Testing Equivalence for Petri Nets with Action Refinement

Lalita Jategaonkar
Albert R. Meyer

August 1992

In *Proceedings of the 3rd International Conference on Concurrency Theory*, Volume 630 of the Lecture Notes in Computer Science, pages 17-31, August 1992.

Testing Equivalence for Petri Nets with Action Refinement: Preliminary Report

(Appeared in the Proceedings of International Conference on Concurrency Theory, LNCS Volume 630, pages 17–31, 1992)

Lalita Jategaonkar* and Albert Meyer**

MIT Laboratory for Computer Science
Cambridge, MA 02139

Abstract. A definition of "action refinement" is given for an operational model of concurrent processes based on safe Petri Nets, generalizing previous work of Vogler and van Glabbeek/Goltz. A failure-style denotational semantics is described for process nets. The semantics is fully abstract for Hennessy Testing-equivalence on nets acting as refinement *operators* as well as operands. The semantics embodies the notions of deadlock, failures and divergences found in the Hoare/CSP and Hennessy Testing-equivalence theories, as well as some of the basic ideas of "pomset runs" and "causal" partial orders of Net theory.

1 Introduction

The operation of refining atomic actions in a concurrent process suggests aspects of top-down "modular" development and also requires use of some sort of "true" concurrency process model [1, 2, 3, 5, 8, 12, 13, 14, 17]. In a previous paper [10], we developed a semantics, $[\![\cdot]\!]^{\mathrm{MUST}}$, for certain simple "splitting" and "choice" action refinements on a Petri Net model of processes. Our semantics generalizes and simplifies a similar semantics developed in a seminal paper by Vogler [17]. These semantics support full process theories involving parallel process communication, deadlock, failures, hidden actions, and divergences (*cf.* [4, 7, 9, 11]). Their essential component consists of pomsets paired with failure sets. We observed that Vogler's semantics can equivalently be described as the restriction, $[\![\cdot]\!]^{\mathrm{MUST}}_{\mathrm{intvl}}$, of our general pomset semantics to interval pomsets.

Definition 1. A semantics, $[\![\cdot]\!]$, assigning to any process, N, a meaning, $[\![N]\!]$, 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 the semantics is adequate for the equivalence and semantic equality is the coarsest congruence for those operators.

* Supported by the AT&T GRPW Fellowship, NSF Grant No. 8511190–DCR and ONR grant No. N00014–83–K–0125.
** Supported by NSF Grant No. 8511190–DCR and ONR grant No. N00014–83–K–0125.

Both our and Vogler's semantics are adequate for MUST-equivalence [7] of nets and are compositional for splitting and choice refinements as well as for net operations corresponding to the familiar CCS/CSP operations. It follows from [17] that $[\![\cdot]\!]^{\text{MUST}}_{\text{intvl}}$ is, in fact, *fully abstract* for MUST-equivalence with respect to splitting and choice refinements.

Vogler generalizes splitting and choice refinements to allow a large class of refinement operators corresponding to a class of "refinement nets" required to satisfy some rather technical structural and behavioral conditions [17]. Both $[\![\cdot]\!]^{\text{MUST}}$ and $[\![\cdot]\!]^{\text{MUST}}_{\text{intvl}}$ semantics are compositional with respect to each of the operators corresponding to Vogler's refinement nets. Namely, if two nets are equivalent under these semantics, then applying the *same* action refinement ρ to each of them yields semantically equivalent nets.

However, it is *not* the case that these semantics are compositional for nets as action refinement *operators*. For example, the nets a and $\tau.a$, where τ is the hidden action, are semantically equivalent as operands or *targets* of action refinement, but they behave differently when used as operators refining an action b, *viz.*,

$$
\begin{array}{lrcl}
 & [\![a]\!] & = & [\![\tau.a]\!]\,, \\
\text{but} & [\![(b+c)[b{:=}a]]\!] & \neq & [\![(b+c)[b{:=}\tau.a]]\!].
\end{array}
$$

In this paper, we resolve this problem by establishing that a surprisingly simple variation of the earlier semantics yields MUST-adequate semantics that are compositional for nets as *targets and operators* of action refinement, with the modified $[\![\cdot]\!]^{\text{MUST}}_{\text{intvl}}$ semantics being fully abstract. We similarly show how to handle MAY-equivalence; the MAY- and MUST-semantics together provide a fully abstract semantics for Testing Equivalence [7].

We begin by presenting in Section 2 our general class of *Well Terminating (WT) Nets*. These are possibly infinite, safe nets with designated transitions for signaling successful termination. We then give a definition of action refinement that allows *any* WT net to be used as a refinement operator. Our class of nets and refinements generalizes those of Vogler and van Glabbeek/Goltz [14], since both their target and refinement nets, and indeed arbitrary safe nets, can be understood as special cases of WT nets.

In addition to action refinement, we indicate how to define net operators corresponding to familiar CCS/CSP operations of prefixing ($a.$), restriction ($\backslash a$), hiding ($-a$), renaming, CSP-style sequencing (;), CSP-style parallel-composition-with-synchronization ($\|_L$), CCS-style parallel-composition-with-hiding ($|$), internal choice, and CCS-style choice ($+$).

Section 3 illustrates in some more detail the difficulties with compositionality encountered by our earlier semantics [10], and describes our modification to the semantics that repairs these problems. A discussion of other results, related work and future work is given in Section 4. Finally, in order to keep this paper self-contained, the Appendix provides the definitions of our earlier semantics [10].

2 Nets and Operations

Our class of "Well-Terminating" Nets is related to the class of CSP processes that signal successful termination by performing a distinguished action, $\surd$. In a similar manner, our well-terminating nets signal successful termination by firing any transition labeled with $\surd$. In order to ensure that the net has actually terminated, we require that all places in the net be unmarked after any $\surd$-labeled transition fires.

We wish to restrict our attention to nets with "computable behavior," and we thus impose some syntactic conditions that guarantee finite-markings and finite-branching.

Definition 2. The class of *Well-Terminating (WT) Nets* is the class of labeled, safe, possibly infinite Petri nets that satisfy the following properties:

- The initial marking is finite.
- All places have finite out-degree.
- All transitions have finite in-degree and finite out-degree.
- All places of the net are unmarked immediately after any $\surd$-labeled transition fires. This condition must be satisfied in every reachable marking of the net.

We note that the first three conditions together imply that all reachable markings are finite and that only a finite number of transitions are enabled under any reachable marking. In particular, our nets have only finite concurrency.

The condition on the $\surd$-transitions ensures that no transition (not even a $\surd$-transition) can be fired concurrently with, or following, a $\surd$-transition.

We assume for expository simplicity that all transitions have non-empty presets, and that the initial marking is non-empty. Formally, we write a WT net N as a triple $\langle S_N, T_N, Start_N \rangle$, where S_N is the set of places, T_N is the set of transitions, and $Start_N$ is the (finite) set of initially marked places. Furthermore, for every transition $t \in T_N$, we write $l_N(t)$ to refer to its label, and $pre_N(t)$ and $post_N(t)$ to refer to its preset and post-set. Similarly, for every place $s \in S_N$, we write $pre_N(s)$ and $post_N(s)$ to refer to its preset and post-set. Our syntactic conditions on nets imply that all places have finite postsets and that all transitions have finite presets and finite postsets.

We can understand the target nets of [14, 17] to be a special case of WT nets having no $\surd$-labeled transitions. The connection between WT nets and the refinement nets of [14, 17] is slightly more subtle, since the latter signal successful termination by marking some designated "accept" places rather than by firing some designated transitions. However, these nets can be easily understood as WT nets by adding a $\surd$-transition whose preset is essentially this set of "accept" places. (For technical reasons, some of the initially marked places of their refinement nets must also feed into this $\surd$-transition; we omit the details here.)

Our $\surd$-labeled transitions serve to distinguish deadlock from successful termination. We say that a net *successfully terminates* when a $\surd$-labeled transition fires, while a net is *deadlocked* exactly when no transition is enabled. We write *Succ* to denote the WT net which must immediately successfully terminate, *i.e.*, exactly one transition is enabled under its initial marking and this transition is $\surd$-labeled. For notational convenience, we simply write a to refer to the net $a.Succ$. Furthermore, we write *Dead* to denote the deadlocked process, and $a.Dead$ to refer to the net that does an a and then deadlocks.

The $\surd$-action plays a distinguished role in our theory, and we forbid prefixing with $\surd$ and renaming of other actions to $\surd$. We also forbid refinement of $\surd$-labeled transitions.

Our sequencing operator $N_1; N_2$ makes critical use of the $\surd$-transitions of N_1 by relabeling them with τ and using them as a hidden (τ-labeled) signal to transfer control to N_2. We illustrate the definition of "sequencing" through the following simple example. Suppose that we are given the WT nets $N_1 = a_1 + a_2$ and $N_2 = b_1 \|_\emptyset b_2$ of Figure 1, and we want to define $N_1; N_2$. We want the firing of either of the $\surd$-transitions of N_1 to be a hidden signal that enables both b_1 and b_2 to fire concurrently. Therefore, we relabel the $\surd$-transitions of N_1 to τ, and then have both of these τ-transitions feed into both of the start places of N_2. The resulting net $N_1; N_2$ is given in Figure 1.

We also illustrate the definition of $+$ for N_1 and N_2 of Figure 1. Clearly, we want to introduce conflicts between the a_i and the b_j but preserve the concurrency within the b_j, and so we do a simple cross product construction on the start places of both nets. We note that this causes all the $\surd$-labeled transitions to be in conflict, as desired. The resulting net is also given in Figure 1. As discussed in [15], one technical complication arises due to initially marked places that have incoming transitions, and in general, we apply a *start-unwinding* operator on nets [15] before doing the above construction.

We also have a CSP-style parallel composition operator $\|_L$ on WT nets, where two nets are placed in parallel, but must synchronize on all actions in the set $L \cup \{\surd\}$, where L is a set of visible labels. Our definition is essentially the same as [17] and is omitted here. Similar to [6], we also have a CCS-style parallel composition operator $|$, where two nets are placed in parallel and are allowed to perform hidden synchronizations on all complementary actions; again, they must (visibly) synchronize on the $\surd$ action. The net operators for prefixing, restriction, hiding, renaming, and internal choice are straightforward, and are omitted. The class of WT nets is closed under all of these operations.

Except for the parallel composition operators, all of our net operations are closely related to the corresponding CCS/CSP operators on labeled transition systems (*lts*'s). In particular,

Lemma 3. *For our CCS/CSP WT net-operators other than $\|_L$ and $|$, the lts of the constructed net is strongly bisimilar to the lts obtained by applying the corresponding CCS/CSP lts-operator to the lts's of the component nets. Also, lts$(N_1 \|_L N_2)$ is strongly bisimilar to lts$(N_1) \|_{L \cup \{\surd\}}$ lts(N_2).*

The relationship between our WT net operator, $|$, and the corresponding CCS *lts* operator is slightly more complicated, but similar.

Two simple WT net operators play a significant role in our technical development. Namely, *split refinements* ($split_{(a,a_1,a_2)}$) replace every a-labeled by two consecutive transitions labeled a_1 and a_2, and *choice refinements* ($choice_{(a,a_1,a_2)}$) replace every a-labeled transition by two conflicting transitions labeled a^L and a^R. Figure 2 gives examples of these kinds of refinements.

The major new WT net operator we develop is *action refinement.* Our action refinement operator has a rich algebraic theory. For example, the following simple

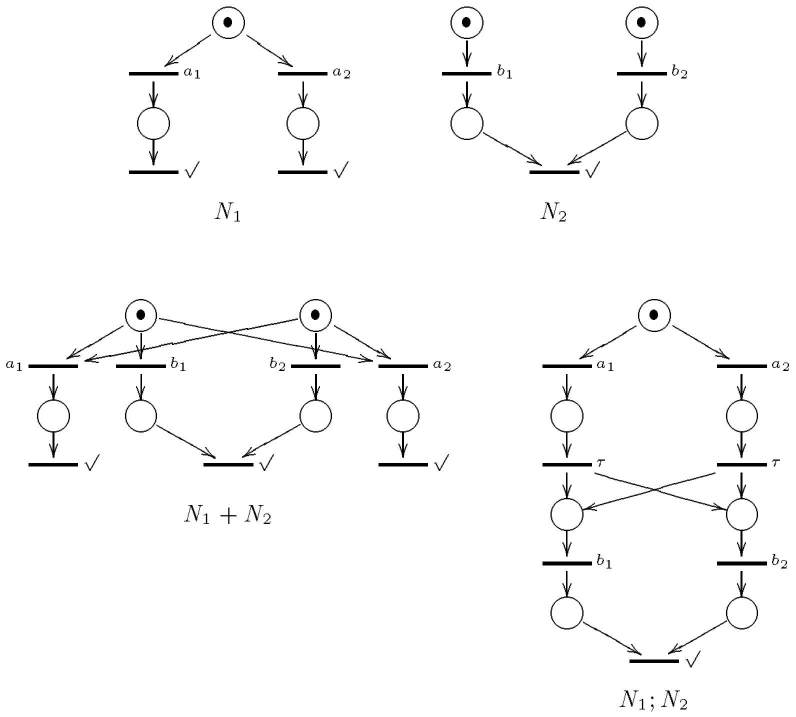

Fig. 1. Examples of Choice and Sequencing

identities hold up to semantical equality:

$$
\begin{aligned}
a[a{:=}N] &= N \\
N[a{:=}a] &= N \\
N[a{:=}Dead] &= N \backslash a \\
N[a{:=}\tau] &= N[a{:=}Succ] = N - a \\
split_{(a,a_1,a_2)}(N) &= N[a{:=}a_1.a_2] \\
choice_{(a,a^L,a^R)}(N) &= N[a{:=}a^L + a^R]
\end{aligned}
$$

Assuming that a and b are "fresh" labels, we also have:

$$
\begin{aligned}
((a+b)[a{:=}N_1])[b{:=}N_2] &= N_1 + N_2 \\
((a.b)[a{:=}N_1])[b{:=}N_2] &= N_1; N_2 \\
((a\|_\emptyset b)[a{:=}N_1])[b{:=}N_2] &= N_1\|_\emptyset N_2
\end{aligned}
$$

For all refinements ρ, the following distributivity properties hold:

$$
\begin{aligned}
(N_1 + N_2)\rho &= N_1\rho + N_2\rho \\
(N_1; N_2)\rho &= N_1\rho; N_2\rho \\
(N_1\|_\emptyset N_2)\rho &= N_1\rho\|_\emptyset N_2\rho
\end{aligned}
$$

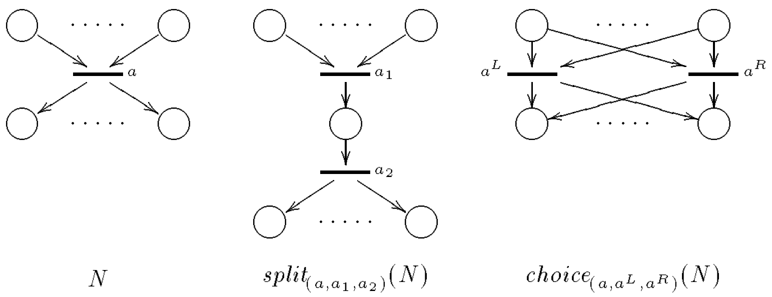

Fig. 2. Split Refinements and Choice Refinements

Our action refinement operator $N[a{:=}A]$ "replaces" each a-labeled transition in N by a separate but identical copy of A; these copies are distinguished by "tagging" the names of the places and transitions of A with the name of the corresponding a-labeled transition. Since we want our action refinement operator to satisfy the distributivity properties mentioned above, we need to be careful in how we hook up the copies of A to the places of N.

In the same spirit as the definition of the $+$ operator, we take cross products of the start places of appropriate copies of A; in particular, for every place s in N, we take a cross product v of the start places of the copies of A corresponding to the a-labeled transitions emanating from s. Furthermore, in the same spirit as the definition of sequencing, we relabel with τ all of the $\surd$-labeled transitions of the copies of A and connect them all up to the post-set of the corresponding a-labeled transition. The other transitions of the copies of A and the non-a-labeled transitions of N are then hooked up to all of these places in the expected manner.

Not surprisingly, we encounter the same difficulties as the $+$ operator when our refinement nets have initially marked places that have incoming transitions, and we thus start-unwind the refinement net before performing our replacements.

We now define the action refinement operator. For simplicity we assume that the refinement net A is already start-unwound; otherwise, we first start-unwind A and then carry out this construction using the start-unwound version of A rather than A itself.

Definition 4. Let N and A be WT Nets, and let a be a label other than $\surd$. Then $P = N[a{:=}A]$ is defined as:

$$S_P = \{(s,v) \mid s \in S_N \text{ and } v\colon T \rightarrow Start_A,\ \text{where } T = \{t \in post_N(s) \mid l_N(t) = a\}\ \} \\ \cup\ \ \{(t,s') \mid t \in T_N,\ l_N(t) = a \text{ and } s' \in S_A - Start_A\}$$

$$T_P = \{(t,*) \mid t \in T_N \text{ and } l_N(t) \neq a\} \cup \{(t,t') \mid t \in T_N,\ l_N(t) = a \text{ and } t' \in T_A\}$$

$$pre_P((t,*)) = \{(s,v) \in S_P \mid s \in pre_N(t)\}$$

$$post_P((t,*)) = \{(s,v) \in S_P \mid s \in post_N(t)\}$$

$$l_P((t,*)) = l_N(t)$$

$$pre_P((t,t')) = \{(s,v) \in S_P \mid s \in pre_N(t) \text{ and } v(t) \in pre_A(t')\} \\ \cup \quad \{(t,s') \in S_P \mid s' \in pre_A(t')\}$$

$$post_P((t,t')) = \begin{cases} \{(t,s') \in S_P \mid s' \in post_A(t')\} & \text{if } l_A(t') \neq \surd \\ \{(s,v) \in S_P \mid s \in post_N(t)\} & \text{otherwise} \end{cases}$$

$$l_P((t,t')) = \begin{cases} l_A(t') & \text{if } l_A(t') \neq \surd \\ \tau & \text{otherwise} \end{cases}$$

$$Start_P = \{(s,v) \in S_P \mid s \in Start_N\}$$

Our definition of refinement generalizes the definitions of refinement given by Vogler and van Glabbeek/Goltz in the sense that our refined net is semantically equivalent to their nets. In fact, there is an even tighter relationship between them, namely, these nets are equivalent up to a weaker form of history-preserving bisimulation [13] which treats τ-moves as hidden and respects concurrent divergences. We omit the definition here since it is not necessary in the development below.[3]

We note that our definition of action refinement preserves finiteness of nets, and thus, in the same spirit as our full class of WT nets, we can allow arbitrary finite WT nets to function as both target nets and refinement nets. The class of finite WT nets is also closed under all of the CCS/CSP operations discussed previously.

3 The Semantics

We presume that the reader is familiar with the theory of MUST-equivalence on labeled transition systems developed in [9, 7]. The definition of MUST-equivalence carries over directly to WT nets: two WT nets will be said to be MUST-equivalent iff their labeled transition systems are MUST-equivalent under the standard definition.

As mentioned in the introduction, all of the semantics of [10] (*cf.*, the Appendix) are compositional for WT nets as *targets* of action refinement. Namely, if two nets are semantically equivalent, then applying the *same* action refinement ρ to both of them yields semantically equivalent nets.

Theorem 5. *$[\![\cdot]\!]^{\text{MUST}}$ and $[\![\cdot]\!]^{\text{MUST}}_{\text{intvl}}$ on WT nets are adequate for* MUST*-equivalence and compositional for split refinements, choice refinements, and our CCS/CSP operators; likewise for $[\![\cdot]\!]^{\text{MAY}}$ and $[\![\cdot]\!]^{\text{MAY}}_{\text{intvl}}$. Furthermore, all four of these semantics are compositional for WT nets as* targets *of action refinement.*

[3] Since Vogler and van Glabbeek/Goltz use a cross-product construction on the "accept" places of their refinement nets rather than using τ-moves to transfer control back to the target net, our refined net is not quite *strongly* history-preserving bisimilar to their nets.

However, as explained in the introduction, neither $[\![\cdot]\!]^{\text{MUST}}$ nor $[\![\cdot]\!]^{\text{MUST}}_{\text{intvl}}$ are compositional as *operators* of action refinement. The problem with both semantics is improper accounting of "initial" τ-moves. For example, as indicated in the introduction, $[\![a]\!]^{\text{MUST}} = [\![\tau.a]\!]^{\text{MUST}}$, but the net $(b+c)[b{:=}a]$ must offer a c, while $(b+c)[b{:=}\tau.a]$ would not offer a c after a hidden move. Hence, these refined nets are MUST-distinguishable and so have different meanings.

Rather surprisingly, this is the *only* problem with the semantics, even for WT nets that can diverge. In contrast to WT nets, another problem arises with the semantics when applied to Vogler's refinement nets; namely, for these nets, the semantics does not properly detect successful termination. In particular, although Vogler's refinement nets signal successful termination by marking some designated "accept" places, the semantics is not tuned to detect whether these accept places are marked.

For example, $[\![a]\!]^{\text{MUST}} = [\![a.\mathit{Dead}]\!]^{\text{MUST}}$, where a is the Vogler refinement net that fires an a-transition and then successfully terminates (by marking its accept places); however, $(a;b)$ is trivially MUST-distinguishable from $((a.\mathit{Dead});b)$. The net $a.\mathit{Dead}$ would be disallowed as a refinement net by [17], but the problem occurs even for non-deadlocking refinement nets. For example, let $N_3 = a + ab + a(b+c)$ and $N_4 = a + a(b+c)$. Then N_3 and N_4 are refinement nets in the sense of [17], and $[\![N_3]\!]^{\text{MUST}} = [\![N_4]\!]^{\text{MUST}}$. However, $(N_3;c)$ and $(N_4;c)$ are MUST-distinguishable, since $(N_3;c)$ can do an a—the "middle" one—and then refuse to do the action c, while $(N_4;c)$ is ready to do the action c after doing any a.

However, since WT nets signal successful termination by firing $\surd$-transitions (which are visible), the $[\![\cdot]\!]^{\text{MUST}}$ semantics applied to WT nets *does* properly detect successful termination. Thus, the semantics of WT nets as refinement operators can be captured by simply +'ing the net with a fresh, distinguished action γ, and taking the semantics of the resulting net. In fact, the MAY-semantics $[\![\cdot]\!]^{\text{MAY}}$ and $[\![\cdot]\!]^{\text{MAY}}_{\text{intvl}}$ of [10] are already compositional *for WT nets.*

Definition 6. For a WT net N,

$$
\begin{aligned}
[\![N]\!]^{\text{MUST}}_{\gamma} &=_{\text{def}} [\![\gamma + N]\!]^{\text{MUST}} \\
[\![N]\!]^{\text{MUST}}_{\text{intvl-}\gamma} &=_{\text{def}} [\![\gamma + N]\!]^{\text{MUST}}_{\text{intvl}}
\end{aligned}
$$

Theorem 7. *$[\![\cdot]\!]^{\text{MUST}}_{\gamma}$ and $[\![\cdot]\!]^{\text{MUST}}_{\text{intvl-}\gamma}$ on WT nets are adequate for* MUST-*equivalence and compositional for split refinements, choice refinements, and our CCS/CSP operators; likewise for $[\![\cdot]\!]^{\text{MAY}}$ and $[\![\cdot]\!]^{\text{MAY}}_{\text{intvl}}$. Furthermore, all four of these semantics are compositional for WT nets as targets* and operators *of action refinement.*

Using the information provided by the γ transition as well as that provided by the $\surd$-transitions of the original net, it is fairly straightforward to prove these results for the $[\![\cdot]\!]^{\text{MUST}}_{\gamma}$ semantics and for both of the MAY-semantics. However, the fact that every interval pomset-divergence contains information about only a *single* divergence makes it much more difficult to reason compositionally about the $[\![\cdot]\!]^{\text{MUST}}_{\text{intvl-}\gamma}$ semantics when *concurrent* transitions are refined with nets that can *diverge.* Since our semantics "blurs" all information that "extends" a pomset-divergence, we have to be careful that combining such "blurred" information about these divergent nets does not somehow make finer distinctions based on concurrent divergences (*cf.* [10]).

As promised, the full abstraction properties hold of the interval semantics.

Theorem 8. *The* $[\![\cdot]\!]^{\text{MUST}}_{\text{intvl-}\gamma}$ *semantics on WT nets is fully abstract for* MUST*-equivalence with respect to the set of operators consisting of split refinements, choice refinements, and +; likewise for the* $[\![\cdot]\!]^{\text{MAY}}_{\text{intvl}}$ *except that + is not needed.*

As in [12, 17], the proof of full abstraction is based on the well-known result that every interval ordering is order-isomorphic to a set of intervals of the real line with (interval x) $<$ (interval y) iff every point in x is less than every point in y. Since split refinements and choice refinements allow us to associate unique beginnings and endings with all of the transition firings, we can use certain *sequential* observations of these beginnings and endings to fully determine the ordering of the pomset-failures and pomset-divergences in our $[\![\cdot]\!]^{\text{MUST}}_{\text{intvl-}\gamma}$ semantics. Using the + operator appropriately, we can then show that all of the semantical distinctions can be detected by *sequential* experiments.

These ideas are crystallized in the following lemma. For the sake of simplicity, we assume here that our action alphabet is finite; however, our results easily extend to infinite alphabets in the same manner as [17].

Lemma 9. *Let* σ *be a sequence of split refinements, one for each action in the alphabet, and for all* $k \geq 1$*, let* ρ_k *be a sequence of choice refinements mapping each action* a *in the alphabet to the net* $a^1 + \ldots + a^k$*. If* N_1 *and* N_2 *are WT nets with* $[\![N_1]\!]^{\text{MUST}}_{\text{intvl-}\gamma} \neq [\![N_2]\!]^{\text{MUST}}_{\text{intvl-}\gamma}$*, then there is some integer* k *bounded by the maximum concurrency of* N_1 *and* N_2 *(which may be infinite), such that the net* $(\gamma + N_1\rho_k\sigma)$ *is* MUST*-distinct from the net* $(\gamma + N_2\rho_k\sigma)$*.*

An analogous result holds for the $[\![\cdot]\!]^{\text{MAY}}_{\text{intvl}}$ *semantics.*

We now restrict our attention to *finite* WT nets. Since all finite WT nets have bounded concurrency, we can use Theorem 7 and Lemma 9 to reduce the decidability of $[\![\cdot]\!]^{\text{MUST}}_{\text{intvl-}\gamma}$-equivalence and $[\![\cdot]\!]^{\text{MAY}}_{\text{intvl}}$-equivalence for finite WT nets to the decidability of MUST-equivalence and MAY-equivalence for finite safe nets. As illustrated by Vogler [16], there are simple automata-theoretic arguments that reduce these latter problems to the equality of regular languages, which is known to be decidable. We thus have:

Corollary 10. *For finite WT nets,* $[\![\cdot]\!]^{\text{MUST}}_{\text{intvl-}\gamma}$*-equivalence and* $[\![\cdot]\!]^{\text{MAY}}_{\text{intvl}}$*-equivalence are decidable.*

We remark that Vogler [18] obtained this result for the $[\![\cdot]\!]^{\text{MUST}}_{\text{intvl}}$ semantics by using a ST-representation of interval pomsets, which he argued was more convenient for the purpose than pomsets. Our reduction of interval pomset-failures to ordinary failures via Lemma 9 indicates that this alternative representation need not be introduced.

4 Other Results, Related Work, and Future Work

All of 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 is easy to show that our decidability results extend to testing-approximation.

We expect that all of our semantical spaces form continuous partial orders, and that our action refinement and CCS/CSP operators on nets correspond to continuous semantical operations. Consequently, we expect that our theory will routinely support *arbitrary* (not merely guarded) *recursive* definitions of nets, with recursion understood as usual via least fixed points.

An important direction for further research is development of the algebra of process terms with refinement. One immediate problem to consider is finding a complete axiom system for equations between closed recursion-free CSP/CCS process terms—corresponding to the (non-divergent) isolated elements in our semantical spaces.

There is not yet a consensus on what an action refinement operator should be. For example, the action refinement operator of [3, 8, 14, 17] contrasts with the one used in [1, 2], since the operators of [3, 8, 14, 17] distribute over CCS-choice but not over a Hoare/Hennessy external choice operator, $+_H$, while the operators of [1, 2] distribute over $+_H$ but not over CCS-choice. While our $[\![\cdot]\!]^{\mathrm{MUST}}_{\mathrm{intvl}}$ semantics is compositional for all our CSP operations, including $+_H$, it is not compositional for CCS-choice, and therefore not compositional for nets as action refinement operators. However, we can define a modified action refinement operator that is tuned to $+_H$ and for which $[\![\cdot]\!]^{\mathrm{MUST}}_{\mathrm{intvl}}$ is fully abstract. We believe this $[\![\cdot]\!]^{\mathrm{MUST}}_{\mathrm{intvl}}$ semantics subsumes that of Aceto/Engberg [1], who give a fully abstract failure semantics for action refinement in a restricted framework without parallel synchronization, a restriction operator, or divergence.

A more significant contrast in approaches to action refinement is that our action refinement operator and that of [17] are tuned to a CSP-style synchronization-with-restriction, while those of [3, 8] are tuned to a CCS-style synchronization-by-hiding-complementary-actions. In this regard, an action-refinement theory closely related to ours has been proposed by Hennessy [8]. His theory incorporates an interesting, and in certain respects more powerful, action refinement operation, and he has compositionality and full abstraction results similar to ours. Unlike our action refinement operation, Hennessy's definition allows "concurrent" refinement nets to "communicate" with one another in a manner closely related to CCS-style parallel composition, where concurrent, complementary actions (*i.e.*, a and $\bar{a}$) can synchronize and perform a hidden move. However, in order for Hennessy's semantics to remain compositional for this powerful sort of action refinement, this inter-communication must in fact be quite restricted: in particular, "initial" hidden communications between refinement nets must be disallowed. As a result, Hennessy forbids some simple action refinements like $(a \mid b)[a{:=}c,\ b{:=}\bar{c}]$. The connection between Hennessy's and our theories of action refinement will be the topic of a paper now in progress.

Acknowledgments

We are grateful to Rob van Glabbeek, Ursula Goltz, Matthew Hennessy, Wolfgang Reisig, Boris Trakhtenbrot, Frits Vaandrager, Walter Vogler, and David Wald for helpful discussions. We thank Roberto Segala for proofreading previous versions of this paper.

References

1. L. Aceto and U. Engberg. Failure semantics for a simple process language with refinement. Technical report, INRIA, Sophia-Antipolis, 1991.
2. L. Aceto and M. Hennessy. Towards action-refinement in process algebras. In *Proceedings of 4^{th} LICS*, pages 138–145. IEEE Computer Society Press, 1989.
3. L. Aceto and M. Hennessy. Adding action refinement to a finite process algebra. In *Proceedings of 18^{th} ICALP*, volume 510 of *Lecture Notes in Computer Science*. Springer-Verlag, 1991.
4. 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. Springer-Verlag, 1984.
5. L. Castellano, G. De Michelis, and L. Pomello. Concurrency vs. interleaving: an instructive example. *Bull. Europ. Assoc. Theoretical Computer Sci.*, 31:12–15, 1987.
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 3^{rd} CONCUR*, 1992. Appears in this volume.
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 split and choice refinements. Paper presented at the Eighth Workshop on the Mathematical Foundations of Programming Semantics, Oxford, England, Apr. 1992.
11. R. Milner. *Communication and Concurrency.* Series in Computer Science. Prentice-Hall, Inc., 1989.
12. 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 for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 523–548. Springer-Verlag, 1988.
13. R. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions.* PhD thesis, CWI, 1990.
14. R. van Glabbeek and U. Goltz. Refinement of actions in causality based models. In *Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness*, volume 430 of *Lecture Notes in Computer Science*, pages 267–300. Springer-Verlag, 1990.
15. R. van Glabbeek and F. Vaandrager. Petri net models for algebraic theories of concurrency. In *Proceedings of PARLE Conference*, volume 259 of *Lecture Notes in Computer Science*, pages 224–242. Springer-Verlag, 1987.
16. W. Vogler. Failure semantics and deadlocking of modular petri nets. *Acta Informatica*, 26(4):333–348, 1989.
17. W. Vogler. Failures semantics based on interval semiwords is a congruence for refinement. *Distributed Computing*, 4:139–162, 1991.
18. W. Vogler. Is partial order semantics necessary for action refinement? Technical report, Technische Universitat Munchen, 1991.

A Appendix

In order to keep this paper self-contained, this appendix provides the definitions of our semantics [10] for safe nets. The main idea is that we first simultaneously "split" every *visible* transition t into two consecutive transitions labeled a_1 and a_2, where a is the label of t. We leave all τ-labeled transitions unsplit. We then straightforwardly extract the "pomset-failures" and "pomset-divergences" of the split net, perform some closure operations, and then restrict to interval pomsets.

We begin with the standard notions of pomsets and pomset runs of safe nets:

Definition 11. A *pomset* is a labeled partial order. The *pomset runs* of a net N are pomsets whose elements, called "events," are occurrences of transitions of N, labeled with the labels of the corresponding transitions, and partially ordered by the usual "causal" ordering defined on firings of transitions of N [15], *cf.* Figure 3.

Since pomset runs of nets may contain τ-labeled events which are unobservable, we define an operation, *visible*, on pomset runs which keeps only the visible events of the pomset run.

Definition 12. Let q be a pomset run of a net N. Then $visible(q)$ is the restriction of q to its events with visible labels. The *pomset-traces* of N are the set of $visible(q)$ such that q is a finite pomset run of N, *cf.* Figure 3.

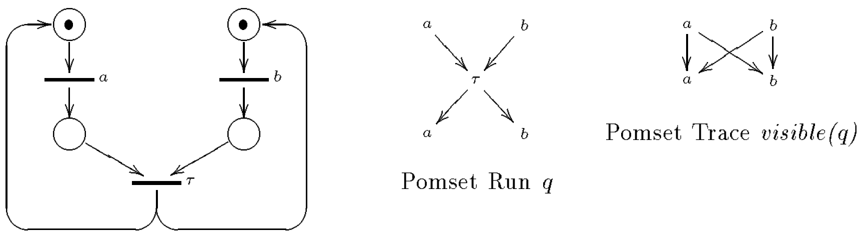

Fig. 3. An Example of a Pomset Run and Pomset Trace

Our definition of "pomset-failures" is a natural generalization of (sequential) "failures" in that it associates "failure sets" to finite pomsets. In particular, a *pomset-failure* is a pair $\langle p, F\rangle$, where p is a finite pomset with visible labels, and F is a set of visible labels.

It is well known that there is a uniquely determined final marking associated with each finite pomset run of a net; this is the marking reached after sequentially firing the events of the run in any order that is consistent with its partial order. Now, for any marking which is not divergent, we define a "failure set" of that marking as any set of *visible* actions that are not enabled under that final marking, even after further firing any finite sequence of τ-labeled transitions. (Note that this is exactly the standard definition of "failure sets" on the labeled transition system whose states are the reachable markings of the net and whose labeled transitions correspond to firings of single net transitions.)

Definition 13. The *pomset-failures* of a net N are the pairs $\langle visible(q), F\rangle$ such that q is a finite pomset run of N and F is a failure set of the marking after q.

We also wish to define a notion of "pomset-divergences" that is a natural generalization of (sequential) "divergences."

Definition 14. The *downward closure*, $\widehat{e}$, of an element e in a pomset p is $\{e' \in p \mid e' \leq_p e\}$. The downward closure, $\widehat{E}$, of a subset E of (the carrier of) a pomset p is $\bigcup\{\widehat{e} \mid e \in E\}$; E is *downward closed* iff $E = \widehat{E}$. A *pomset-divergence* is a pair $\langle p, \mathcal{D}\rangle$, where p is a finite pomset and $\mathcal{D}$ is a non-empty set of downward closed subsets of (the carrier of) p.

Given any pomset run with only a finite number of visible events, it is easy to see that any infinite chain of τ-labeled-events indicates a divergence of the net. We wish to define pomset-divergences of nets in such a way that we keep track of all the *concurrent divergences* within a pomset run while abstracting away from the τ-labeled events.

Definition 15. Let q be an infinite pomset run of a net N with a finite number of visible events. Let $\mathcal{D}$ be the family of sets of the form (carrier of) $visible(\widehat{C})$ such that C is an infinite chain of τ-labeled events of q. Then $\langle visible(q), \mathcal{D}\rangle$ is a *pomset-divergence* of N, *cf.* Figure 4.

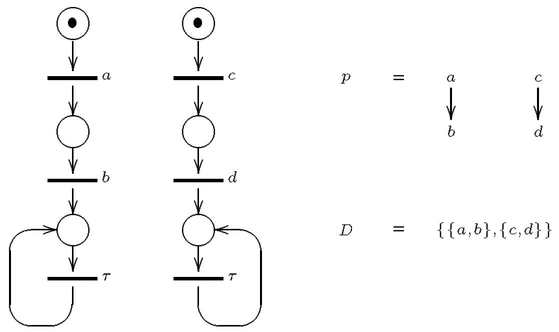

Fig. 4. An Example of a Pomset Divergence $\langle p, D\rangle$

It turns out that the semantics defined by simply taking these pomset-failures and pomset-divergences makes too many distinctions between nets, and we need to "blur" certain kinds of information from our runs. This we accomplish through various closure operations. The first such closure involves taking "augmentations" of our pomset-failures and pomset-divergences. We first restate the standard definition for pomsets, where an augmentation is simply an increase in the partial ordering.

Definition 16. Pomset p' is an *augmentation* of pomset p iff p and p' have the same set of elements with the same labels, and the partial ordering of p' contains the partial ordering of p. Let $augment(p)$ be the set of augmentations of p. The augmentations, $augment(\langle p, F\rangle)$, of a pomset-failure $\langle p, F\rangle$ is the set $\{\langle p', F\rangle \mid p' \in augment(p)\}$. For pomset-divergences, let $augment(\langle p, \mathcal{D}\rangle)$ be

$$\begin{aligned}\{\langle p', \mathcal{D}'\rangle \mid\ & p' \in augment(p) \\ & \text{and } \mathcal{D}' = \{d' \mid d' \text{ is the downward closure } in\ p' \text{ of some } d \in \mathcal{D}\}\}\end{aligned}$$

Our other closure operation arises from the fact that MUST-experiments fail to yield information about the behavior of a net after a divergence. To get around this difficulty, we define below the notion of an *extension* of a pomset-divergence; the idea is that the extension is another pomset-divergence which may contain more information concerning events and divergences which "happen after" one or more divergences in the original pomset-divergence. All the information about a process after a pomset-divergence is blurred by throwing in all possible pomset-failures and pomset-divergences which extend the original pomset-divergence.

Definition 17. Pomset-divergence $\langle p', \mathcal{D}'\rangle$ *extends* pomset-divergence $\langle p, \mathcal{D}\rangle$ iff

p is the restriction of p' to a downward-closed subset of p';
for all $e \in p' - p$, there is some $d \in \mathcal{D}$ with $d \subseteq \widehat{e}$; and
for all $d' \in \mathcal{D}'$, there is some $d \in \mathcal{D}$ with $d \subseteq d'$.

Let $extend(\langle p, \mathcal{D}\rangle)$ be the set of pomset-divergences which extend it. Finally, let

$$implied\text{-}failures(\langle p, \mathcal{D}\rangle) =_{\rm def} \{\langle p, F\rangle \mid F \text{ is a set of visible labels}\}.$$

We lift these operations on individual pomsets, failures, *etc.*, to sets of individuals by pointwise union. For example,

$$augment(X) =_{\rm def} \bigcup_{x \in X} augment(x).$$

We are now ready to define the MAY- and MUST-semantics. For the MAY-semantics, we can work with the pomsets of the original net. On the other hand, as explained earlier, for the MUST-semantics we need to work with the pomsets of the "split" net.

Definition 18. For any safe net N,

$$\begin{aligned}
[\![N]\!]^{\rm MAY} &=_{\rm def} augment(pomset\text{-}traces(N)),\\
Div(N) &=_{\rm def} extend(augment(pomset\text{-}divergences(split(N)))),\\
Fail(N) &=_{\rm def} augment(pomset\text{-}failures(split(N))) \cup implied\text{-}failures(Div(N)),\\
[\![N]\!]^{\rm MUST} &=_{\rm def} \langle Fail(N), Div(N)\rangle.
\end{aligned}$$

Our definition of semantical equality implicitly equates label-preserving order-isomorphic pomsets. For finitely-marked safe nets with finite out-degree of places and transitions, the semantics satisfy the following properties:

Theorem 19. *$[\![\cdot]\!]^{\rm MUST}$ is adequate for MUST-equivalence and compositional for split refinements, choice refinements, and our CCS/CSP operators; likewise for $[\![\cdot]\!]^{\rm MAY}$.*

However, it turns out that $[\![\cdot]\!]^{\text{MUST}}$ and $[\![\cdot]\!]^{\text{MAY}}$ make more distinctions than are apparent to a single experimenter. Namely, single experimenters can only detect differences between pomsets with *interval orderings* [12, 17]. (A partial order $\leq$ is considered to be an interval ordering iff whenever both $w < x$ and $y < z$, then either $w < z$ or $y < x$.)

We define the interval-MUST- and interval-MAY-semantics by restricting the general semantics to interval pomsets:

Definition 20. Let $\mathcal{P}$ be a set of pomsets, let $\mathcal{PF}$ be a set of pomset-failures, and let $\mathcal{PD}$ be a set of pomset-divergences. Then *intervals*($\mathcal{P}$) is the set of interval pomsets $p \in \mathcal{P}$. Let *intervals*($\mathcal{PF}$) be the set of $\langle p, F\rangle \in \mathcal{PF}$ such that p is an interval pomset, and *intervals*($\mathcal{PD}$) be the set of $\langle p, \{p\}\rangle \in \mathcal{PD}$ such that p is an interval pomset. Let

$$\begin{aligned} [\![N]\!]^{\text{MUST}}_{\text{intvl}} &=_{\text{def}} \langle \mathit{intervals}(\mathit{Div}(N)), \mathit{intervals}(\mathit{Fail}(N))\rangle \\ [\![N]\!]^{\text{MAY}}_{\text{intvl}} &=_{\text{def}} \mathit{intervals}([\![N]\!]^{\text{MAY}}) \end{aligned}$$

We now have:

Theorem 21. *The* $[\![\cdot]\!]^{\text{MUST}}_{\text{intvl}}$ *semantics is fully abstract for* MUST*-equivalence with respect to split refinements and choice refinements; likewise for* $[\![\cdot]\!]^{\text{MAY}}_{\text{intvl}}$*. Furthermore, both are compositional for our CCS/CSP operators.*