Self-Synchronization of Concurrent Processes

Lalita Jategaonkar
Albert R. Meyer

July 1993

In *Proceedings of the 8th IEEE Symposium on Logic in Computer Science*,pages 409-417, July 1993.

Self-Synchronization of Concurrent Processes (Preliminary Report)

(Appeared in the Proceedings of the Symposium on Logic in Computer Science, pages 409–417, 1993)

Lalita Jategaonkar *
MIT Lab. for Computer Science
Cambridge, MA 02139
lalita@theory.lcs.mit.edu

Albert R. Meyer†
MIT Lab. for Computer Science
Cambridge, MA 02139
meyer@theory.lcs.mit.edu

Abstract

We introduce a unary "self-synchronization" operation on concurrent processes that synchronizes concurrent transitions within *a process. Standard parallel synchronization and communicating action refinement operations can be reduced to simple combinations of self-synchronization and unsynchronized non-communicating operations. We show that modifying familiar fully abstract process semantics so that actions are replaced by action multi-sets (steps) typically yields semantics which are fully abstract for processes with self-synchronization.*

1 Introduction

1.1 Self-Synchronization

We introduce a unary "self-synchronization" operation on concurrent processes similar in spirit to the binary operations of parallel-composition-with-communication found in CCS [12], CSP [9] and ACP [2]. Self-synchronization, however, allows sets of concurrent transitions *within* a process to synchronize and perform a new transition.

Self-synchronization on action set $\{a, b\}$ and action c of process P, written $\mathrm{SS}_{\langle a,b\rangle\rightarrow c}(P)$, will be a new process which acts like P, except that whenever P has a pair of concurrent transitions with labels a and b, then $\mathrm{SS}_{\langle a,b\rangle\rightarrow c}(P)$ has an additional transition, labeled c, leading to the same state reachable by concurrently performing the a and b transitions.

In the presence of self-synchronization and the usual sequential operations of process algebra, a variety of synchronized parallel composition operations are expressible by pure, unsynchronized parallel composition. For example, the TCSP-style parallel-composition-with-synchronization operator $||_{\{a\}}$, which requires processes to synchronize with one another on the action a, can be defined by self-synchronization and unsynchronized parallel composition, $||$. Namely, for all processes P_1 and P_2, the following identity holds in our theory:

$$P_1 \,||_{\{a\}}\, P_2 = (\mathrm{SS}_{\langle a_1,a_2\rangle\rightarrow a}(P_1[a{::=}a_1] \,||\, P_2[a{::=}a_2]))\backslash a_1, a_2.$$

Here, $\backslash a_1$ is the operation restricting a_1-transitions from firing, $[a{::=}a_1]$ is the renaming operation, and a_1 and a_2 are "fresh" actions. This equation straightforwardly generalizes to express the TCSP operator $||_L$, which requires a pair of processes to synchronize on a set L of actions.

A self-synchronization operation tailored to *complementary actions* $a, \overline{a}$ is necessary to define the CCS-style synchronized parallel composition $P \mid Q$. We define the "synchronize complements" operator, SC, to allow pairs of concurrent transitions with complementary labels to synchronize and perform the hidden action τ. That is, if the alphabet of process P is $\{a, \overline{a}, b, \overline{b}\}$, we have:

$$\mathrm{SC}(P) =_{\mathrm{def}} \mathrm{SS}_{\langle b,\overline{b}\rangle\rightarrow\tau}(\mathrm{SS}_{\langle a,\overline{a}\rangle\rightarrow\tau}(P)).$$

For all processes P_1 and P_2, the following identity then holds in our theory:

$$\mathrm{SC}(P_1) \mid \mathrm{SC}(P_2) = \mathrm{SC}(P_1 \,||\, P_2).$$

In a CCS-style setting, processes are generally already self-synchronized on complementary actions, that is,

*Supported by the AT&T GRPW Fellowship and ONR grant No. N00014–83–K–0125.
†Supported by ONR grant No. N00014–83–K–0125.

they satisfy the identity $\mathrm{SC}(Q) = Q$. For example, the CCS-like processes considered by Aceto&Hennessy in [1] and Hennessy in [8] are all self-synchronized in this way. For such processes, we then have

$$Q_1 \mid Q_2 = \mathrm{SC}(Q_1 \mid\mid Q_2).$$

Similar identities hold for SCCS-style [11] and Meije-style [3] synchronized parallel composition.

The intuitive simplicity of self-synchronization is reflected in various algebraic identities such as

$$\mathrm{SS}_{\langle a,b\rangle\to c}(P+Q) = \mathrm{SS}_{\langle a,b\rangle\to c}(P) + \mathrm{SS}_{\langle a,b\rangle\to c}(Q)$$
$$\mathrm{SS}_{\langle a,b\rangle\to c}(a.P) = a.(\mathrm{SS}_{\langle a,b\rangle\to c}(P))$$
$$\mathrm{SS}_{\langle a,b\rangle\to c}(P\backslash a) = P\backslash a$$

Here $a.()$ is the prefixing operator and $+$ is the CCS choice operator.

1.2 Self-Synchronization and Action Refinement

We have found self-synchronization particularly helpful in clarifying the relationship between contrasting notions of "action refinement" proposed in the literature.

Aceto&Hennessy [1] and Hennessy [8] have developed a CCS-like process language with an action refinement operator based on syntactic substitution, in which communication can occur between refining processes. Hennessy [8] suggests that such an action-refinement-with-communication may be more useful than the non-communicating version defined by Vogler [22] and van Glabbeek&Goltz [18] and generalized by the present authors in [10].

An action refinement operator is specified by a function ρ from actions to processes. We use ρ as a postfix operator denoting application of the *non*-communicating action refinement operator determined by ρ [10, 18, 22]. Aceto&Hennessy require ρ to respect complementary labels in a suitable sense [1]; we refer to such action refinements as "complement-respecting" refinements. We use $[\rho]_{comm}$ as a postfix operator denoting application of the *communicating* action refinement operator determined by complement-respecting ρ [1].

From [1], we have

$$(P \mid Q)[\rho]_{comm} =_{\mathrm{def}} P[\rho]_{comm} \mid Q[\rho]_{comm}.$$

In contrast, our non-communicating action refinement operator [10] distributes over non-communicating parallel composition:

$$(P \mid\mid Q)\rho = P\rho \mid\mid Q\rho.$$

(Aceto&Hennessy did not specify the behavior of their action refinement operator over non-communicating parallel composition. Indeed, since non-communicating parallel composition does not preserve self-synchronization under complementary actions, it appears to be incompatible with the theory of communicating action refinement they aim to develop.)

Our self-synchronization operator and our earlier non-communicating action refinement operator together can define a Aceto&Hennessy-style communicating action refinement operator. Namely, for any process term P in Aceto&Hennessy's CCS-like language of [1] and any complement-respecting action refinement ρ, the following identity holds in our theory:

$$P[\rho]_{comm} = \mathrm{SC}(P\rho).$$

In fact, we use this as a defining identity for a communicating action refinement which nicely generalizes to net-processes Aceto&Hennessy's refinement operation on process terms.

Aceto&Hennessy's action refinement operator has essentially the same expressive power as our non-communicating action refinement operator together with *two-way hidden self-synchronization*, $\mathrm{SS}_{\langle a,b\rangle\to\tau}$, which allows pairs of concurrent, possibly non-complementary actions a, b to synchronize and perform a τ action. The precise identities are fairly simple, and are omitted here.

1.3 Self-Synchronization and Steps

Self-synchronization makes concurrent actions visible to ordinary sequential observers. For example, self-synchronization visibly distinguishes the process $(a \mid\mid b)$ — which can perform the actions a and b concurrently — from the process $(ab+ba)$, which cannot. In particular, the process $\mathrm{SS}_{\langle a,b\rangle\to c}(a \mid\mid b)$ can perform the action c while the process $\mathrm{SS}_{\langle a,b\rangle\to c}(ab+ba)$ cannot. Thus, self-synchronization invalidates the interleaving axiom, namely, in the presence of the self-synchronization operator,

$$a \mid\mid b \neq ab + ba.$$

In general, self-synchronization has the power to make *steps*, *i.e.*, finite, non-empty multi-sets of concurrent visible actions, detectable to sequential observers. For any step σ and any action γ, we write $\mathrm{SS}_{\sigma\to\gamma}$ to denote the self-synchronization operator that allows each set of concurrent transitions whose multi-set of labels is σ, to synchronize and perform a

γ. Then, if γ is a fresh action (not in the alphabet of P), we have:

$$\mathrm{SS}_{\sigma\to\gamma}(P) \xrightarrow{\gamma} \mathrm{SS}_{\sigma\to\gamma}(P') \text{ iff } P \xrightarrow{\sigma} P'.$$

Thus, formalizing self-synchronization requires some process model which reflects concurrency, as opposed to the state-transition models of CCS, CSP, *etc.*. Such "true concurrency" models include Petri Nets, Mazurkiewicz nets of communicating automata [15], and event structures [4, 14, 24]. In this paper, we have chosen to use a safe Petri Net model of processes, but the choice is not critical.

1.4 Overview of Results

The failure of the interleaving axiom also shows that the standard interleaving equivalences, such as traces, failures [5], and bisimulation [12], are *not compositional* for self-synchronization. However, we suggest that this is typically quite simple to fix. Our thesis is that the natural "step-versions" of most semantics gotten by simply replacing actions by steps will typically be just right — in the sense of *full abstraction* — with respect to self-synchronization.

Definition A semantics assigning a meaning to every process is *compositional* for an operator on processes if semantic equality is a congruence for the operator, *i.e.*, the operator preserves semantic equality. We say that a semantics is *adequate* for an equivalence on processes if semantic equality implies process equivalence. Finally, we say that a semantics is *fully abstract* for a process equivalence *with respect to* a set of operators if semantic equality is the coarsest congruence for those operators that is adequate for the process equivalence.

Thesis For typical semantics arising in process algebra and net theory, there is a corresponding "step-version" of the semantics that is fully abstract for the original semantical equivalence with respect to self-synchronization.

As illustrations of the thesis, we show:

- Step-trace semantics, previously considered in [17], step-failure semantics [16, 17], and step-observational-equivalence [17, 20] are respectively fully abstract for MAY-equivalence [7], MUST-equivalence [7], and observational equivalence [12] with respect to self-synchronization and the usual CCS/CSP operators. We note that although these step-semantics may be considered to be among the most basic true concurrency semantics, full abstraction results were previously unknown.
- Let $\mathcal{IPF}$ be the interval pomset-failure semantics of [10] (*cf.* [22]); $\mathcal{IPF}$ is fully abstract for MUST-equivalence on nets with respect to non-communicating action refinement and the CCS/CSP operators. Then a natural "2-step-version" of $\mathcal{IPF}$ is fully abstract for MUST-equivalence when we include the operation of two-way hidden self-synchronization.

Using the equalities highlighted earlier in this section, we also show that the "2-step-version" of the $\mathcal{IPF}$ semantics is fully abstract with respect to MUST-equivalence for the CCS-like language with general *communicating* action refinement proposed by Aceto&Hennessy in [1]. (We note that Hennessy [8] proves full abstraction for the original $\mathcal{IPF}$ semantics[1] by suitably restricting action refinement. For example, he forbids an action refinement like $[a{:=}a+c.d, b{:=}b+\overline{c}.d]$, where a and $\overline{b}$ are distinct actions. These restrictions are necessary since otherwise $\mathcal{IPF}$ semantics is not compositional.)

These and similar results are discussed further in Sections 3–5. A summary of new full abstraction results supporting our Thesis appears in Table 1.

2 Self Synchronization in a Net Model of Concurrent Processes

Our model of concurrent processes consists of possibly infinite, initially marked, safe Petri Nets whose transitions have labels from some set $Act = \mathcal{L} \cup \{\tau\}$, where τ is a *hidden* action, and $\mathcal{L}$ is a set of *visible* actions. When convenient, we implicitly assume $\mathcal{L}$ is closed under action complement as in CCS [12]. In order to define sequencing and action refinement, we extend our nets to signal successful termination, as in [10]. A number of other technical conditions that our nets satisfy are discussed in detail in [10]. We have net operators corresponding to the usual CCS/CSP operators; the definitions of these operators are well-known (*cf.* [6, 10, 22]) and are omitted here.

We define the *places* of a transition t of a net N to be the places directly connected to it, *i.e.*, the union

[1] In fact, Hennessy [8] shows that a ST-failures-style semantics is fully abstract. Vogler [23] explains the correspondence between $\mathcal{IPF}$-style semantics and ST-failures-style semantics, from which it is easy to show that the $\mathcal{IPF}$ semantics of [10] and Hennessy's semantics of [8] define the same equivalence.

		PROCESS EQUIVALENCE		
		MAY	MUST	*Observational*
OPERATORS	Self-synchronization ($SS_{\sigma \to a}$)	Step-traces	Step-failures	Step-obs-equiv
	Hidden self-synchronization ($SS_{\sigma \to \tau}$)	Step-traces	Step-failures	?
	2-way hidden self-synchronization ($SS_{\langle a,b \rangle \to \tau}$)	2-step-traces	2-step-failures	?
	Self-synchronization ($SS_{\sigma \to a}$) and non-communicating action refinement (ρ)	Interval pomset-traces	Interval pomset-step-failures	ST-obs-equiv
	Hidden self-synchronization ($SS_{\sigma \to \tau}$) and non-communicating action refinement (ρ)	Interval pomset-traces	Interval pomset-step-failures	ST-obs-equiv
	2-way hidden self-synchronization ($SS_{\langle a,b \rangle \to \tau}$) and non-communicating action refinement (ρ)	Interval pomset-traces	Interval pomset-2-step-failures	ST-obs-equiv
	Communicating action refinement ($[\rho]_{comm}$)	Interval pomset-traces	Interval pomset-2-step-failures	ST-obs-equiv

Table 1: Fully Abstract Semantics*

of the preset and postset of t. We say that transitions t_1 and t_2 of a net N are *statically concurrent* in N iff the places of t_1 are disjoint from those of t_2. We define a *step* to be any finite, non-empty multi-set of visible actions. For any step σ and any action α, the self-synchronization operator $SS_{\sigma \to \alpha}$ applied to a net N introduces a new α-labeled transition for every set of pairwise statically concurrent transitions of N whose corresponding multi-set of labels is σ. The preset of each new transition is the union of the presets of the corresponding set of transitions; likewise for the postset.

Some illustrations of self-synchronization are given in Figures 1– 4. A simple case of self-synchronization is illustrated in Figure 1, where one a-labeled transition is statically concurrent with the b-labeled transition, while the other a-labeled transition is not. Figure 2 illustrates that there may be ambiguity in which sets of concurrent transitions synchronize. Figure 3 illustrates that some statically concurrent transitions may not be *dynamically concurrent*, that is, they may never fire concurrently. Hence it is straightforward to see that the new c-labeled transition in Figure 3 will never fire. Finally, Figure 4 illustrates that self-synchronization in general is defined over *multi-sets* of actions.

We use the standard definition by which a net determines a transition system on its finite safe markings (*cf.* [10]), and we thus can apply to nets the standard definitions of ordinary (interleaving) trace equivalence, MAY-equivalence [7], ordinary (interleaving) failure equivalence [5], MUST-equivalence [7], and observational equivalence [12] based on labeled transition systems.

3 The Simplest Case: Step Traces

To illustrate our full abstraction results, we sketch the simplest case, namely, step-traces. We first generalize the usual definitions of runs (*i.e.* firing sequences of transitions) to *step-runs* as follows. We define a *step-transition* to be any finite set of transitions which are pairwise statically concurrent; moreover, we define the *visible-multiset* of a step-transition to be the multi-set of corresponding labels, restricted to visible actions. We say that a step-transition is *enabled* under a marking iff each transition in the step-transition is enabled under that marking; furthermore, the marking (necessarily unique) reached after firing the step-transition is defined to be the marking reached after firing the transitions in the step-transition in any sequential order. A *step-run* is then the obvious generalization of firing sequences of transitions to firing sequences of step-transitions, and a *step-trace* of a step-run is the sequence of steps which are the visible-multisets of the step-transitions, omitting empty multi-sets. The set of *step-traces* of a net is the set of step-traces of its step-runs. This step-trace semantics is essentially the same as [17].

We have:

Theorem 3.1 Step-trace semantics is fully abstract for MAY-equivalence on nets with respect to self-synchronization and the CCS/CSP operators.

Proof. An ordinary (interleaving) trace of a net corresponds in the obvious manner to a step-trace of

*Fully abstract with respect to the indicated operators *together with* all the usual CCS/CSP operators.

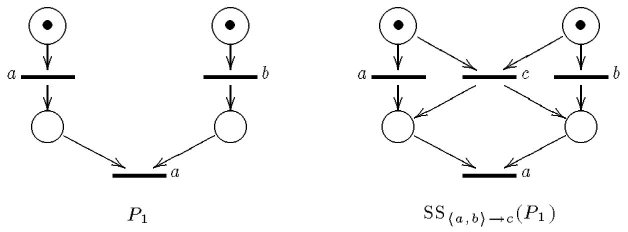


Figure 1: Simple Case of Self-Synchronization

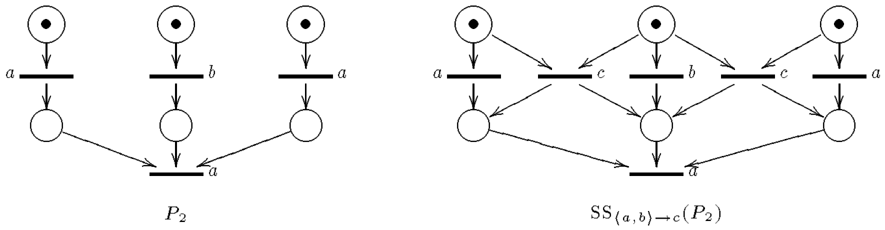


Figure 2: Self-Synchronization Introduces Ambiguity

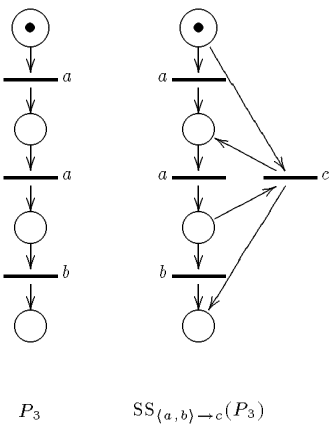
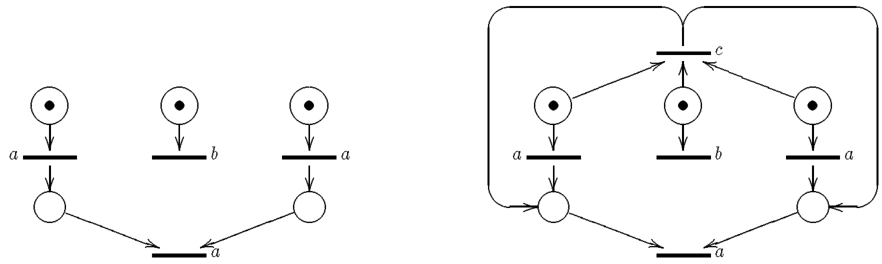


Figure 3: Static Concurrency versus Dynamic Concurrency



P_4 $\qquad$ $\mathrm{SS}_{\langle a,a,b\rangle \to c}(P_4)$

Figure 4: Self-Synchronization over Multi-Sets of Actions

the net consisting only of singleton steps, from which adequacy follows trivially. The compositionality of step-traces for self-synchronization and the CCS/CSP operators is straightforward. Finally, full abstraction follows from the fact that self-synchronization transforms steps into actions, as discussed in the Introduction. ■

We note that step-trace equivalence is already fully abstract for MAY-equivalence on nets solely with respect to the self-synchronization operator.

4 Further Steps

A *step-failures* semantics corresponding to step-traces with associated step-failure sets can also be defined easily. We omit the details from this preliminary report. This step-failure semantics is essentially the same as that defined by Taubner&Vogler [16]. (Note that since we allow the CCS choice operator, +, we also incorporate the usual modification of CSP failures to achieve compositionality with respect to +.) While Taubner&Vogler [16] had given a compositional definition of step-failure semantics, they had not considered the full abstraction problem: in fact, *ordinary* (interleaving) failure semantics is fully abstract for MUST-equivalence with respect to the set of operators they consider. Using the same sort of reasoning as in the previous section, we obtain a full abstraction result:

Theorem 4.1 Step-failure semantics is fully abstract for MUST-equivalence on nets with respect to self-synchronization and the CCS/CSP operators.

We remark that step-failure semantics is already fully abstract for MUST-equivalence on nets solely with respect to self-synchronization and CCS choice.

Further supporting our Thesis, a similar full abstraction property holds for *step-observational-equivalence*, modified to be compositional for CCS choice in the standard way (*cf.* [12]) discussed above. We refer to this modified equivalence as *step-observational-equivalence.* This semantics is essentially the same as the step-bisimulations defined in [17, 20].

Theorem 4.2 Step-observational-equivalence is fully abstract for observational equivalence on nets with respect to self-synchronization and the CCS/CSP operators.

We note that step-observational-equivalence is already fully abstract for observational equivalence on nets solely with respect to self-synchronization and CCS choice.

An interesting case that arises concerns pomset-failure semantics [10, 22], which are a mix of true concurrency and interleaving information. Namely, they consist of pomsets, which implicitly provide information about steps, together with failure sets of actions, which do not. These semantics are not compositional for self-synchronization. However, our Thesis suggests the remedy: generalize the failure sets to contain steps. The resulting interval pomset-*step*-failure semantics is then fully abstract:

Theorem 4.3 Interval pomset-*step*-failure semantics is fully abstract for MUST-equivalence on nets with respect to self-synchronization, non-communicating action refinement, communicating action refinement, and the CCS/CSP operators.

We remark that interval pomset-step-failure semantics is already fully abstract for MUST-equivalence on nets solely with respect to self-synchronization, CCS choice, and two very simple forms of non-communicating action refinement — split refinements and choice refinements. Split and choice refinements are discussed in [10].

Another interesting case highlighted by Aceto and Hennessy is the setting consisting only of communicating action refinement and the CCS operators. As discussed in the Introduction, communicating action refinement has the same expressive power as non-communicating action refinement together with *two-way* hidden self-synchronization; thus, we consider the interval pomset-step-failure semantics with failure sets restricted to steps of size at most 2. The resulting semantics, which we refer to as interval pomset-*2-step*-failure semantics, has the following property:

Theorem 4.4 Interval pomset-*2-step*-failure semantics is fully abstract for MUST-equivalence on nets with respect to two-way hidden self-synchronization, non-communicating action refinement, communicating action refinement, and the CCS/CSP operators.

In fact, the interval pomset-2-step-failure semantics is fully abstract for MUST-equivalence solely with respect to 2-way self-synchronization, CCS choice, non-communicating split refinements and non-communicating choice refinements. It is also fully abstract for CCS choice and *communicating* action refinement, from which we conclude:

Corollary 4.5 Interval pomset-2-step-failure semantics is fully abstract for MUST-equivalence on Aceto&Hennessy's processes with respect to the operations of Aceto&Hennessy's process algebra with syntactic general communicating action refinement [1].

5 Steps Already Taken

Certain true concurrency semantics already embody complete information about steps. Such semantics are equivalent to their "step-versions" and are already compositional for self-synchronization. An example is pomset-trace semantics [10, 13, 17, 22]:

Lemma 5.1 General pomset-trace equivalence and interval pomset-trace equivalence on nets are compositional for self-synchronization.

Interval pomset-trace semantics on nets has been shown to be fully abstract for MAY-equivalence with respect to non-communicating action refinement and the CCS/CSP operators [10]. As a corollary of full abstraction and Lemma 5.1, we have:

Theorem 5.2 Interval pomset-trace semantics is fully abstract for MAY-equivalence on nets with respect to self-synchronization, non-communicating action refinement, communicating action refinement, and the CCS/CSP operators.

We remark that interval pomset-trace semantics is already fully abstract for MAY-equivalence on nets solely with respect to non-communicating split refinements and non-communicating choice refinements.

We next show that similar results hold for "ST-observational-equivalence" [17, 19, 20]:

Lemma 5.3 ST-observational-equivalence on nets is compositional for self-synchronization.

Vogler [21] has proved that ST-observational equivalence on prime event structures is fully abstract for observational equivalence with respect to conflict-free, non-communicating action refinement. A straightforward adaptation of this proof shows that ST-observational-equivalence on nets is fully abstract for observational equivalence with respect to general non-communicating action refinement. This property together with Lemma 5.3 yields:

Theorem 5.4 ST-observational-equivalence is fully abstract for observational equivalence on nets with respect to self-synchronization, non-communicating action refinement, communicating action refinement, and the CCS/CSP operators.

We remark that ST-observational equivalence is already fully abstract for observational equivalence on nets solely with respect to CCS choice, non-communicating split refinements, and non-communicating choice refinements.

6 Conclusions and Open Problems

We have argued that self-synchronization is a natural operation under which other operations such as parallel synchronization and communicating action refinement can be understood more clearly. We have suggested that simply by replacing actions with steps, existing semantics can be refined to yield fully abstract semantics with respect to self-synchronization. We have described a series of results that support this Thesis.

The Thesis, of course, is offered as a guide rather than a theorem. Its soundness becomes less sure in technically restricted situations involving, for example, hidden self-synchronization and CCS-style processes. An instance where the Thesis fails is the setting of two-way hidden self-synchronization and the CCS operators, *not* including CSP parallel composition. Here the 2-step-trace and 2-step-failure semantics are adequate and compositional, but *not* fully abstract. In fact, the fully abstract semantics are quite *ad hoc* and will be discussed elsewhere.

We remark that all our semantics are, in fact, fully abstract for MAY- or MUST-*approximation*, where MAY-semantics are partially ordered by set-theoretic containment, and MUST-semantics are partially ordered by component-wise reverse containment. As usual, the conjunction of MAY and MUST semantical equality corresponds to full Testing congruence [7].

It remains to show that our theory supports recursive definitions of nets. To this end, we have still to characterize the semantic domains, show that they are cpo's, and verify that our operators on nets correspond to continuous operations, a task we expect can be completed routinely.

7 Acknowledgments

We are grateful to Matthew Hennessy for several very helpful discussions. We are also thankful to Rob van Glabbeek and Frits Vaandrager for helpful suggestions and information about prior results.

References

[1] L. Aceto and M. Hennessy. Adding action refinement to a finite process algebra. In *Proceedings of ICALP '91, Volume 510 of Lecture Notes in Computer Science*, pages 506–519, 1991.

[2] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theor. Comput. Sci.*, 37(1):77–121, May 1985.

[3] G. Boudol. Notes on algebraic calculi of processes. Technical report, INRIA, Sophia-Antipolis, 1985.

[4] G. Boudol and I. Castellani. Permutations of transitions: An event structure semantics for CCS and SCCS. In *Linear Time, Branching Time, and Partial Order in Logics and Models of Concurrency, Volume 354 of Lecture Notes in Computer Science*, pages 411–427, 1989.

[5] S. D. Brookes and A. W. Roscoe. An improved failures model for communicating processes. In *Seminar on Concurrency, Volume 197 of Lecture Notes in Computer Science*, pages 281–305, 1984.

[6] U. Goltz. CCS and Petri nets. Technical report, GMD, July 1990.

[7] M. C. Hennessy. *Algebraic Theory of Processes*. Series on Foundations of Computing. MIT Press, 1988. 272 pp.

[8] M. C. Hennessy. Concurrent testing of processes. In *Proceedings of CONCUR '92, Volume 630 of Lecture Notes in Computer Science*, pages 94–107, 1992.

[9] C. A. R. Hoare. *Communicating Sequential Processes*. Series in Computer Science. Prentice-Hall, Inc., 1985. 256 pp.

[10] L. Jategaonkar and A. R. Meyer. Testing equivalence for Petri nets with action refinement. In *Proceedings of CONCUR '92, Volume 630 of the Lecture Notes in Computer Science*, pages 17–31, 1992.

[11] R. Milner. Calculi for synchrony and asynchrony. *Theor. Comput. Sci.*, 25(3):267–310, July 1983.

[12] R. Milner. *Communication and Concurrency*. Series in Computer Science. Prentice-Hall, Inc., 1989.

[13] M. Nielsen, U. Engberg, and K. S. Larsen. Fully abstract models for a process language with refinement. In *Linear Time, Branching Time, and Partial Order in Logics and Models of Concurrency, Volume 354 of Lecture Notes in Computer Science*, pages 523–548, 1988.

[14] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures, and domains I. *Theor. Comput. Sci.*, 13:85–108, 1981.

[15] A. Rabinovich and B. Trakhtenbrot. Behavior structures and nets of processes. *Fundamenta Informaticae*, 11(4):357–404, 1988.

[16] D. Taubner and W. Vogler. Step failures semantics and a complete proof system. *Acta Inf.*, 27(2):125–156, Nov. 1989.

[17] R. van Glabbeek and U. Goltz. Equivalence notions for concurrent systems and refinement of actions. In *Proceedings of MFCS '89, Volume 379 of the Lecture Series in Computer Science*, pages 237–248, 1989.

[18] R. van Glabbeek and U. Goltz. Refinement of actions in causality based models. In *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness, Proceedings of REX Workshop '89, Volume 430 of Lecture Notes in Computer Science*, pages 267–300, 1989.

[19] R. van Glabbeek and F. Vaandrager. Petri net models for algebraic theories of concurrency. In *Proceedings of PARLE '87, Volume 259 of the Lecture Notes in Computer Science*, pages 224–242, 1987.

[20] W. Vogler. Bisimulation and action refinement. In *Proceedings of STACS '91, Volume 480 of the Lecture Notes in Computer Science*, pages 309–321, 1991.

[21] W. Vogler. Bisimulation and action refinement. Technical report, Technische Universität München, 1991.

[22] W. Vogler. Failures semantics based on interval semiwords is a congruence for refinement. *Distributed Computing*, 4:139–162, 1991.

[23] W. Vogler. Is partial order semantics necessary for action refinement? Technical report, Technische Universität München, 1991.

[24] G. Winskel. Event structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency, Volume 255 of Lecture Notes in Computer Science*, pages 325–392, 1987.

Cambridge, MA
April 7th, 1993