than* Q.

Proposition 12.1 $P \overset{u}{\to} Q \;\&\; Q \overset{v}{\to} R \;\Rightarrow\; P \overset{u\widehat{\ }v}{\to} R$

Proof. By (82)

$$P \overset{u}{\to} Q \;\Leftrightarrow\; Q \subseteq \{(v, X) | (u\hat{\ }v, X) \in P\},$$
$$Q \overset{v}{\to} R \;\Leftrightarrow\; R \subseteq \{(w, Y) | (v\hat{\ }w, Y) \in P\}.$$

Hence

$$(w, Y) \in R \;\Rightarrow\; (v\hat{\ }w, Y) \in Q \;\Rightarrow\; (u\hat{\ }v\hat{\ }w, Y) \in P.$$

Upon applying (82) again we obtain $P \overset{u\widehat{\ }v}{\to} R$ as claimed.

∎

Proposition 12.2 $P \overset{u\widehat{\ }v}{\to} R \;\Rightarrow\; \exists Q \,.\, P \overset{u}{\to} Q \;\&\; Q \overset{v}{\to} R.$

Proof. By (82) we have

$$P \overset{u\widehat{\ }v}{\to} R \;\Leftrightarrow\; R \subseteq \{(w, X) | (u\hat{\ }v\hat{\ }w, X) \in P\}.$$

Defining $Q := \{(y, X) | (u\hat{\ }y, X) \in P\}$, it follows directly from (83) that $Q = P/u$ so that $P \overset{u}{\to} Q$. To conclude the proof, we need to show that $Q \overset{v}{\to} R$. This is equivalent to showing that

$$R \subseteq \{(w, X) | (v\hat{\ }w, X) \in Q\}.$$

Indeed, let $(w, X) \in R$. Then, by hypothesis, we have $(u\hat{\ }v\hat{\ }w, X) \in P$ or, alternatively, we have $(u\hat{\ }(v\hat{\ }w), X) \in P$. But $Q = P/u$ so that $(v\hat{\ }w), X) \in Q$ and the proof is complete.

∎

Definition 12.2 A process P is called Θ-*stuttering* for a subset $\Theta \in \Sigma$ if for any trajectory

$$e = ((X_0, \sigma_1) \ldots (X_{j-1}, \sigma_j) \ldots (X_{k-1}, \sigma_k), X_k) \in P$$

Example 12.2 (The Deadlock Process Δ.) This is the process that has no nonempty trajactories by virtue of its initial deadlock. It is given by

$$\Delta = gen((\varepsilon, \Sigma)). \tag{80}$$

Example 12.3 (The Divergence Process ∇.) This is the chaotic process that diverges from the start. It is given by

$$\nabla = \mathcal{O}_{td}. \tag{81}$$

Thus, the process ∇ includes every process in $\mathcal{P}_{td}$.

12.5 Postprocesses and Transitions

Let P be a process and let w be a left execution string of P, that is, $e = (w, \emptyset) \in P$. A process Q is called a *w-postprocess* of P if

$$Q \subseteq \{(v, X) | (w\hat{\ }v, X) \in P\}. \tag{82}$$

It is easily verified by checking the conditions of Definition 12.1 that the right hand side of (82) is itself a process.The process Q for which (82) holds with equality is called the *supremal w-postprocess* of P. We denote this process by P/w, that is,

$$P/w = \{(v, X) | (w\hat{\ }v, X) \in P\}. \tag{83}$$

For two (left) execution strings v and w, we then have

$$(P/v)/w = P/v\hat{\ }w. \tag{84}$$

The above discussion allows us to interpret condition C7 of Definition 12.1 as a postprocess condition. Specifically, the condition states that if $(w, X) \in P$ and $\Uparrow \in X$, then $P/w = \nabla$, that is, the postprocess after occurrence of divergence is the divergence process.

A *process transition* for an execution string w, denoted $\xrightarrow{w}$, is a relation on $\mathcal{P}_{\Sigma_{td}}$, the set of Σ-processes, defined as

$$P \xrightarrow{w} Q \Leftrightarrow Q \subseteq P/w \tag{85}$$

We say that process P undergoes transition to process Q along the execution string w if Q is a w-postprocess of P.

The following properties of postprocesses and transitions follow easily:

Theorem 12.1 *The union of a nonempty set of processes in is a process.*

Proof. Let $\mathcal{D}$ be a nonempty set of processes and let $P = \cup\mathcal{D}$. Then

$$(w, X) \in P \Leftrightarrow \exists Q \in \mathcal{D} \text{ s.t. } (w, X) \in Q.$$

It must be shown that P satisfies conditions C1 to C7 of Definition 12.1. We shall prove condition C6. Thus, let

$$e := ((X_0, \sigma_1)(X_1, \sigma_2) \ldots (X_{k-1}, \sigma_k), X_k) \in P$$

and assume that for some j, $0 \leq j \leq k$, and some $\sigma \in \Sigma - X_j$,

$$e' := ((X_0, \sigma_1)(X_1, \sigma_2) \ldots (X_{j-1}, \sigma_j)(X_j, \sigma), \emptyset) \notin P.$$

By definition of P it follows that $\exists Q \in \mathcal{D}$ such that $e \in Q$ and $e' \notin Q$. Since Q is a process, it must satisfy condition C6 so that

$$\overline{e} := ((X_0, \sigma_1) \ldots (X_{j-1}, \sigma_j)(X_j \cup \{\sigma\}, \sigma_{j+1}) \ldots (X_{k-1}, \sigma_k), X_k) \in Q.$$

Thus it follows that $\overline{e} \in P$ and condition C6 follows. The remaining conditions are quite straightforward.

∎

Let P be a process and let $\mathcal{G}$ be a set of trajectories of P. We shall call $\mathcal{G}$ a *generating set* or *generator* of P and write $P = gen(\mathcal{G})$ if

$$P = \bigcup_{e \in \mathcal{G}} cl(e). \tag{78}$$

Below are a number of interesting and useful process examples.

Example 12.1 (The Null Process $\mathcal{N}$.) This is the process that has no nonempty trajectories by virtue of the fact that it is initially successfully terminating. It is given by

$$\mathcal{N} = gen((\varepsilon, \{\Downarrow\})). \tag{79}$$

Definition 12.1 A *(discrete event) process* P is a subset $P \subseteq \mathcal{O}_{td} := (2^{\Sigma_{td}} \times \Sigma)^* \times 2^{\Sigma_{td}}$ $(= 2^{\Sigma_{td}} \times (\Sigma \times 2^{\Sigma_{td}})^*)$ satisfying the following conditions:

(C1) $(\varepsilon, \emptyset) \in P$.

(C2) $((X_0, \sigma_1)(X_1, \sigma_2) \ldots (X_{k-1}, \sigma_k), X_k) \in P \;\&\; \exists j :\; 0 \leq j \leq k-1 \;; \sigma_{j+1} \in X_j \;\Rightarrow$
$((X_0, \sigma_1) \ldots (X_{j-1}, \sigma_j), X_j \cup \{\Uparrow\}) \in P$.

(C3) $e = (w, X) \in P \;\Rightarrow\; cl(e) \subseteq P$.

(C4) $(w, X) \in P \;\&\; \{\Downarrow\} \neq X \;\Rightarrow \Downarrow \notin X$.

(C5) $((X_0, \sigma_1)(X_1, \sigma_2) \ldots (X_{k-1}, \sigma_k), X_k) \in P \;\&\; \exists j :\; (0 \leq j \leq k-1) \;; \Downarrow \in X_j \;\Rightarrow$
$((X_0, \sigma_1) \ldots (X_{j-1}, \sigma_j), X_j \cup \{\Uparrow\}) \in P$.

(C6) $((X_0, \sigma_1)(X_1, \sigma_2) \ldots (X_{k-1}, \sigma_k), X_k) \in P \;\&\; \sigma \in \Sigma - X_j,\; 0 \leq j \leq k,\; \&$
$((X_0, \sigma_1)(X_1, \sigma_2) \ldots (X_{j-1}, \sigma_j)(X_j, \sigma), \emptyset) \notin P \;\Rightarrow$
$((X_0, \sigma_1) \ldots (X_{j-1}, \sigma_j)(X_j \cup \{\sigma\}, \sigma_{j+1}) \ldots (X_{k-1}, \sigma_k), X_k) \in P$.

(C7) $e = (w, X) \in P \;\&\; \Uparrow \in X \;\Rightarrow\; (w\hat{\ }v, Y) \in P.\; \forall (v, Y) \in \mathcal{O}_{td}$.

Condition C1 states that the null trajectory is in every process. Condition C2 states that all trajectories of a non-divergent process must be valid. Condition C3 states that a process is a closed family of trajectories. Condition C4 states that the termination symbol is a stand-alone symbol. Condition C5 states that a proper prefix of a trajectory in a non-divergent process is always nonterminating, implying that once the termination symbol appears, no further events can be executed (unless the process diverges and becomes chaotic). Condition C6 states that if an event is impossible it will be refused. (It is worth remarking here that in nondeterministic processes events need not be impossible to be refused.) Finally, condition C7 states that once a process diverges it becomes chaotic forever, or in other words, if $e \in P$ then $div(e) \subseteq P$.

We shall denote the class of all processes (in $\mathcal{O}_{td}$) by $\mathcal{P}_{\Sigma_{td}}$. Similarly, we shall denote the class of all divergence-free processes (i.e., processes in $\mathcal{O}_t$) by $\mathcal{P}_{\Sigma_t}$, the class of nonterminating processes (in $\mathcal{O}_d$) by $\mathcal{P}_{\Sigma_d}$, and the class of divergence-free and nonterminating processes (in $\mathcal{O}_\Sigma$) by $\mathcal{P}_\Sigma$.

The set of all traces $s \in \Sigma^*$ such that $s = tr(e)$ for some $e \in P$, where $P \in \mathcal{P}_{\Sigma_{td}}$ is a process, is called the *language* generated by P and is denoted $\mathcal{L}(P)$. By condition C3 of the above definition it follows immediately that $\mathcal{L}(P)$ is (prefix) closed.

successfully terminating execution string, we define the associated trajectory to be $(w, \{\Downarrow\})$, where $\Downarrow$ indicates the successful termination. Thus, the set $\{\Downarrow\}$ assumes the role of the refusal set in the trajectory, and we apply to it the convention that it is a stand-alone symbol, that is, either $\{\Downarrow\} = X$ or $\Downarrow \notin X$. We denote the alphabet extended by the termination symbol $\Sigma \cup \{\Downarrow\}$ by Σ_t, and the set of observations that include the possibility of successful termination by $\mathcal{O}_t := (2^{\Sigma_t} \times \Sigma)^* \times 2^{\Sigma_t}$ $(= 2^{\Sigma_t} \times (\Sigma \times 2^{\Sigma_t})^*)$. We shall also use the notation $\mathcal{E}^l_t$ and $\mathcal{E}^r_t$ for $(2^{\Sigma_t} \times \Sigma)^*$ and $(\Sigma \times 2^{\Sigma_t})^*)$, respectively.

12.3 Divergence

One further form of process behavior that we want to model is a *catastrophic* form of termination that is called *divergence*. By divergence we intuitively mean that the process has become completely chaotic and unpredictable in its behavior. This can occur as a consequence of inadequate observability, for example when the process can undergo an unbounded sequence of unobservable state transitions. Alternatively, it might conceivably occur as a result of process failure. Since divergence cannot be finitely observed, i.e., there does not exist a finite experiment that can determine divergence, a special *divergence* symbol $\Uparrow$ is introduced into the process alphabet to indicate that divergence has occurred. We denote the alphabet extended by the divergence symbol $\Sigma \cup \{\Uparrow\}$ by Σ_d, and the set of observations that include the possibility of divergence by $\mathcal{O}_d := (2^{\Sigma_d} \times \Sigma)^* \times 2^{\Sigma_d}$ $(= 2^{\Sigma_d} \times (\Sigma \times 2^{\Sigma_d})^*)$. We shall also use the notation $\mathcal{E}^l_d$ and $\mathcal{E}^r_d$ for $(2^{\Sigma_d} \times \Sigma)^*$ and $(\Sigma \times 2^{\Sigma_d})^*)$, respectively. A trajectory

$$(X_0, (\sigma_1, X_1) \ldots (\sigma_k, X_k))$$

will then be called *chaotic* (*non-chaotic*) if there exists (respectively, does not exist) $j : 0 \leq j \leq k$ such that $\Uparrow \in X_j$. A chaotic trajectory is called *divergent* if it has no proper prefix which is also chaotic. If e is a divergent trajectory, we define the *divergence* of e by

$$div(e) := \{e\hat{\ }v \mid v \in calE^r_d\}.$$

12.4 Specification Models

We shall now turn to formalize the concept of a process. Intuitively we identify a process with its behavioral specification, that is, with the set of all its possible trajectories. This leads us to the definition of an abstract object called a *process* as follows. Let Σ denote the process alphabet and let Σ_{td} denote its *extended alphabet*, that is, $\Sigma_{td} = \Sigma \cup \{\Downarrow, \Uparrow\}$.

(so that e is a prefix of f) we say that f is an *extension* of e and that v is a *post execution-string* of e. Sometimes, with some abuse of notation, we shall also write the above simply as $f = e\hat{\ }v$.

Similarly we may consider a trajectory in left representation

$$f = (v\hat{\ }w, X_k) = ((X_0, \sigma_1) \dots (X_{k-1}, \sigma_k), X_k), \tag{75}$$

where

$$v = (X_0, \sigma_1) \dots (X_{l-1}, \sigma_l) \tag{76}$$

and

$$w = (X_l, \sigma_{l+1}) \dots (X_{k-1}, \sigma_k), \tag{77}$$

and consider the trajectory $e = (w, X_k)$. We shall say that the trajectory e is a *suffix* of f, that v is a *pre-execution string* of e and that e is a *post-trajectory* of v. We shall also sometimes use the notation $f = v\hat{\ }e$.

Finally, it is also clear from the earlier discussion that if $s \in \Sigma^*$ is a trace of some trajectory e of a process, then every prefix t of s is also a trace of a trajectory of the process. (A string $t \in \Sigma^*$ is a prefix of s if there exists $r \in \Sigma^*$ such that $t\hat{\ }r = s$, where as before, $\cdot\hat{\ }\cdot$ denotes concatenation. The empty string ε is a prefix of every string.)

12.2 Termination

We turn now to a discussion of various aspects of *process termination.* The simplest way in which a process can terminate is if it can undergo no further state changes. Specifically, suppose that a process possesses a trajectory (w, X) with $X = \Sigma$. That is, after executing the string w, the process refuses every event in its alphabet. The process then terminates by necessity, since no further event executions are physically possible. If this is the case we say that the process has reached *deadlock.*

However, processes can terminate in another way as well. Suppose we assign our process tasks to be completed. We can then say that the process *terminates successfully* whenever it completes a task. Successful termination is, of course, a *legislated* property (rather than a physical one) and is, in general, not accompanied by deadlock. We identify successful termination with the execution of a prescribed set of execution strings. Specifically, we introduce a special *termination symbol* $\Downarrow$ into our process alphabet as follows. If w is a

Then it must clearly include also every *prefix* of e, that is, every trajectory of the form

$$pref_j(e) := (X_0, (\sigma_1, X_1) \dots (\sigma_j, X_j)), \tag{67}$$

with $j = 0, \dots, k$, (where $pref_0(e) := (X_0, \varepsilon)$ and $pref_k(e) = e$). We shall denote the set of all prefixes of e by $pref(e)$, that is

$$pref(e) = \bigcup_{j=0,\dots,k} pref_j(e). \tag{68}$$

We shall sometimes use the notation $f \preceq e$ to denote the fact that $f \in pref(e)$, and $f \prec e$ to denote the fact that $e \neq f \in pref(e)$. If E is a set of trajectories, we shall denote by $pref(E)$ the set of all prefixes of trajectories in E, that is,

$$pref(E) = \bigcup_{e \in E} pref(e). \tag{69}$$

Next, it is clear that the order in which we press the various buttons is arbitrary. Thus, if instead of (σ_j, X_j), the j-th (right) execution had been (σ_j, Y_j) where $Y_j \subseteq X_j$ is any subset, it would have been just as successful. Consequently, we may add to our set of observed trajectories also the set of all the trajectories of the form

$$(Y_0, (\sigma_1, Y_1) \dots (\sigma_k, Y_k)), \tag{70}$$

where $Y_j \subseteq X_j$ for $j = 0, \dots, k$. We call this set of trajectories the *completion* of e, and denote it $comp(e)$. It is clear, in view of our discussion, that the set of trajectories of our process must also include the set of all prefixes for each trajectory in $comp(e)$. The union of all trajectories thus obtained is called the *closure* of e and is denoted $cl(e)$, that is,

$$cl(e) := \bigcup_{v \in comp(e)} pref(v). \tag{71}$$

It thus follows (without necessarily having to make this observation experimentally) that if e is a trajectory of a process so is every trajectory in $cl(e)$.

We proceed now with some extension of our terminology. Let

$$e = (X_0, w) = (X_0, (\sigma_1, X_1) \dots (\sigma_k, X_k)), \tag{72}$$

be a trajectory in right representation. If

$$v = (\sigma_{k+1}, X_{k+1}) \dots (\sigma_l, X_l) \tag{73}$$

$(l > k)$ is a (right) execution string such that $f := (X_0, w\hat{\ }v)$ is also a trajectory, where $w\hat{\ }v$ is the *concatenation* of w and v given by

$$(\sigma_1, X_1) \dots (\sigma_k, X_k)(\sigma_{k+1}, X_{k+1}) \dots (\sigma_l, X_l), \tag{74}$$

successful event, and for $i > 0$, $X_i (\subseteq \Sigma)$ is the set of refused event symbols after the i-th success. The refusal set X_k of e is called its *final refusal*. Thus, a trajectory is an element of the set of *observations* $\mathcal{O} := (2^\Sigma \times \Sigma)^* \times 2^\Sigma$. It will be assumed that under normal behavior (see further discussion below regarding the possibility of divergence) all process trajectories are *valid*, that is, they satisfy the condition that $\sigma_i \notin X_{i-1}$ for all $i > 0$. This assumption is consistent with our interpretation of refusals as persistent, that is, events cannot be executed if they have just been refused. It is convenient to denote each pair (X_{i-1}, σ_i) by w_i and the string $w_1 w_2 \dots w_k$ by w. The number of elements in the string w is called the *length* of w and is denoted $|w|$. We call w_i the i-th *left execution* of e, with *refusal* X_{i-1} and *event* σ_i. The string w $(\in \mathcal{E}^l := (2^\Sigma \times \Sigma)^*)$ is called the *left execution string* of e. We call the trajectory representation of Equation (61), which can also be expressed as

$$e := (w, X), \tag{62}$$

the *left representation* of e. In the above equation it is clearly understood that $X = X_k$. If we terminate our experiment before we ever reach a successful event, the trajectory will be given by the pair (ε, X_0) where ε denotes the empty execution string. The trajectory $(\varepsilon, \emptyset)$ is called the *null trajectory*. The *length* of a trajectory $e = (w, X_k)$ is defined as $|e| = |w| = k$.

The trajectory of Equation (61) can also be written in the form

$$e = (X_0, (\sigma_1, X_1) \dots (\sigma_k, X_k)), \tag{63}$$

which we call the *right representation*. Thus, we think of e as an element of $\mathcal{O}$ represented as $\mathcal{O} := 2^\Sigma \times (\Sigma \times 2^\Sigma)^*$. Here we refer to each pair $v_i = (\sigma_i, X_i)$ as the i-th *right execution* and to the string $v = v_1 \dots v_k$ $(\in \mathcal{E}^r := (\Sigma \times 2^\Sigma)^*)$ as the *right execution string* of the trajectory. The trajectory can, thus, also be written as

$$e = (X, v) \tag{64}$$

where it is clearly understood that $X = X_0$. Obviously, $|v| = |e| = k$. In the sequel we shall use both representations and it will always be clearly understood whether we refer to left or right trajectory representations. Finally, we associate with a trajectory e its *trace* $tr(e)$, that is, its associated string of events

$$tr(e) = s = \sigma_1 \dots \sigma_k. \tag{65}$$

Suppose now that, for a given process, our set of observed trajectories includes a trajectory

$$e = (X_0, (\sigma_1, X_1) \dots (\sigma_k, X_k)). \tag{66}$$

12 Models and Specifications

12.1 Experimentation and Trajectories

In order to get an intuition about the model that we shall later formalize, we introduce it here intuitively through the idea of experimentation on processes.

We think of a discrete event process as a device that can undergo state changes at discrete points in time in response to certain isolated events. In general these events occur asynchronously (i.e. without reference to a clock) and sometimes also nondeterministically. Some of the events (and associated transitions) are observable (or accessible) from the external environment but some may be internal and unobservable. With each observable event is associated an event symbol σ. The set of all observable events, called the *process alphbet*, is denoted Σ.

Let us assume for the sake of the current discussion, that all events in Σ are available for 'external experimentation'; that is, we imagine that there is a pannel of buttons, each marked with a symbol $\sigma \in \Sigma$. When a button is pressed, the corresponding event symbol can either be *accepted* by the process, resulting in a state transition, or it can be *refused* and nothing happens. We regard a refusal as persistent in that repeated experimentation with the same button will not change the outcome. The unobservable events are assumed to occur spontaneously (and at unknown times) and the associated transitions are assumed to be undetectable.

Now, at any time we may press any button. If the corresponding event is possible at that time and a state transition occurs, we record this fact. Otherwise, if the process refuses the event, we record the refusal and we can choose another event button for experimentation. This can be repeated until (if it ever happens) we hit a successful button and an event is accepted. We can now move on to the execution of the next observable event by pressing buttons until another successful event is encountered and so on. The experiment can be terminated at any time. Of course, internal unobservable transitions can occur at any time without our knowledge, thereby introducing an element of nondeterminism into the process that will play a major role in the theory.

Now, our experimental record, or *trajectory* is given by

$$e = ((X_0, \sigma_1)(X_1, \sigma_2) \dots (X_{k-1}, \sigma_k), X_k), \qquad (61)$$

where k is the number of successful event transitions in the experiment, σ_i is the i-th successful event, X_0 (called the *initial refusal* of e), is the set of refused events prior to the first

Part II

Algebra of Discrete Event Processes

11 Concluding Remarks.

TIn this article we surveyed aspects of the theory of concurrency and process-algebra and showed how the theory can be adapted to deal with issues of DEP modeling and control. The proposed framework is suitable for modeling a wide range of process-interaction formalisms and is capable of dealing adequately with aspects of nondeterminism.

We believe that the algebraic approach to DEP modeling and control proposed here can alleviate some of the computational difficulties caused by high dimensionality of practical DEPs.

is then clear that if S is a supervisor for a process P, then

$$\mathcal{L}(S/P) = \mathcal{L}(P_\Sigma||_{\Sigma_c}S) \subseteq \mathcal{L}(P). \tag{56}$$

We can now introduce within our framework the concept of controllable languages.

Definition 10.1 Let $\mathcal{K}$ be a closed[6] sublanguage of $\mathcal{L}(P)$. $\mathcal{K}$ is said to be *controllable* if and only if there exists a supervisor S such that

$$\mathcal{K} = \mathcal{L}(P_\Sigma||_{\Sigma_c}S). \tag{57}$$

A characterization of controllability[7] is the following easily proved theorem:

Theorem 10.1 *A closed sublanguage $\mathcal{K} \subseteq \mathcal{L}(P)$ is controllable if and only if for all strings $t\sigma \in \mathcal{L}(P)$ such that $t \in \mathcal{K}$ and $\sigma \in \Sigma$,*

$$t\sigma \notin \mathcal{K} \;\Rightarrow\; \sigma \in \Sigma_c. \tag{58}$$

An immediate consequence of (48) (which is actually valid also for an arbitrary (not necessary finite) number of Processes)is the following

Proposition 10.1 *Let P be a process. The class of controllable sublanguages of $\mathcal{L}(P)$ is closed under set union.*

Consider now a process P, and let Δ, the deadlock process, serve as supervisor. The controlled process is then given as

$$(\Delta/P) = P_\Sigma||_{\Sigma_c}\Delta, \tag{59}$$

and it is clear from 51 that if S is any supervisor for P, then

$$\mathcal{L}(\Delta/P) \subseteq \mathcal{L}(S/P). \tag{60}$$

Thus, the language $\mathcal{L}(\Delta/P)$ is the smallest controllable sublanguage of P. We denote this sublanguage by $\mathcal{U}_P$ and call it the *uncontrollable* or *spontaneous* language of P.

Theorem 10.2 *Let P be a process. Let $\mathcal{K} \subseteq \mathcal{L}(P)$ be a nonempty closed sublanguale. If $\mathcal{U}_P \subseteq \mathcal{K}$, then $\mathcal{K}$ contains a unique (nonempty) supremal controllable sublanguage.*

[6] A language is closed if it includes all its prefixes [21].

[7] This characterization was used as definition of controllability by Wonham and Ramadge in [45].

Notice that the trajectory sets for the processes P and P' (both of which have the same failures set) are distinct. This distinction accounts for their different behavior under parallel composition that was evidenced in Example 9.1.

It has been shown (see part 2 of this report) that the algebraic identities (18)-(28) as well as the identities (32)-(35) also hold for the trajectory model with respect to $\mathcal{A}_t$. For parallel composition the following identities can be shown to be true:

$$P_A||_B P = P \tag{45}$$
$$P_A||_B Q = Q_B||_A P \tag{46}$$
$$(P_A||_B Q)_{A\cup B}||_C R = P_A||_{B\cup C}(Q_B||_C R) \tag{47}$$
$$P_A||_B(Q \oplus R) = (P_A||_B Q) \oplus (P_A||_B R) \tag{48}$$

We conclude this section with a list of some language relations that hold true for the trajectory model. These relations are useful in deriving certain properties of controlled DEPs that are discussed briefly in the following section.

$$A_1 \subseteq A_2 \Rightarrow \mathcal{L}(P_{A_2}||_B Q) \subseteq \mathcal{L}(P_{A_1}||_B Q) \tag{49}$$
$$\mathcal{L}(P) \subseteq \mathcal{L}(Q) \Rightarrow \mathcal{L}(P) \subseteq \mathcal{L}(P_A||_B Q) \tag{50}$$
$$\mathcal{L}(P_1) \subseteq \mathcal{L}(P_2) \Rightarrow \mathcal{L}(P_{1A}||_B Q) \subseteq \mathcal{L}(P_{2A}||_B Q) \tag{51}$$
$$\mathcal{L}(P_1) = \mathcal{L}(P_2) \Rightarrow \mathcal{L}(P_{1A}||_B Q) = \mathcal{L}(P_{2A}||_B Q) \tag{52}$$
$$A \subseteq B \Rightarrow \mathcal{L}(P_A||_B Q)_A||_B Q = \mathcal{L}(P_A||_B Q) \tag{53}$$
$$\mathcal{L}((P_A||_B Q)\backslash_{\Sigma-A}) \subseteq \mathcal{L}(P\backslash_{\Sigma-A}) \tag{54}$$

10 Aspects of Controllability

We conclude this paper with some remarks about controllability in discrete event control viewed within the framework of concurrency.

A behavioral specification for a DEP is, typically, a statement about languages. If $\Sigma_s \subseteq \Sigma$ is some event subset, then a *local specification* consists of a pair of languages $\underline{\mathcal{K}_s}$, $\overline{\mathcal{K}_s} \subseteq \Sigma_s^*$ such that $\mathcal{L}(P\backslash_{\Sigma-\Sigma_s})$, the language of the process localized to Σ_s, satisfies the constraint

$$\underline{\mathcal{K}_s} \subseteq \mathcal{L}(P\backslash_{\Sigma-\Sigma_s}) \subseteq \overline{\mathcal{K}_s}. \tag{55}$$

Sometimes $\underline{\mathcal{K}_s} = \emptyset$, and the specification consists of the upper-bound constraint only. If $\Sigma_s = \Sigma$, we call the corresponding specification *global.*

We shall assume that $\Sigma = \Sigma_u \cup \Sigma_c$, $A := \Sigma_u \cup \Sigma_c = \Sigma$ and $B = \Sigma_c$. In view of (54), it

To see that they do not behave the same way under prioritized synchronous composition, let them have priority set $A = \{a, b, d\}$ and run them in parallel with the process Q with priority set $B = \{a, b, c\}$, where

$$Q = a \to c \to b \to \Delta.$$

We obtain distinct results:

$$\begin{aligned} R = P_A||_B Q &= (a \to c \to \Delta) \oplus (a \to c \to b \to d \to \Delta), \\ R' = P'_A||_B Q &= (a \to c \to \Delta) \oplus (a \to c \to b \to \Delta). \end{aligned}$$

In view of the above, it is clear that the failures model is not a language congruence with respect to $\mathcal{A}_t = \mathcal{A}(\sigma \to \cdot, +, \oplus, (\cdot) \setminus {}_\sigma, \cdot_A||_B\cdot, \text{recursion})$.

We turn now to a brief discussion of a model, called the *trajectory-model*, that is a language-congruence with respect to $\mathcal{A}_t$, and which has been examinded in detail in part 2 of this report in the framework of a complete axiomatic theory. In the trajectory-model the process is specified by its set of *trajectories*. A process trajectory is a record of an 'experiment' that describes an *execution* of a string of events, and records in addition to the executed events, also the events that the process can reject (or refuse) after each successful event. A typical trajectory is then an object of the form

$$(X_0, \sigma_1, X_1, \ldots, X_{k-1}, \sigma_k, X_k)$$

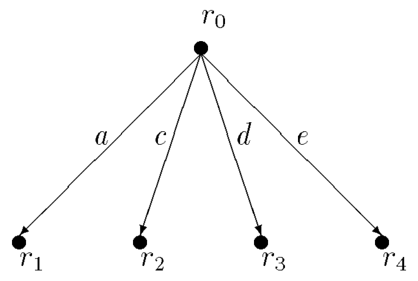
where σ_i denotes the ith successful event, where X_i denotes the set of events that can be refused after the ith executed event and where X_0 denotes the set of events that can be refused initially. The following is an example of the trajectory set of a process (listing only the trajectories of maximal length with maximal refusal sets).

Example 9.2 The set of trajectories of maximal length with maximal refusals of the process P of Example 9.1 is given by

$$\begin{aligned} \mathcal{T}(P) = \quad & \{(\{b, c, d\}, a, \{a, d\}, c, \{a, b, c, d\}), \\ & (\{b, c, d\}, a, \{a, d\}, b, \{a, b, c, d\}), \\ & (\{b, c, d\}, a, \{a, c, d\}, b, \{a, b, c, \}, d, \{a, b, c, d\})\}. \end{aligned}$$

The set of trajectories of maximal length (with maximal refusals) of the process P' of Example 9.1 is given by

$$\begin{aligned} \mathcal{T}(P') = \quad & \{(\{b, c, d\}, a, \{a, d\}, c, \{a, b, c, d\}), \\ & (\{b, c, d\}, a, \{a, c, d\}, b, \{a, b, c, d\}), \\ & (\{b, c, d\}, a, \{a, d\}, b, \{a, b, c, \}, d, \{a, b, c, d\})\}. \end{aligned}$$



Process R

Figure 12:

Finally, as was stated earlier, we identify process behavior with the language that it generates. Thus, we must guarantee that we have a behavioral model for DEPs that constitutes a language-congruence with respect to an algebraic framework that includes prioritized synchronous composition. We turn to this topic in the next section.

9 The Trajectory Model and Associated Algebra

We have seen in the previous section how the prioritized synchronous composition operator $\cdot_A||_B\cdot$ can be used to model a wide range of parallel composition formalisms and is, in particular, suitable for modeling dynamics and discrete-event control. We have also mentioned that the failures model captures adequately deadlock phenomena in nondeterministic behavior. It turns out, however, that in general, the failures model cannot adequately account for the range of possible interleavings that can occur in the framework of the operator $\cdot_A||_B\cdot$ when nondeterminism is also present. This is illustrated in the following simple example:

Example 9.1 Consider the two processes

$$P = (a \to ((c \to \Delta) + (b \to \Delta))) \oplus (a \to b \to d \to \Delta)$$
$$P' = (a \to ((c \to \Delta) + (b \to d \to \Delta))) \oplus (a \to b \to \Delta).$$

It is easily seen that P and P' have the same failures set which is given by[5]

$$\begin{aligned}\mathcal{F} = \quad & \{(\varepsilon, \{b, c, d\}), (a, \{a, d\}).(a, \{a, c, d\}), (ac, \{a, b, c, d\}), \\ & (ab, \{a, b, c\}), (ab, \{a, b, c, d\}), (abd, \{a, b, c, d\})\}.\end{aligned}$$

[5]again we list just the failures with maximal refusals.

■

Next, we consider an example of control with driven events.

Example 8.3 *A* Let Σ, P and S be defined as in Example 8.2 but suppose that the events consist of uncontrollable events and driven events. Thus, the set of uncontrollable events is $\Sigma_u = \{d, e\}$ as before, and the set of driven events is $\Sigma_d = \{a, b, c\}$. Clearly, the priority set A must include Σ_u in view of the physical nature of the uncontrollable events. However, the events of Σ_d may or may not be included in A, depending on the specific control mechanism employed (rather than on the physical character of the events). Thus, we shall say that driven events are synchronized in *closed-loop*, if they are included in the priority set A. In this case the supervisor S, that initiates the the driven events, waits for an acknowledgement that the triggered event is actually possible (and executed) before it proceeds with further state transitions of its own. Driven events that are not included in A are said to be executed in *open-loop*. In open-loop mode, the forcing process does not wait for acknowledgement.

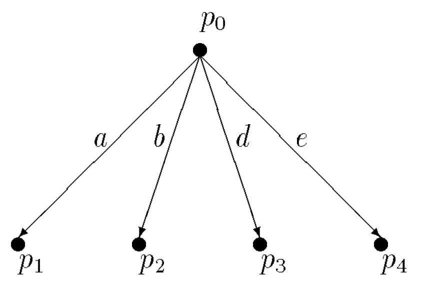
Thus, if we assume in our example closed-loop control with driven events, we have $A = \Sigma$ and $B = \{a, b, c\}$. It is easy to see that the controlled process R is again obtained as in Figure 11, and in fact there is no sharp distinction between closed-loop control with driven events and the enablement mechanism of controllable events, except for the physical interpretation.

Assume now, on the other hand, that the control of driven events is performed in open-loop. Then $A = \{d, e\}$ and $B = \{a, b, c\}$. Thus, the driven events will occur in the concurrent process whenever they triggered by, and hence occur in, S regardless of their actual occurrence in P. The controlled process in this case is obtained as in Figure 12 where $r_0 = (p_0, s_0)$, $r_1 = (p_1, s_1)$, $r_2 = (p_0, s_2)$, $r_3 = (p_3, s_0)$ and $r_4 = (p_4, s_3)$. Notice that the event c is executed without the participation of P while the event d is executed without participation of S.

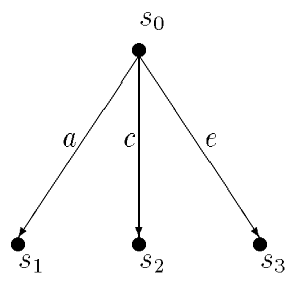
We can summarize the discussion of this Section with a formal classification of the events with respect to the priority sets A and B as follows: First, we have the requirement that

1. $A \supseteq \Sigma_u \cup \Sigma_c$,
2. $B = \Sigma_c \cup \Sigma_d$.

The subset $\Sigma_{dc} := \Sigma_d \cap A$ is then the set of closed-loop driven events and the set $\Sigma_{do} := \Sigma_d - \Sigma_{dc}$ is the set of open-loop driven events.



Process P

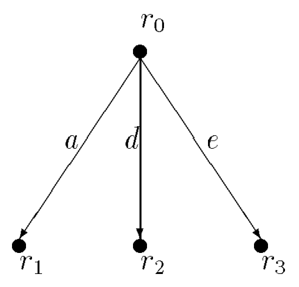


Process S

Figure 10:

state transitions in response to their occurrence in P. The controllable events, however, will not occur in the process unless they are enabled by the supervisor. Thus, the controllable events must be in the supervisor's priority set B, and we have $B = \{a, b, c\}$. Notice that in this case $\Sigma - A \cup B = \emptyset$, $A \cap B = \Sigma_c$, $A - A \cap B = \Sigma_u$, and $B - A \cap B = \emptyset$.

In our example the controlled process $R := P_A||_B S = P_\Sigma||_{\Sigma_c} S$ is then obtained as



Process $R = P_\Sigma||_{\Sigma_c} S$

Figure 11:

where $r_0 = (p_0, s_0)$, $r_1 = (p_1, s_1)$, $r_2 = (p_3, s_0)$ and $r_3 = (p_4, s_3)$. Notice that the controllable events a and c are both enabled by S while the event b is not. Hence in the controlled process R, the event a is present and will occur if it occurs in P. The events b and c do not appear in R; the first because it is not enabled by S and the second because it is not possible in P. The events d and e appear in R and occur whenever they occur in P regardless of their possibility (or lack of it) in S. Thus, if e happens in P, then S participates synchronously while if d happens in P, then S remains in its initial state s_0.

respect to a given event, depends not just on the event, but also on the context and event availability.

∎

We turn now to an examination of how our prioritized synchronous composition can model control of DEPs. First we remark that DEPs frequently have other mechanisms for interaction with the environment than the one investigated by Ramadge and Wonham. For example, the process may possess also *driven* events that must be forced or triggered by the environment in order to take place. Driven events are then distinguished from controllable events in that when they are possible in the process and triggered by the controller, they are guaranteed to take place immediately and instantaneously.

We shall let P denote the process under consideration, and let S denote the *supervisor*. The controlled, or *closed-loop*, process is then given by

$$R = (S/P) := P_A||_B S, \tag{43}$$

where the priority sets A and B are suitably chosen so as to correctly model the physical environment. The event set Σ will be partitioned into three disjoint subsets

$$\Sigma = \Sigma_u \cup \Sigma_c \cup \Sigma_d, \tag{44}$$

where Σ_u is the subset of uncontrollable events, Σ_c is the set of controllable events, and Σ_d is the set of *driven* events. We turn now to two examples. First we consider an example in the Ramadge-Wonham framework, that is, $\Sigma_d = \emptyset$.

Example 8.2 Let the event set be $\Sigma = \{a, b, c, d, e\}$, and consider the simple processes P and S of Figure 10.

Suppose P is the controlled process and let the subset of controllable events be $\Sigma_c = \{a, b, c\}$, and the subset of uncontrollable events be $\Sigma_u = \{d, e\}$. Thus, the events in Σ_c can be disabled while the events in Σ_u cannot. Let us now see how we can model control of process P by supervisor S through prioritized synchronous composition of P and S. Since all events are assumed to be spontaneous events of the process P, its priority set A must include them all, that is, $A = \Sigma$. The uncontrollable events cannot be influenced by the environment. Thus, the priority set B (of the supervisor S) must not include any uncontrollable event of P. This means that uncontrollable events of P cannot be blocked by the supervisor, but the supervisor may (if the designer so wishes) execute (concurrently) (ii). The set $\Sigma - A \cup B$ of *broadcast* synchronization events. Each process can offer these events for execution and the other process will participate in their execution synchronously if it can. But if it cannot (i.e., if the event is impossible in its current state), the initiating process will execute the event by itself.

(iii). The set $A - A \cap B$ of *priority events* of process P. The execution of these events will take place if and only if the process P participates. The participation of the process Q in these events will take place whenever possible, i.e., whenever Q can in its respective state. But lack of participation by Q cannot block execution by P.

(iv). The set $B - A \cap B$ of *priority events* of process Q. (Similar to (iii) above.)

To illustrate the prioritized synchronous composition, consider the following simple example:

Example 8.1 Let $\Sigma = \{a, b, c\}$ and consider the parallel composition of processes P and Q as described in Figure 9, where $A = \{a, c\}$ and $B = \{a, b\}$.

P $\qquad$ Q $\qquad$ $R = P_A||_B Q$

Figure 9:

Observe that the event a occurs only when both processes P and Q participate in the execution taking R from state $r_4 = (p_1, q_1)$ to $r_3 = (p_2, q_2)$. However the event c occurs in R either by participation of both processes, for example in transition from $r_1 = (p_0, q_1)$ to $r_2 = (p_1, q_0)$, or by execution of P alone, in case the event is not available in Q, as for example in transition from $r_0 = (p_0, q_0)$ to $r_2 = (p_1, q_0)$. Notice also that the transition of process P from p_2 to p_0 never takes place when it runs concurrently with Q because the event b is in the priority set of Q, but Q is never at state q_0 when P is at p_2. The important property that is demonstrated above and that distinguishes prioritized synchronous composition from other concurrency operators, is the fact that the behavior of the concurrent process with

8 Concurrency by Prioritized Synchronization

In the present section we introduce a new concurrency operator, called *prioritized synchronous composition* that is suitable for modeling a wide range of practical control formalisms.

Let the event set be denoted by Σ and consider two processes P and Q with events in Σ. With each process we associate a subset of special events, called its set of *priority* (or *blocking*) events. These are events in whose execution the given process must participate; otherwise they cannot take place. Thus, let $A, B \subseteq \Sigma$ be the priority sets of P and Q, respectively, and define the prioritized synchronous composition of P and Q, denoted $P_A||_BQ$, as follows:

$$P : p \overset{\sigma}{\to} p' \;\;\&\;\; Q : q \overset{\sigma}{\to} q' \;\;\Rightarrow\;\; P_A||_BQ : (p,q) \overset{\sigma}{\to} (p',q') \tag{40}$$

$$P : p \overset{\sigma}{\to} p' \;\;\&\;\; Q : q \overset{\sigma}{\to} \backslash \;\;\Rightarrow\;\; P_A||_BQ : (p,q) \overset{\sigma}{\to} \begin{cases} (p',q) & \text{if } \sigma \notin B \\ \backslash & \text{if } \sigma \in B \end{cases} \tag{41}$$

$$Q : q \overset{\sigma}{\to} q' \;\;\&\;\; P : p \overset{\sigma}{\to} \backslash \;\;\Rightarrow\;\; P_A||_BQ : (p,q) \overset{\sigma}{\to} \begin{cases} (p,q') & \text{if } \sigma \notin A \\ \backslash & \text{if } \sigma \in A. \end{cases} \tag{42}$$

Expression (40) states that if, at their respective states, both processes P and Q can execute a given event σ, then it will be executed concurrently (i.e., in synchronization) in both processes. Both processes will then undergo simultaneosly their respective state transitions. Notice that, when both processes can execute an event concurrently, the mathematical model does not distinguish which process initiates the event. Indeed, as we shall see shortly, this is a matter for the physical interpretation. Expressions (41) and (42) define the concurrency operator in case that an event is possible in (initiated by) one of the processes but is not possible in the other: In this case, the initiating process will execute the event without participation of the other, unless the event is in the priority set of the latter; In this case the execution of the event is blocked.

It is not difficult to see that the prioritized synchronous composition operator partitions the event set Σ into four distinct (and disjoint) subsets:

(i). The set $A \cap B$ of *strict-synchronization* events. These events are either executed by both processes concurrently or by none.

As it turns out, this is not quite as straightforward when we introduce dynamics, or uncontrollable events. Let us first try to clarify the synchronization status of the various events. Clearly, the controllable events must belong to the synchronization set as before, because it takes the supervisor to enable an event and the process to execute it. But what about the uncontrollable events? If an uncontrollable event is possible both in P and in S (at their respective states), its occurrence in the concurrent process must be given the physical interpretation as having been executed in S in response to its (spontaneous) occurrence in P. If it is possible only in P, but not in S, it will still occur in P, and hence in the concurrent process, because of its uncontrollability. But if an uncontrollable event is possible only in S, it will not occur because S cannot initiate the event.

Let us reconsider the above example, but this time assume that the event d is uncontrollable. Thus let $\Sigma'_c = \{a, b, c\}$ and $\Sigma'_u = \{d\}$. Let us reexamine the process $R = P||_{\Sigma}S$ of Figure 7. The event d appears in R after the occurrence of b but not after a. This is physically incorrect because once a occurs, the event d cannot be blocked by its absence in the supervisor. If, on the other hand, we remove the uncontrollable events from the synchronization set, and try to model controlled behavior by $R' = P||_{\Sigma_c}S$, we would obtain the process R' as in Figure 8, which is unsatisfactory because it permits the occurrence of the event d without participation of P, which is impossible in the physical process.

$$R' = P||_{\{a,b,c\}}S$$

Figure 8:

Thus, it is clear that concurrency with strict synchronization cannot be used as a satisfactory framework for modeling the interaction of dynamical discrete-event processes with their environment. More specifically, strict concurrency is an inadequate formalism for modeling control of discrete event processes within the Ramadge-Wonham framework (unless we impose special restrictive conditions on the supervisor[4]).

[4]such as the condition of supervisor completeness [45]

Let $\mathcal{L}_c(G)$ denote the language generated by G under control, i.e., in closed loop. Then it is clear that the domain of the map h can be restricted to $\mathcal{L}_c(G)$. In practice, it is convenient to use a state machine realization for $\mathcal{L}_c(G)$. Thus one defines $S = (\Sigma, X, \xi, x_0)$ as the automaton realizing $\mathcal{L}_c(G)$, and the map h is replaced by a *feedback* map $\phi : X \to \Gamma$ such that for $s \in \mathcal{L}_c(G)$

$$\phi(\xi(s, x_0)) = h(s), \tag{39}$$

where $\xi(s, x_0)$ is the standard extension of the transition map to strings [21].

7 Strict Concurrency and Discrete Event Control

A key element in the Ramadge-Wonham control problem formulation, is the introduction of what may be thought of as discrete-event *dynamics*, where by dynamics we refer to the presence of *spontaneity*, that is, the existence of events whose occurrence cannot be influenced by the environment.

Let us next examine the possibility of modeling the control of discrete event processes using the formalism of strict concurrency as described in Section 3. To this end consider first the simple control problem described in Figure 7.

P $\qquad\qquad$ S $\qquad\qquad$ $R = P||_{\Sigma}S$

Figure 7:

Here all events of the process P are controllable, that is $\Sigma_c = \Sigma = \{a, b, c, d\}$ and $\Sigma_u = \emptyset$. The process S is to be thought of as the *supervisor* for P, with supervision achieved through concurrency with full synchronization. Specifically, when P is at state p_i and S is at state s_j, then the possibility of occurrence of an event, say a, in $R = P||_{\Sigma}S$ at state $r_k = (p_i, s_j)$, means that the event is enabled by S and possible (subject to enablement) in P. An event in R is, thus, interpreted as enabled by S and occurring in P, and the participation of both processes in an event is essential for its occurrence. Thus, when all events are controllable, we can model control by strict concurrency with full event synchronization.

6 The Ramadge-Wonham Discrete-Event Control Formalism

In their pioneering work on the control of DEPs, Ramadge and Wonham (RW) [45, 49, 50] introduced the following formalism. A DEP is modeled as a deterministic state-machine or automaton, called *generator*, which is given by a 4-tuple[3]

$$G = (\Sigma, Q, \delta, q_0) \tag{36}$$

where Q is a set of states, Σ is a set of events, $\delta : \Sigma \times Q \to Q$ is a partial function called the *transition function*, and q_0 is the *initial* state. The statement that δ is a partial function means that it need not be defined for all pairs $(\sigma, q) \in \Sigma \times Q$.

Control is introduces as follows. It is assumed that all events occur in the process spontaneously and asynchronously, but some of the events have a mechanism for their disablement at any time. Thus the event set Σ is partitioned into two disjoint subsets

$$\Sigma = \Sigma_u \cup \Sigma_c \tag{37}$$

where Σ_c is the subset of events that can be disabled, called *controllable events*, and Σ_u is the subset of events whose occurrence cannot be disabled, called *uncontrollable*. A *control input* for G is now defined as a subset $\Gamma \subseteq \Sigma_c$ of events that are disabled at any instant of time. Control of a DEP consists of switching the disablement set Γ as the process progresses in its run. With this event-set partition and associated disablement mechanism the DEP is called a controlled DEP, or CDEP.

The control execution is performed by a *supervisor* which can abstractly be thought of as a map

$$h : \mathcal{L}(G) \to \Gamma. \tag{38}$$

Concretely, this means that after every event that takes place in the process, a new event set is supplied to the process for disablement. Thus when the CDEP is supervised by a supervisor h, the generator G must be modified by redifining the transition map δ as $\hat{\delta}$, where

$$\widehat{\delta}(\sigma, q) := \begin{cases} \delta(\sigma, q) & \text{if } \sigma \in \Gamma \\ undefined & \text{otherwise.} \end{cases}$$

[3]Actually, Ramadge and Wonham have a somewhat more general setting where a DEP is a 5-tuple that includes also *marker* states, but these are inessential to the present exposition.

5 Process Algebra

By a *Process Algebra* we refer to a set of algebraic identities between process expressions. Such an algebra can then be used to manipulate, combine and simplify process expressions and perform a variety of computations with processes symbolically rather than explicitly. We have already encountered in the foregoing several algebraic process identities, and a simple example of their use in computational simplification was seen in Example 4.1. The chief utility of a behavioral (or semantic) modeling framework of processes, is in establishing the algebraic identities. It is for this reason that we must guarantee that the modeling framework constitutes a behavioral (in our case, a language) congruence. The derivation of these identities are beyond the scope of the present paper but for illustrational purposes we give below a partial list of algebraic identities that are valid for the failures model with respect to $\mathcal{A}_f$ (see e.g., [35] for details).

$$P + Q = Q + P \tag{18}$$
$$(P + Q) + R = P + (Q + R) \tag{19}$$
$$P + P = P \tag{20}$$
$$P + \Delta = P \tag{21}$$
$$P \oplus Q = Q \oplus P \tag{22}$$
$$(P \oplus Q) \oplus R = P \oplus (Q \oplus R) \tag{23}$$
$$P \oplus P = P \tag{24}$$
$$(P + Q) \oplus R = (P \oplus R) + (Q \oplus R) \tag{25}$$
$$(P \oplus Q) + R = (P + R) \oplus (Q + R) \tag{26}$$
$$(\sigma \to P) + (\sigma \to Q) = (\sigma \to P) \oplus (\sigma \to Q) \tag{27}$$
$$(\sigma \to P) \oplus (\sigma \to Q) = (\sigma \to P \oplus Q) \tag{28}$$
$$P||_A P = P \tag{29}$$
$$P||_A Q = Q||_A P \tag{30}$$
$$P||_A(Q \oplus R) = (P||_A Q) \oplus (P||_A R) \tag{31}$$
$$(P\backslash_a)\backslash_b = P\backslash_{a\cup b} \tag{32}$$
$$(P \oplus Q)\backslash_a = P\backslash_a \oplus Q\backslash_a \tag{33}$$
$$(b \to P)\backslash_a = \begin{cases} P\backslash_a & \text{if } b = a \\ (b \to P\backslash_a) & \text{if } b \neq a \end{cases} \tag{34}$$
$$((a \to P) + Q)\backslash_a = P\backslash_a \oplus (P + Q)\backslash_a \tag{35}$$

Example 4.1 Consider the process $R := \hat{P}||_{\Sigma}Q$, where $Q = (c \to \Delta)$. Using the definition of parallel composition with full synchronization as given by (10)-(12), we obtain

$$R = (c \to \Delta).$$

Next, consider the process $R' := P\setminus_a||_{\Sigma}Q$. While in this simple example the computation of R' can be performed directly without difficulty, we shall take the opportunity to demonstrate the use of process-algebra in computational simplification. First we shall use Equation (5) to obtain

$$R' = (((b \to \Delta) + (c \to \Delta)) \oplus (b \to \Delta))||_{\Sigma}(c \to \Delta). \tag{17}$$

Next we use the following identity (see e.g., [10])

$$(P \oplus Q)||_A R = (P||_A R) \oplus (Q||_A R),$$

which together with (17) gives

$$\begin{aligned} R' &= ((b \to \Delta) + (c \to \Delta)||_{\Sigma}(c \to \Delta)) \oplus ((b \to \Delta)||_{\Sigma}(c \to \Delta)) \\ &= (c \to \Delta) \oplus \Delta, \end{aligned}$$

where the last equality is obtained with the aid of (10)-(12). Comparing R with R', we see that R' can deadlock initially, while R cannot. Indeed, the choice of whether R' will initially deadlock or not, is completely nondeterministic. This nondeterminism can best be understood upon noting that $P\setminus_a$ can undergo a silent transition from p_0 to p_1 (see Figure 1), and there is no observable mechanism to guarantee that the event c be offered by Q prior to such transition.

The above example illustrates the fact that the language model is not a language-congruence when nondeterminism is present. Specifically, the language model cannot adequately express the possibility of deadlock. This fact motivated the introduction by [10] (see also [9, 20, 34, 35]) of the more sophisticated *failures-model*. This model, which is obviously more detailed than the language model, represents a process by its *failures set* $\mathcal{F} = \{(s, X)\}$, where a failure (s, X) consists of a *trace* s, i.e., a string of events that the process can execute, and a *refusal set* X that consists of the events that the process can reject (or refuse) after the execution of s. We shall not elaborate here on the failures model except for giving a simple illustrative example.

Example 4.2 The failures set of the process $P\setminus_a$ of Example 2.1 is given by[2]

$$\mathcal{F}(P\setminus_a) = \{(\varepsilon, \emptyset), (\varepsilon, \{c\}), (b, \{b, c\}), (c, \{b, c\})\}.$$

[2]We give here only the failures with maximal refusals.

event-alphabet Σ. If $\mathcal{E}_1, \mathcal{E}_2 \in \mathcal{C}$ are two equivalence relations, we say that $\mathcal{E}_1$ is *coarser* than $\mathcal{E}_2$, denoted $\mathcal{E}_1 \succeq \mathcal{E}_2$, if for any pair of DEPs P and Q,

$$P\mathcal{E}_2 Q \Rightarrow P\mathcal{E}_1 Q.$$

It is easily seen that $\succeq$ constitutes a complete partial order on DEPs [30]. We now have the following

Definition 4.2 A DEP modeling framework is called *efficient* if it induces the coursest language-congruence with respect to $\mathcal{A}$.

Thus, a modeling framework is efficient if it includes in the model of a DEP the least amount detail necessary to distinguish DEPs that differ in behavior, but identifies all processes that cannot be distinguished behaviorally. It is important to realize that the detail needed in the model is crucially dependent on the operators that are included in $\mathcal{A}$. As their expressiveness increases, the complexity of the models must, in general, increase as well.

Definition 4.3 A framework $\mathcal{A}$ is called *deterministically closed* if for each $f \in \mathcal{A}$, $f(P)$ is deterministic whenever P id deterministic.

It can be shown that (see e.g. [35]), if A is deterministically closed, then $\mathcal{L}(P)$ is an adequate model for P. That is, $\mathcal{L}$ itself constitutes a language congruence. Obviously $\mathcal{L}$ is then the coarsest language congruence. The reader can convince himself without too much difficulty that $\mathcal{A}_l = \mathcal{A}(\sigma \to \cdot, +, \cdot||_A\cdot, \text{recursion})$ is deterministically closed. Thus, the behavior of deterministic processes that interact only through strict synchronization, can be adequately modeled by their languages[1].

We turn now to the case $\mathcal{A}_f = \mathcal{A}(\sigma \to \cdot, +, \oplus, (\cdot)\backslash_\sigma, \cdot||_A\cdot, \text{recursion})$. That is, $\mathcal{A}_f$ includes also the operators of uncontrolled alternative and event internalization. Nondeterminism is now included in our framework.

It is of interest, at this stage, to return to the question raised in Example 2.1 of comparing the processes $\hat{P} = (b \to \Delta) + (c \to \Delta)$ and $P\backslash_a$ where $P = (a \to b \to \Delta) + (c \to \Delta)$, both of which generate the same languages. To this end, let us consider the following example that shows that processes $\hat{P}$ and $P \backslash_a$ are not language congruent.

[1]This fact has been of key importance in the interesting work of Smedinga [46] on control of discrete events.

4 Process Models and Language-Congruence

In the present section we discuss certain questions regarding DEP modeling. The main purpose of a mathematical model of a DEP is to describe its behavior. We must, therefore, require of a model to capture enough detail about the DEP's structure, so as to ensure that its behavior is fully exhibited in all circumstances. A model can be regarded as *efficient* if it captures just enough detail (for our purposes) but no more detail than necessary. Thus, an efficient model must not distinguish between DEPs that, in a given framework, exhibit identical behavior. Next we proceed to make these ideas somewhat more precise.

As we have already seen earlier, in a DEP modeling environment, DEPs are given by algebraic expressions whose arguments are also DEPs. The range of such algebraic expressions is determined by the range of algebraic operators that are defined in the given framework. Let us denote such a framework by $\mathcal{A} = \mathcal{A}(O_1, \ldots, O_k)$, where $O_1, \ldots, O_k$ are the operators under consideration. In the context of the framework exhibited thus far, the operators include the prefix operator, the alternative operators, the internalization operator, the recursion and, most importantly, the operator of strict concurrency.

By the behavior of a process P, we refer to the language $\mathcal{L}(P) \in \Sigma^*$, consisting of all event strings, or traces, that P generates. Let $\mathcal{M}$ denote a modeling framework for DEPs, so that $\mathcal{M}(P)$ denotes a model for a DEP P. Then $\mathcal{M}$ induces an equivalence relation, denoted $\mathcal{E}_{\mathcal{M}}$, on the class of all DEPs under consideration. Specifically, we then say that DEPs P and Q are equivalent, denoted $P\mathcal{E}_{\mathcal{M}}Q$, whenever $\mathcal{M}(P) = \mathcal{M}(Q)$. Clearly then for the modeling framework $\mathcal{M}$ to be adequate, we must require that if $\mathcal{M}(P) = \mathcal{M}(Q)$, then P and Q must exhibit the same behavior under all circumstances. This leads us to the following

Definition 4.1 An equivalence relation $\mathcal{E}_{\mathcal{M}}$ on the class of DEPs is called a *language-congruence* (with respect to $\mathcal{A}$) if for every $f \in \mathcal{A}$ and any two DEPs P and Q,

$$P\mathcal{E}_{\mathcal{M}}Q \Rightarrow \mathcal{L}(f(P)) = \mathcal{L}(f(Q)). \tag{16}$$

In the above definition $f(P)$ denotes an expression with P as an argument. (We do not preclude the possibility that f is an expression in more than a single argument, in which case our notation implies that the other arguments are held fixed.)

Thus, in terms of the above definition, an adequate modeling framework must induce a language-congruence. But this, of course, does not guarantee that the modeling framework is efficient. Let $\mathcal{C}$ denote the set of all equivalence relations on the class of DEPs over a fixed

An example parallel composition with full synchronization is given in Figure 5.

P $\qquad$ Q $\qquad$ $P||_{\Sigma}Q$

Figure 5:

A generalization of the synchronization convention, that includes parallel composition by interleaving and parallel composition by intersection as special cases, is given by the operator $P||_A Q$, where $A \subseteq \Sigma$ is an arbitrary subset called the *synchronization* set. Informally, this is the process obtained when P and Q run independently in parallel, except that they must fully synchronize their events in A. This operator is defined formally by

$$P : p \xrightarrow{\sigma} p' \;\&\; Q : q \xrightarrow{\sigma} q' \;\Rightarrow\; P||_A Q : (p,q) \xrightarrow{\sigma} \begin{cases} (p',q') & \text{if } \sigma \in A \\ (p',q) \text{ or } (p,q') & \text{otherwise} \end{cases} \tag{13}$$

$$P : p \xrightarrow{\sigma} p' \;\&\; Q : q \xrightarrow{\sigma} \backslash \;\Rightarrow\; P||_A Q : (p,q) \xrightarrow{\sigma} \begin{cases} (p',q) & \text{if } \sigma \notin A \\ \backslash & \text{otherwise} \end{cases} \tag{14}$$

$$Q : q \xrightarrow{\sigma} q' \;\&\; P : p \xrightarrow{\sigma} \backslash \;\Rightarrow\; P||_A Q : (p,q) \xrightarrow{\sigma} \begin{cases} (p,q') & \text{if } \sigma \notin A \\ \backslash & \text{otherwise.} \end{cases} \tag{15}$$

An example of parallel composition with partial synchronization is given in Figure 6.

P $\qquad$ Q $\qquad$ $P||_{\{a\}}Q$

Figure 6:

without synchronization, which is modeled by the interleaving behavior of the component processes. We shall denote this parallel composition by $(\cdot \,||_{\emptyset}\, \cdot)$, where the subscript $\emptyset (\subseteq \Sigma)$ denotes the fact that the set of synchronized events is empty. Thus, if P and Q are DEPs, then the DEP $P||_{\emptyset}Q$ is the process obtained from operating P and Q in parallel completely independently. The only assumption that is generally made about this parallel operation is that events of P and Q never coincide in time. (An exception to this assumption can be found e.g. in [32].) Using our notational convention, we can thus define the operator of parallel composition without synchronization, formally, by

$$P : p \xrightarrow{\sigma} p' \;\Rightarrow\; P||_{\emptyset}Q : (p,q) \xrightarrow{\sigma} (p',q) \tag{8}$$
$$Q : q \xrightarrow{\mu} q' \;\Rightarrow\; P||_{\emptyset}Q : (p,q) \xrightarrow{\mu} (p,q'). \tag{9}$$

As an example of process interleaving consider the simple processes P and Q in Figure 4.

P $\qquad$ Q $\qquad$ $P||_{\emptyset}Q$

Figure 4:

At the other extreme of the range of possible synchronizations, is the parallel composition with full synchronization, denoted $(\cdot \,||_{\Sigma}\, \cdot)$. In this case the synchronization of events is complete in that all events in the event set Σ must be synchronized. Thus, if P and Q are Σ-processes, i.e., processes over the event set Σ, then an event in $P||_{\Sigma}Q$ can take place if and only if it can take place simultaneously (and synchronously) in both processes. If one of the processes cannot participate in an event initiated by the other, the event will not take place in either process. If no common events exist at a given time, the composite process $P||_{\Sigma}Q$ *deadlocks*. Parallel composition with full synchronization can thus be defined formally by

$$P : p \xrightarrow{\sigma} p' \;\&\; Q : q \xrightarrow{\sigma} q' \;\;\Rightarrow\;\; P||_{\Sigma}Q : (p,q) \xrightarrow{\sigma} (p',q') \tag{10}$$

$$P : p \xrightarrow{\sigma} p' \;\&\; Q : q \xrightarrow{\sigma} \backslash \;\;\Rightarrow\;\; P||_{\Sigma}Q : (p,q) \xrightarrow{\sigma} \backslash \tag{11}$$

$$Q : q \xrightarrow{\mu} q' \;\&\; P : p \xrightarrow{\mu} \backslash \;\;\Rightarrow\;\; P||_{\Sigma}Q : (p,q) \xrightarrow{\mu} \backslash \tag{12}$$

The above operator is sometimes also called composition by *intersection* because the trace set of $P||_{\Sigma}Q$ is easily seen to be precisely the intersection of the trace sets of P and of Q.

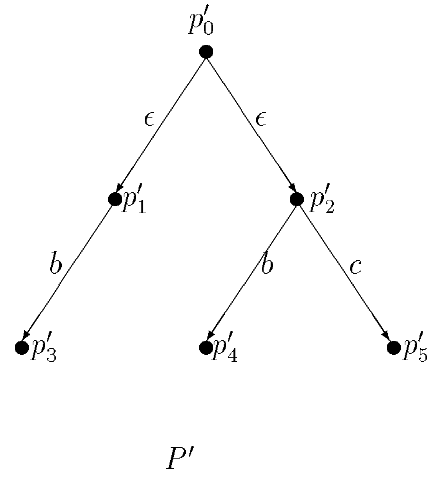


Figure 2:

Example 2.2 The (recursive) solution to the fixed point equation

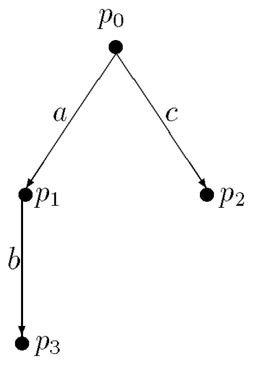
$$P = (a \to b \to P) + (c \to P) \tag{7}$$

is the process whose transition graph is given in Figure 3 below.

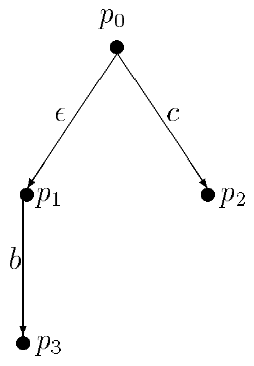
Figure 3:

3 Formalisms of Concurrency

Processes interact with their environment through communication. That is, they operate in parallel with a specified degree of event synchronization. Thus we speak of *parallel composition* or *concurrency* of DEPs. Various formalisms of concurrency have been studied in the computer science literature. The simplest form of concurrency is parallel composition



Process P



Process $P \setminus_a$

Figure 1:

Definition 2.1 A DEP P is called *deterministic* if it has no silent transitions and for every state p of P and every event $\sigma \in \Sigma$ there is at most one state p' such that $P : p \overset{\sigma}{\rightarrow} p'$.

An interesting and important question is how the process $P\setminus_a$ of Example 2.1 differs from the deterministic process $\hat{P} := (b \rightarrow \Delta) + (c \rightarrow \Delta)$ which generates the same event-strings (or *traces*). We shall return to this and related questions in some more detail later but in the meantime we shall only note that the following identity holds true:

$$((a \rightarrow b \rightarrow \Delta) + (c \rightarrow \Delta))\setminus_a = ((b \rightarrow \Delta) + (c \rightarrow \Delta)) \oplus (b \rightarrow \Delta). \tag{5}$$

Equation 5 means that the process $P\setminus_a$ can be identified in some sense with the process P' whose state transition graph is depicted in Figure 2. In the process P' there is an initial nondeterministic (unobserved) transition from p_0' to either p_1' or p_2' after which it becomes deterministic. The identification of two distinct processes like that in Equation 5 is at the heart of a process-algebra and we shall return to this issue later.

We shall conclude this section with a brief discussion of *recursive* equations for process description.

An equation of the form

$$P = f(P), \tag{6}$$

where f is a function of (or an operator on) P, is a *fixed-point* equation, which, under suitable conditions (see e.g., [30, 20]), has a recursive solution (for P). Under appropriate restrictions this solution is even unique. Fixed point equations are a convenient way for process formulation. A simple illustration is given by the following example.

This allows us to introduce the important *prefix* operator (or prefix construction) by defining the process Q as

$$Q := \sigma \to P. \tag{1}$$

That is, Q is the process that starts at its initial state (say q_0) and, in response to event σ, undergoes transition to P. For example, if $P = \Delta$, the *deadlock-process* (that cannot undergo any state transitions), then (1) means that Q is the process that can execute (or respond to) event σ and then deadlock.

Another important process-operator is the *controlled alternative* operator $+$, which is defined as follows. Let $Q_1 = \sigma_1 \to P_1$ and $Q_2 = \sigma_2 \to P_2$. Then

$$Q := Q_1 + Q_2 = (\sigma_1 \to P_1) + (\sigma_2 \to P_2), \tag{2}$$

is the process that in its initial state can either respond to σ_1 and undergo transition to P_1, or respond to σ_2 and undergo transition to P_2. The choice of the initial event is at the disposal of the environment.

An important element of nondeterministic process behavior is provided by the *uncontrolled alternative* operator $\oplus$. A simple illustration of this operator is provided by the following situation. If $Q_1 = \sigma \to P_1$ and $Q_2 = \sigma \to P_2$, then

$$Q := Q_1 \oplus Q_2 = (\sigma \to P_1) \oplus (\sigma \to P_2) = (\sigma \to P_1 \oplus P_2). \tag{3}$$

This is the process that, in response to the initial event σ, undergoes transition either to P_1 or to P_2, but the choice is completely nondeterministic. Actually, as we shall see shortly, this operator is much more subtle than indicated by (3). First we need to introduce the *event-internalization* operator.

Let P be a process with event set Σ. By the internalization of an event $\sigma \in \Sigma$, we refer to the removal of all occurrences of the event σ from external view so that all state transitions associated with σ become *silent*, or unobserved, (denoted by ϵ). We denote the resultant process by $P\backslash_\sigma$.

Example 2.1 Consider the process $P\backslash_a$ where P is given by

$$P = (a \to b \to \Delta) + (c \to \Delta). \tag{4}$$

Notice that the process $P\backslash_a$ possesses nondeterministic behavior in that the internalized event can occur at any time without the explicit knowledge of the observer. Thus we may not know whether the process is at state p_0 or at p_1.

is modeled by *parallel composition* with a specified degree of *event synchronization.* While various formalisms of parallel composition have been defined and investigated in the literature, they all rely on some framework of *strict synchronization.* That is, specific events of distinct processes must either strictly synchronize or be completely independent and interleave. These formalisms are inherently inadequate for modeling the interaction of DEPs in which spontenaity of events is an essential behavioral feature.

The present paper is a tutorial introduction to the theory of concurrency and to the associated process-algebra and the suitability of such a methodology for modeling and control of DEPs is examined. It is shown that the existing formalisms of synchronization are inadequate for modeling the interaction of (dynamic) DEPs with the environment. Accordingly, a new parallel composition operator, called *prioritized synchronous composition*, that can model a wide range of interactions among DEPs, is introduced. Aspects of the corresponding process-algebra are examined. Finally, some comments are made about aspects of controllability within the framework of the new methodology. A more detailed and formal account of the new algebra of DEPs can be found in part 2 of this report.

2 Process Components and Operators

Following standard notation, let Σ be a finite set of event labels and let Σ^* denote the set of all finite strings of elements of Σ, including the empty string ε. A process P with events in Σ is then a device that undergoes state transitions in response to events in Σ. A *local* description of P can be given in terms of individual state transitions as follows. If p and p' are states of P, and σ is an event in Σ, then we shall use the notation

$$P : p \overset{\sigma}{\rightarrow} p'$$

to express the possibility for process P to undergo transition from state p to state p' in response to the event σ. Similarly, we shall use the notation

$$P : p \overset{\sigma}{\rightarrow} \backslash$$

to express the fact that when the process P is at state p, no state transition is possible in response to the event σ. At this stage we do not concern ourselves with the mechanism of event generation.

We shall also find it convenient to refer formally to P as the global process structure consisting of its complete state transition tree, or graph, and its designated initial state p_0.

studied in the Ramadge-Wonham framework. Their research had a profound impact on the control systems research community and generated a growing interest in control of DEPs as evidenced by the expanding number of research contributions to this subject (e.g. [11, 12, 22, 23, 25, 26, 48, 7, 8]).

In spite of their inherent simplicity and corresponding attractiveness, state machines have a weakness as models of complex processes because they suffer from an exponential explosion in the number of their states. To be effective and useful, it is desirable that a state/event modeling formalism have the capability to somehow relax the requiremwnt that all states as well as all event sequences be present explicitly in the model at all times. Thus, one would like to be able to suppress in such a model all aspects of its description that are irrelevant in a particular context. This can be achieved by event-internalization, or partial observation, which leads to nondeterminism in process behavior (in the automata-theory sense) and to inadequacy of formal languages as models of behavior. A further aspect of effective modeling is the ability to construct a process description from individual components, thus introducing as an intergral element of the modeling framework modularity and hierarchy. Also, to obtain an effective description tool, it is important to have the capability of describing behavior recursively. Finally, since all modules of the process must interact and correctly synchronize when operating in parallel, asuitable mechanism for communication and interaction between the various process components must be formulated, that includes a suitable formalism for DEP control.

The importance of developing a framework for modeling, specification, verification and synthesis of discrete event processes, with particular emphasis on computer operating systems, data-base management, concurrent programs, and distributed computing, has been recognized in the computer science community for well over a decade, and a diverse and extensive literature has developed on this subject. Notable among the various approaches that have been developed are Petri-Net Theory [39], linear-time and branching-time temporal logics [13, 31, 40, 24], and, of particular interest in the context of the present paper, a number of (closely related) algebras of concurrent processes that were inspired by Hoare's Communicating Sequential Processes (CSP) [20] and Milner's Calculus of Communicating Systems (CCS) [33], and became widely known as the theory of *concurrency* [9, 10, 18, 32, 34, 4, 5]. (The reader is referred to the two recent volumes [2] and [3] for a broad overview of the current literature.)

In spirit and in general philosophy, the theory of concurrency is well suited for modeling, analysis, and synthesis of discrete event control systems. A central theme in that theory is the description of the interaction between DEPs and their environment. Such interaction

1 Introduction

Traditionally, control theory has dealt with the dynamic behavior of processes whose variables are numerical and whose evolution can be modeled by differential or difference equation. With the widening use of computers as essential components of systems, increasingly complex systems have emerged that can no longer be adequately described by conventional models. Indeed, in an increasing number of processes states may have not just numerical values, but *symbolic* or logical values as well. State changes may then occur in response to the occurrence of discrete events that take place at discrete times, frequently asynchronously and nondeterministically. The control of such systems is of great practical importance and theoretical interest, and poses a wide range of new and intriguing intellectual challenges.

The simplest processes that exhibit such *discrete* behavior are *discrete event processes*, or DEPs. These are processes whose behavior can be modeled entirely within a state-event framework, that is, processes whose states are discrete and state changes take place only in response to *events* that occur at discrete and irregular intervals. Some of the more common and familiar examples of such processes are computer operating systems, manufacturing systems, communication networks, traffic systems, resource (such as power or water) management systems, and computer-based supervisory control systems of complex plants.

A state transition and its associated event constitutes the basic fragment of a DEP. (Finite) state machines and their associated state transition diagrams are the simplest formal mechanism for collecting such fragments into a whole. State machine models are conceptually appealing because of their inherent simplicitly and the fact that they can be described adequately by finite automata and the theory of regular languages.

Recently, Ramadge and Wonham [45, 49, 50] initiated a pioneering effort of developing a control theory of DEPs within the framework of state machines and formal languages. In their framework all events are spontaneous and process-generated. Some of the events, called *controllable* events, possess a disablement mechanism accessible to the environment, and the control problem is to suitably interact with the process, by disabling of controllable evnts, so as to confine its behavior to within specified *legal* bounds. The mechanism examined in the work of Ramadge and Wonham for such interaction is called *feedback control* and consists of certain mappings between the process under consideration and a suitably formulated *supervisor*. Process behavior is modeled by its *language*, i.e., the set of event-strings that the process can generate. Various control-theoretic questions such as controllability [45, 49], observability [28, 42, 37], decentralized and hierarchical control [29, 51, 38] and stabilition [7, 36], as well as such questions as computational complexity [43, 44] and others were

Part I

Concurrency and Discrete Event Control

Abstract

In recent years there has been a growing interest in the control of discrete-event processes. These are processes in which state changes take place in response to events that occur discretely, asynchronously and often nondeterministically. Much of the theory that was developed to date was inspired by the pioneering work of Ramadge and Wonham [45, 49, 50], where a discrete- event control theory was presented within the framework of automata and formal languages. In the present paper an alternate approach is proposed for discrete-event modeling and control. This approach, inspired by the theories of Process-Algebra as developed in the computer science literature (e.g. [20, 34, 5, 35]), rests on a framework of concurrency. Accordingly, a new concurrency formalism is introduced that is suitable for modeling a wide range of interactions between discrete-event processes and, in particular, is suitable for dealing with various formalisms of discrete-event control. The new framework can adequately handle nondeterminism and can be used for analysis of a wide range of discrete-event phenomena. It is particularly effective for synthesis of discrete-event controllers from local specifications. In the present paper the approach of Process-Algebra is briefly reviewed and the new formalism of concurrency is introduced. The paper is tutorial and is primarily intended to appeal to the reader's intuition. A more detailed and formal account can be found in part 2 of this report.

List of Figures

Contents

An Algebra of Discrete Event Processes

Michael Heymann[*] and George Meyer[†]

October 1990

Abstract

This report deals with an algebraic framework for modeling and control of discrete event processes. The report consists of two parts. The first part is introductory, and consists of a tutorial survey of the theory of concurrency in the spirit of Hoare's CSP, and an examination of the suitability of such an algebraic framework for dealing with various aspects of discrete event control. To this end a new concurrency operator is introduces and it is shown how the resulting framework can be applied. It is further shown that a suitable theory that deals with the new concurrency operator must be developed. In the second part of the report the formal algebra of discrete event control is developed. At the present time the second part of the report is still an incomplete and ocassionally tentative working paper.

[*]NRC Senior Research Associate, NASA-Ames Research Center, Moffett Field, CA. 94035, on leave from the Department of Computer Science, Technion, Israel Institute of Technology, Haifa 32000, Israel.

[†]NASA-Ames Research Center Moffett-Field, CA. 94035