The Semantics of `Triveni`: A Process-Algebraic API for Threads + Events

Christopher Colby* Lalita Jategaonkar Jagadeesan† Radha Jagadeesan
Konstantin Läufer Carlos Puchol

May 17, 1998

1 Introduction

This paper describes compositional semantics (operational, denotational and logical) for a process algebra enhanced with input/output actions and preemption combinators, in the presence of fairness. The context of this paper is `Triveni`, a process-algebra-based design methodology that combines threads and events in the context of object-oriented programming [CJJ$^+$98a, CJJ$^+$98b]. `Triveni` has been realized as an API, `JavaTriveni` [CJJ$^+$98a], in the Java programming language. A case study in `JavaTriveni` is described in [CJJ$^+$98b], involving the re-implementation of a piece of telecommunication software – the *Carrier Group Alarms (CGA)* software of Lucent Technologies' 5ESS switch.

The semantics described in this paper is closely tied to the `JavaTriveni` programming language and environment. In particular, the design and implementation of `JavaTriveni` motivate the combination of features described in our semantics and the criteria placed on our semantic study.

- The operational model described in this paper is the precise formalization of the `JavaTriveni` implementation.
- The denotational semantics serves as the basis for a non-definability result. This result justifies the introduction of certain powerful preemption combinators as primitives in `JavaTriveni`.
- The logical semantics forms the basis of our specification-based testing environment for `JavaTriveni`.

The underlying process algebra and communication model. `Triveni` is based on a process algebra that adds preemption combinators [Ber93] to the standard combinators from process algebra such as parallel composition, waiting for events, hiding events, and so forth. The Java event model is based on the Observer pattern[1] [GHJV95]; hence, compatibility requirements with event models based on this pattern dictate certain aspects of the process algebra.

Criterion 1.1 The communication model is *multicast*; the processes are *input-enabled* [LT89, Dil88, Jos92] and output actions cannot block [Vaa91].

*Loyola University Chicago: {colby,radha,laufer}@math.luc.edu
†Lucent Technologies, Bell Labs: {lalita,cpg}@research.bell-labs.com

[1] In the Observer pattern events are generated by event sources (subjects), and one or more listeners (observers) can register with a source to be notified about events of a particular kind.

In addition, `Triveni` is compatible with existing threads standards (e.g. Pthreads, Java threads). In particular, `Triveni` allows existing threads (in the host language) that conform to an Observer-pattern-based interface to be used as subcomponents. This forces the following "openness" requirement on the semantic model.

Criterion 1.2 The semantic model must identify a general class of processes that can be used as primitives in the process algebra. This description must provide a criterion that can be checked on the behavior of processes and must thus be independent of the syntax of processes.

Fairness in `Triveni`. Fairness is necessary to provide useful techniques for reasoning about preemption combinators in the context of threads; in particular, fairness ensures some liveness properties which would not hold otherwise (liveness enhancing [AFK88]). For example, consider the program

$$(\ \mathtt{DO}\ p\ \mathtt{WATCH}\ e?\ \mathtt{TOUT}\ q)\ ||\ \mathtt{EMIT}\ e!$$

in which the left hand process is executing p until the occurrence of event e, after which p is aborted and q is started. The right hand process simply emits the $e!$ event. Fairness among parallel processes is necessary to guarantee that the right hand side will eventually get a chance to emit the $e!$ event, and hence that the left hand side will eventually abort its execution of p and begin execution of q. As in any programming language, compositional reasoning is vital.

Criterion 1.3 The semantic model must support a compositional treatment of fairness.

1.1 Our results

The concerns of input/output actions, preemption, and fairness form the basis of our semantic study. We describe operational, denotational and logical semantics and show correspondence theorems relating the three semantics. Our results hold for both strong fairness (a process that is able to execute some output event occurrence infinitely often is given a chance to execute that output event occurrence infinitely often) and weak fairness (a process that is continually able to execute some output event occurrence is given a chance to execute that output event occurrence infinitely often).

Operational Semantics. Our operational semantics is given by SOS-style reduction rules, and our operational model is based on a Petri-net algebra of input enabled processes, extended with a notion of fairness. Our definitions of strong and weak fairness satisfy two of the criteria given in [AFK88]; namely, they are liveness enhancing and feasible.

We believe that our Petri Net description (Definition 3.2) is general enough to encompass the (event-related) behaviors of threads in the Java programming language; thus, these can be added as primitives to the algebra without affecting the results. Thus, from a programming language viewpoint, our semantics is independent of the particular syntax that we have chosen for `Triveni`, making the semantics a study of `Triveni` and the relevant aspects of the Java programming language.

Our operational model is the precise formalization of the `JavaTriveni` implementation.

Denotational Semantics. In order to support local reasoning on the behavior of programs, our denotational semantics is is *compositional* for all our process operators (Theorem 3.9). These results hold for both the (strongly/weakly) fair traces variants of the semantics.

The denotational semantics serves as the basis for a non-definability result (Theorem 3.10). All the `Triveni` combinators – with the exception of one preemption operator – have the following property: they are closed on the class of `Triveni` processes for which our strong fair semantics and weak fair semantics coincide. Namely, this property is satisfied by the preemption combinator for process abortion, but violated by the preemption combinator for process suspension. Hence, this result justifies the introduction of these preemption combinators as primitives in `Triveni`.

Logical semantics. We describe a logical semantics based on propositional linear-time temporal logic (PLTL) [MP91]. The primary purpose of the logical semantics is to show that our definitions of fairness are reasonable from a different viewpoint (Theorem 4.2), and to clarify the relationship between our algebra and synchronous programming languages. This semantics also serves as the basis for specification-based testing (of safety properties expressed in PLTL) in the implementation of `JavaTriveni` (Theorem 4.3). From a technical viewpoint, this semantics is quite standard and yields algorithms for deciding PLTL properties using model-checking techniques.

Related and Future Work. From a programming language viewpoint, one aim of the semantic study of the `Triveni` project is the potential use of concurrency theory techniques for a formal semantic description of substantial fragments of Java — one might even use poetic license and dare to dream of something with the precision and completeness of the formal definition of ML. In this context, the contribution of this paper is to demonstrate that a simple compositional treatment of fairness is necessary and possible. In the future, we plan to study the issue of mobility [MPW89, Mil91, FGL$^+$96], namely dynamic channel creation and passing. We expect that this will nicely enhance `Triveni`, since mobility permits a uniform semantic treatment of dynamic channels and process creation [San92] and the rudiments of object-oriented programming [Wal91].

Our work inherits the ideas of preemption combinators and input-enabled processes from synchronous programming languages such as Esterel [BG92], Lustre [HCRP91], Signal [GBGM91], Statecharts [Har87] *etc.*. (See, for instance, [Ber93, BB91, Hal93] for surveys.). From the viewpoint of synchronous programming languages, the semantic descriptions in this paper demonstrate the extent to which "instantaneous is approximated by eventually + fairness" ; this relationship is explained more precisely in the section on logical semantics. Indeed, a portion of our work can be viewed as adding notions of asynchrony to synchronous programming languages [BRS93]. While `Triveni` is (intentionally) less expressive than the language of [BRS93], we remark that the issues and context of the semantic study of fairness of this paper remain relevant to [BRS93]. In particular, we believe that the methods used to address issues of fairness in this paper do not conflict with the methods used to describe the semantics of the more powerful language of [BRS93], and we plan to study the extension of our results to [BRS93].

Definability results distinguishing different notions of fairness have been studied extensively in the dataflow literature, e.g. see [PS88b, PS88a, Sta90]. Similar results in SCCS distinguish different delay operators [CP91]. In contrast, our definability study focuses on distinguishing preemption combinators; we use the different notions of fairness as tools in this study.

Our work is related to I/O Automata (e.g., see [LT89, GSSAL94, Seg92]), Complete Trace Structures [Dil88], and Receptive Process Theory [Jos92]. These theories distinguish input and output actions and require all input actions to be enabled in every reachable state of the system. However, in all of these theories, only components with disjoint output alphabets can be composed in parallel; this restriction is essential for the substitutivity of the fair traces semantics. Since `Triveni` is the basis for a programming language, this restriction is far too stringent for our purposes, and our fair traces semantics are compositional even for parallel processes whose alphabets may intersect.

Our work is strongly inspired by the elegant results of Vaandrager [Vaa91], who also lifts this restriction, defines a general class of I/O Calculi and gives a definition of (weakly) fair traces based on the actual proof derivations of terms. He then shows that his (weakly) fair traces semantics is substitutive for any I/O Calculus. Our process calculus `Triveni` is in fact an I/O Calculus, and hence his results immediately imply that his weakly fair trace semantics is substitutive for `Triveni`. However, our fair traces semantics differ from that of Vaandrager in a few important respects. First, our results hold for strong fairness and weak fairness, and hence yield our non-definability result. Secondly, we justify our definitions via an alternative treatment of fairness – namely, a standard temporal logic based analysis. Thirdly, the Petri net basis of our semantic study – in particular, the succinct coding of parallel composition – motivates an efficient implementation of `JavaTriveni` programs: the size of the implementation is linear in the size of the program. Finally, in contrast to [Vaa91], our semantics are fair only with respect to the parallel composition operator, and not with respect to different event occurrences that arise from loop unwinding. We clarify this point later, merely noting here that our view permits our semantics to satisfy the equation $\texttt{LOOP}\ p = p;\ \texttt{LOOP}\ p$. This equation is important in a programming context such as `Triveni`. On the other hand, we emphasize that our results apply only to process calculi on our particular class of Petri Nets — the interpretation of the general class of I/O Calculi in our class of Petri nets is a problem to be studied in the future.

Other related work [Hen87, BRV95, NC95] studies failures and testing congruences in the setting of process algebras with fairness. We restrict attention to trace-based semantics, primarily due to the strong connection with PLTL.

Rest of the paper. First, we introduce the process algebra via SOS rules. We follow with a description of the Petri net model. We establish an algebra of combinators on the Petri net model and show that the Petri net model satisfies the desiderata on fairness. This section includes a description of the denotational semantics based on traces. The section concludes with the compositionality theorem of the model of (strongly/weakly) fair traces, and the non-definablity theorem. Finally, we sketch of a logical semantics based on temporal logic and Buchi automata.

2 The Process Algebra

Let e range over a possibly infinite[2] set $\mathcal{E}$ of events. $e?$ denotes an input event, $e!$ denotes an output event, $\mathcal{E}?$ denotes $\{e?:\ e \in \mathcal{E}\}$, and $\mathcal{E}!$ denotes $\{e!:\ e \in \mathcal{E}\}$. There are three distinguished actions $\tau, \surd, f$ not in $\mathcal{E}! \cup \mathcal{E}?$. We write α to denote $\{\surd, f\} \cup \mathcal{E}? \cup \mathcal{E}!$.

The rest of this section illustrates input-enabledness and the preemption combinators in the context of a concrete syntax of processes and labeled transition systems. (Recall, however, that our main results allow a large class of Petri Nets to be used as base cases in the algebra.) Note that the processes are input-enabled; all our constructions will carefully ensure that there is a transition on any input event.

$$\begin{array}{rcl} p & ::= & \texttt{NIL}\ \mid \texttt{DONE} \mid \texttt{EMIT}\ e! \mid p \mid\mid p \mid p\ ;p \mid\ p\ \texttt{HIDE}\ e \mid \tau{\rightarrow}p \mid e?{\rightarrow}p \\ & & \texttt{LOOP}\ p \mid\ \texttt{DO}\ p\ \texttt{WATCH}\ e?\ \texttt{TOUT}\ p \mid\ \texttt{SUSP}\ p\ \text{on}\ e?\ \texttt{RES}\ e? \mid\ \texttt{WAKE}\ p\ \text{on}\ e?\ \texttt{SLEEP}\ e? \end{array}$$

The f action is special, useful in the treatment of fairness. All processes idle on the f action.

$$\boxed{\quad p \xrightarrow{f} p \quad}$$

The processes NIL and DONE cannot perform any output action in $\mathcal{E}!$. The difference between them is that DONE terminates, as indicated by the $\surd$ transition. Importantly, our definition of fairness will guarantee that every fair trace of DONE will contain an occurrence of $\surd$.

$$\boxed{\texttt{NIL} \xrightarrow{e?} \texttt{NIL} \qquad \texttt{DONE} \xrightarrow{e?} \texttt{DONE} \qquad \texttt{DONE} \xrightarrow{\surd} \texttt{NIL} \qquad e \in \mathcal{E}}$$

The process EMIT can perform an output action and evolve to DONE. Our definition of fairness will guarantee that every fair trace of $\texttt{EMIT}\ e!$ will contain $e!$ and $\surd$.

[2] However, the decidability result for model-checking of `Triveni` processes only holds when this set is finite.

$$\texttt{EMIT}\ e! \xrightarrow{e!} \texttt{DONE} \qquad \texttt{EMIT}\ e! \xrightarrow{e'?} \texttt{EMIT}\ e! \qquad e' \in \mathcal{E}$$

Process $\tau{\rightarrow}p$ can perform a silent action τ and evolve to p.

$$\tau{\rightarrow}p \xrightarrow{\tau} p \qquad \tau{\rightarrow}p \xrightarrow{e?} \tau{\rightarrow}p \qquad e \in \mathcal{E}$$

On receipt of event e, process $e?{\rightarrow}p$ evolves to p; on any other input event the process terminates.

$$e?{\rightarrow}p \xrightarrow{e?} p \qquad e?{\rightarrow}p \xrightarrow{e'?} \texttt{DONE} \qquad e' \in \mathcal{E} - \{e\}$$

In parallel composition, $e?$ and $e!$ transitions synchronize. Note that the rules ensure that $e!$ is broadcast; i.e., a single $e!$ satisfies all $e?$ requests at this transition.

$$\frac{p \xrightarrow{e?} p' \quad q \xrightarrow{e!} q'}{p \mid\mid q \xrightarrow{e!} p' \mid\mid q'} \qquad \frac{p \xrightarrow{e!} p' \quad q \xrightarrow{e?} q'}{p \mid\mid q \xrightarrow{e!} p' \mid\mid q'} \qquad \frac{p \xrightarrow{e?} p' \quad q \xrightarrow{e?} q'}{p \mid\mid q \xrightarrow{e?} p' \mid\mid q'} \qquad e \in \mathcal{E}$$

Parallel components can evolve autonomously on τ.

$$\frac{p \xrightarrow{\tau} p'}{p \mid\mid q \xrightarrow{\tau} p' \mid\mid q} \qquad \frac{q \xrightarrow{\tau} q'}{p \mid\mid q \xrightarrow{\tau} p \mid\mid q'}$$

Parallel composition terminates when both components terminate; the component terminating later emits the required $\surd$ action.

$$\frac{p \xrightarrow{\surd} p'}{p \mid\mid q \xrightarrow{\tau} q} \qquad \frac{q \xrightarrow{\surd} q'}{p \mid\mid q \xrightarrow{\tau} p}$$

Hiding e is intended to model local events. All internal $e?$ transitions are restricted, and all internal $e!$ transitions are transformed to τ actions.

$$\frac{p \xrightarrow{e!} p'}{p\ \texttt{HIDE}\ e \xrightarrow{\tau} p'\ \texttt{HIDE}\ e} \qquad p\ \texttt{HIDE}\ e \xrightarrow{e?} p\ \texttt{HIDE}\ e$$

$$\frac{p \xrightarrow{\alpha} p'}{p\ \texttt{HIDE}\ e \xrightarrow{\alpha} p'\ \texttt{HIDE}\ e} \qquad \alpha \in \{\tau, \surd\} \cup \mathcal{E}? \cup \mathcal{E}! - \{e?, e!\}$$

Sequential composition works as expected.

$$\frac{p \xrightarrow{\surd} p'}{p\ ;\ q \xrightarrow{\tau} q} \qquad \frac{p \xrightarrow{\alpha} p'}{p\ ;\ q \xrightarrow{\alpha} p'\ ;\ q} \qquad \alpha \in \{\tau\} \cup \mathcal{E}? \cup \mathcal{E}!$$

$\texttt{LOOP}\ p$ models the looping construct. As is usually the case in event-driven programming languages, we use loop rather than guarded recursion; however, we note that all of our results hold for guarded recursion as well.

$$\frac{p \xrightarrow{\alpha} p'}{\texttt{LOOP}\ p \xrightarrow{\alpha} p'\ ;\ \texttt{LOOP}\ p} \qquad \frac{p \xrightarrow{\surd} p'}{\texttt{LOOP}\ p \xrightarrow{\tau} \texttt{LOOP}\ p} \qquad \alpha \in \{\tau\} \cup \mathcal{E}? \cup \mathcal{E}!$$

The watchdog $\texttt{DO}\ p\ \texttt{WATCH}\ e?\ \texttt{TOUT}\ q$ terminates and invokes q on receipt of an external e ($e?$) or an internal e ($e!$).

$$\texttt{DO}\ p\ \texttt{WATCH}\ e?\ \texttt{TOUT}\ q \xrightarrow{e?} q \qquad \frac{p \xrightarrow{e!} p'}{\texttt{DO}\ p\ \texttt{WATCH}\ e?\ \texttt{TOUT}\ q \xrightarrow{e!} q}$$

Process p continues to evolve until either $e?$ or $e!$ happens or p terminates; the termination of p also terminates the watchdog. Note that internal $e?$ events of p are restricted.

$$\frac{p \xrightarrow{\alpha} p'}{\texttt{DO}\ p\ \texttt{WATCH}\ e?\ \texttt{TOUT}\ q \xrightarrow{\alpha} \texttt{DO}\ p'\ \texttt{WATCH}\ e?\ \texttt{TOUT}\ q} \quad \alpha \in \{\tau\} \cup \mathcal{E}? \cup \mathcal{E}! - \{e?, e!\}$$

$$\frac{p \xrightarrow{\surd} p'}{\texttt{DO}\ p\ \texttt{WATCH}\ e?\ \texttt{TOUT}\ q \xrightarrow{\surd} p'}$$

`SUSP` p on e_1? `RES` e_2? suspends on receipt of an external e_1 (e_1?) or internal e_1 (e_1!), and resumes on receipt of an external e_2 (e_2?).

$$\texttt{SUSP}\ p\ \text{on}\ e_1?\ \texttt{RES}\ e_2? \xrightarrow{e_1?} \texttt{WAKE}\ p\ \text{on}\ e_2?\ \texttt{SLEEP}\ e_1?$$

$$\frac{p \xrightarrow{e_1!} p'}{\texttt{SUSP}\ p\ \text{on}\ e_1?\ \texttt{RES}\ e_2? \xrightarrow{e_1!} \texttt{WAKE}\ p'\ \text{on}\ e_2?\ \texttt{SLEEP}\ e_1?}$$

`SUSP` p on e_1? `RES` e_2? allows p to evolve and terminate.

$$\frac{p \xrightarrow{\alpha} p'}{\texttt{SUSP}\ p\ \text{on}\ e_1?\ \texttt{RES}\ e_2? \xrightarrow{\alpha} \texttt{SUSP}\ p'\ \text{on}\ e_1?\ \texttt{RES}\ e_2?} \quad \alpha \in \{\tau\} \cup \mathcal{E}? \cup \mathcal{E}! - \{e_1?, e_1!\}$$

$$\frac{p \xrightarrow{\surd} p'}{\texttt{SUSP}\ p\ \text{on}\ e_1?\ \texttt{RES}\ e_2? \xrightarrow{\surd} p'}$$

`WAKE` p on e_2? `SLEEP` e_1? sleeps until an external e_2 (e_2?) happens and goes back to sleep on e_1.

$$\texttt{WAKE}\ p\ \text{on}\ e_2?\ \texttt{SLEEP}\ e_1? \xrightarrow{e_2?} \texttt{SUSP}\ p\ \text{on}\ e_1?\ \texttt{RES}\ e_2?$$

$$\texttt{WAKE}\ p\ \text{on}\ e_2?\ \texttt{SLEEP}\ e_1? \xrightarrow{e?} \texttt{WAKE}\ p\ \text{on}\ e_2?\ \texttt{SLEEP}\ e_1? \quad e \in \mathcal{E} - \{e_2\}$$

Defining Other Combinators.

Example 2.1 The analogue of prefixing, a process that waits for e and then starts p can be defined as

$$\texttt{AWAIT}\ e? {\rightarrow} p = \texttt{DO}\ \texttt{NIL}\ \texttt{WATCH}\ e?\ \texttt{TOUT}\ p$$

.

Example 2.2 This example illustrates that non-determinism can arise in `Triveni` via the delays in delivery of events and competition between events. $\tau.P_1 + \tau.P_2 + \tau.P_3$ can be defined as

$$\begin{array}{l}
[\ \texttt{DO}\ (\ \texttt{DO}\ \texttt{EMIT}\ a_1!\ ;\ P_1\ \texttt{WATCH}\ a_2?\ \texttt{TOUT}\ \texttt{DONE})\ \texttt{WATCH}\ a_3?\ \texttt{TOUT}\ \texttt{DONE}\ \ || \\
\ \ \texttt{DO}\ (\ \texttt{DO}\ \texttt{EMIT}\ a_2!\ ;\ P_2\ \texttt{WATCH}\ a_1?\ \texttt{TOUT}\ \texttt{DONE})\ \texttt{WATCH}\ a_3?\ \texttt{TOUT}\ \texttt{DONE}\ \ || \\
\ \ \texttt{DO}\ (\ \texttt{DO}\ \texttt{EMIT}\ a_3!\ ;\ P_3\ \texttt{WATCH}\ a_2?\ \texttt{TOUT}\ \texttt{DONE})\ \texttt{WATCH}\ a_1?\ \texttt{TOUT}\ \texttt{DONE}]\ \texttt{HIDE}\ a_1\ \texttt{HIDE}\ a_2\ \texttt{HIDE}\ a_3
\end{array}$$

where a_1, a_2, a_3 are fresh event names not occurring in P_1, P_2, P_3.

3 Petri-Net Semantics

We now describe an algebra of nets underlying `Triveni`. The heart of this section is Definition 3.2 that identifies *RFT nets*, our class of nets extended with a notion of fairness. We emphasize that *any* primitive process P satisfying Definition 3.2 can be added as a primitive to the algebra, without affecting the semantic results. In addition, if the (strongly/weakly) fair traces of the Petri net model of P form an ω-regular set, all the decidability results of Section 4 also hold. This makes our semantics essentially independent of the particular syntax that we have chosen for the process algebra, and allows it to encompass the relevant aspects of the Java programming language.

The rest of this section is organized as follows. We begin by reviewing standard definitions to keep the paper self-contained. We follow with the definition of *RFT* nets, and study general properties of *RFT* nets with respect to fairness. Next we describe the algebra of *RFT* nets — in this subsection, we rely on the power of pictures, leaving the detailed definitions to the appendix.

3.1 Background

We use the standard definitions (*cf.* [Vog92]) of Petri nets and their operational behavior. We use the following notation. Let σ be a possibly infinite sequence. Then $|\sigma|$ is defined to be the length of σ if σ is finite; $|\sigma|$ is defined as ω if σ is infinite. The restriction of σ to an alphabet Σ is written $\sigma \uparrow_\Sigma$.

Definition 3.1 A labeled Petri Net, N, is a triple $\langle S_N, T_N, \mathit{Start}_N \rangle$, where S_N is the set of places, T_N is the set of transitions, and Start_N is the set of initially marked places (which contain "tokens"). Every transition, t, in T_N has a label $l_N(t)$, a preset $\mathit{pre}_N(t)$, and a post-set $\mathit{post}_N(t)$. Transitions are represented graphically as horizontal bars, places are represented as circles, and tokens are represented as dots in these circles. The preset of a transition is the set of places from which there is an arrow to the transition; the post-set of a transition is the set of places to which there is an arrow from the transition.

A *marking* of a net is an assignment of a non-negative number of "tokens" to each place in the net. A transition, t, is *enabled* under a marking iff every place in the preset of t contains at least one token. If a transition t is enabled in a marking, then t can *fire* by removing a token from each place in its preset and placing a token into each place in its post-set. We write $M \xrightarrow{t} M'$ if t is enabled in marking M, and firing t in M results in marking M'.

A *run* r of a net is a finite or infinite sequence, $M_1 t_1 M_2 t_2 \ldots$, of markings and transitions such that M_1 is the initial marking and $M_i \xrightarrow{t_i} M_{i+1}$ for all i with $1 \leq i < |r|$. The *reachable markings* of a net are exactly those markings that result from firing some run. A net is *1-safe* iff every place contains at most one token under any reachable marking. The *trace* of a run r is the projection of the sequence of transition labels of r onto $\mathcal{E}! \cup \mathcal{E}? \cup \{\surd, f\}$; in particular, all occurrences of τ are erased. The *traces* of N are the traces of runs of N.

3.2 *RFT* nets

RFT Nets are a subclass of labeled 1-safe Petri nets, which extend the above class of nets with a set of "fair places." The following definition captures quite a general notion of an input enabled process:

Definition 3.2 Let $\mathcal{E}$ be a (possibly infinite) set of events. Then $\langle S_N, T_N, \mathit{Start}_N, FS_N \rangle$ is a *RFT* Net iff

1. $\langle S_N, T_N, \mathit{Start}_N \rangle$ is a 1-safe, (possibly infinite) Petri net, with transitions labeled by actions in $\mathcal{E}? \cup \mathcal{E}! \cup \{\tau, \surd, f\}$. All transitions have non-empty presets.

   These conditions are standard in Petri-net-based semantics of process algebras [Vog92, Jat93].

2. Start_N (the initial marking) is finite, and the preset and the post sets of all transitions are finite. For any reachable marking M, the number of markings M' immediately reachable from M (i.e. $M \xrightarrow{t} M'$ for some transition t) is finite.

   This is a computability restriction capturing finite branching constraints (as adapted to input enabled processes). This condition is not restrictive since the primary source of infinite branching in specifications arises from the imposition of fairness considerations, and RFT nets capture the transition system *before* fairness considerations are imposed.

3. $FS_N \subseteq S_N$ is termed the set of "fair places." For every place $s \in FS_N$ and every transition $t \in T_N$ such that $l_N(t) \in \mathcal{E}?$, $s \notin \mathit{pre}_N(t) - \mathit{post}_N(t)$.

   This is the intentional information for modeling fairness — the key innovation in this definition.

4. There is some f-labeled transition enabled in every reachable marking. Furthermore, for all f-labeled transitions $t \in T_N$, $\mathit{pre}_N(t) = \mathit{post}_N(t)$.

   This condition can be satisfied by adding idling transitions on f to every place.

5. For all $\alpha \in \mathcal{E}?$, there is some α-labeled transition enabled in every reachable marking.

   This condition corresponds to input-enabledness.

6. Let M and M' be any reachable markings such that $M \xrightarrow{t} M'$ for some $\surd$-labeled transition $t \in T_N$. Then for all transitions $t' \in T_N$ and markings M'', if $M' \xrightarrow{t'} M''$, then $l_N(t') \in \mathcal{E}? \cup \{f\}$ and $M'' = M'$.

   This condition merely says that the net is "quiescent" once a $\surd$ transition is fired.

The role of the fair places of *RFT* nets is to ensure that if a transition whose label is in $\mathcal{E}! \cup \{\surd, \tau\}$ is enabled infinitely often (or continually) in a run, then it is fired infinitely often in that run. The following definition captures this intuition.

Definition 3.3 Let s be a place in a net N, and M be a reachable marking of N. Then s is *emptiable in* M iff there is some transition t enabled in M such that $s \in \mathit{pre}_N(t) - \mathit{post}_N(t)$, and we say that t *empties* s.

Let $r = M_1 t_1 M_2 t_2 \ldots$ be a run of a RFT net N, and let s be a place of N. Then r is *strongly fair for* s iff the following holds: if there are infinitely many M_j in r such that s is emptiable in M_j, then there are infinitely many t_k in r that empty s. We say that r is *strongly fair* iff it is strongly fair for all $s \in FS_N$.

Let $r = M_1 t_1 M_2 t_2 \ldots$ be a run of a RFT net N, and let s be a place of N. Then r is *weakly fair for* s iff the following holds: if there exists some i such that s is emptiable in M_j for all $j > i$, then there are infinitely many t_k in r that empty s. We say that r is *weakly fair* iff it is weakly fair for all $s \in FS_N$.

We note that Definition 3.3 and conditions (3) and (4) of Definition 3.2 together imply that fair places can be emptied only by transitions with labels in $\mathcal{E}! \cup \{\surd, \tau\}$.

The definitions of fair runs induce the definitions of fair traces.

Definition 3.4 The *strongly fair traces of* N, $[\![N]\!]_{\text{Strong}}$, is the set of infinite traces of strongly fair runs of N. The *weakly fair traces of* N, $[\![N]\!]_{\text{Weak}}$, is the set of infinite traces of weakly fair runs of N. $[\![N]\!]_{\text{Fin}}$ is the set of *finite traces of* N.

3.3 Feasibility

Our definition of fairness is *feasible* [AFK88], as adapted to input-enabled processes [Vaa91].

Lemma 3.5 Let N be any net. Then

- Let σ be a finite trace of N, and let γ be a (possibly infinite) sequence over $\mathcal{E}?$. Then there is some σ' such that γ is the projection of σ' onto $\mathcal{E}?$ and $\sigma\sigma'$ is an infinite strongly (weakly) fair trace of N.

- The set of finite prefixes of elements of $[\![N]\!]_{\text{Strong}}$, the set of finite prefixes of elements of $[\![N]\!]_{\text{Weak}}$, and the set $[\![N]\!]_{\text{Fin}}$ are all equal.

These assertions are proved by the construction of schedulers that essentially maintain a FIFO discipline (details omitted for space reasons). These schedulers permit every finite execution to be extended to an infinite (strongly/ weakly) fair one.

3.4 The Net Translation

We now define operations on RFT nets. In all the following figures, the word "tick" is used to represent the termination symbol $\surd$.

The base cases. Consider Figure 1 – the formal definition is given in Definition A.1 in the appendix. The (strong/weak) fair traces of `NIL` will be $(\mathcal{E}? \cup \{f\})^\omega$ since the net translation of `NIL`

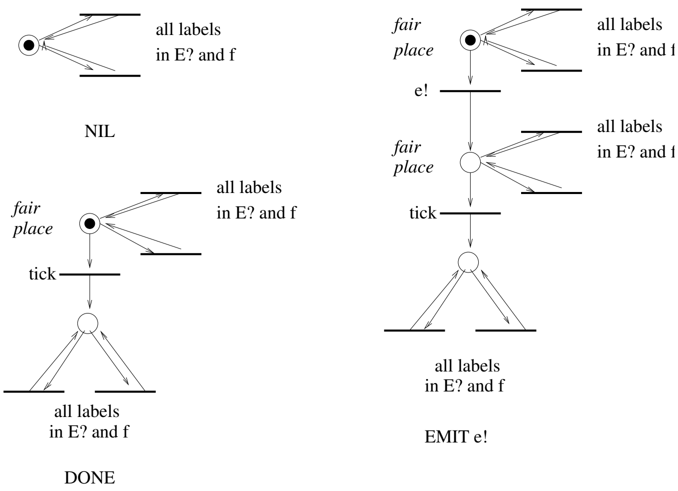

Figure 1: Net translation for `NIL` , `DONE`, and `EMIT` $e!$

has no fair places. For `DONE`, the initial place is a fair place, and the only transition that empties it is the one labeled $\surd$; hence, the definitions now ensure that any (strong/weak) fair trace of `DONE` will contain a $\surd$. For `EMIT` $e!$, the only transition that empties the initial (fair) place is the one labeled $e!$. Similarly, the only transition that empties the second (fair) place is $\surd$. The definitions now ensure that any (strong/weak) fair trace of `EMIT` $e!$ will contain these two events.

Prefixing. The descriptions of the net constructions for $e?{\rightarrow}N$ and guarded choice are not surprising and are given in Definition A.2 in the appendix. We merely note that the fair places of the resulting nets $e?{\rightarrow}\ N$ come from the union of the fair places of N and the fair places of `DONE`. Furthermore, the strongly fair traces of $e?{\rightarrow}N$ are of the form $(\mathcal{E}? - \{e?\})[\![\texttt{DONE}]\!]_{\text{Strong}} \cup e?s$, where $s \in [\![N]\!]_{\text{Strong}}$. A completely analogous property holds for weakly fair traces.

Hiding. N `HIDE` e is informally depicted in Figure 2 (the formal definition is in Definition A.3 of appendix). The fair places of the resulting net are completely induced by the fair places of N.

There are two kinds of strongly fair traces in $[\![N\ \texttt{HIDE}\ e?]\!]_{\text{Strong}}$. For the first kind, let σ be a some strongly fair trace of N that does not contain any occurrences of $e?$, and let $\sigma' = \alpha_1\alpha_2\ldots$ be the projection of σ onto $\mathcal{E}? \cup (\mathcal{E}! - \{e!\}) \cup \{\surd, f\}$. If σ' is infinite, then any sequence of the form $\sigma'' = \alpha_1 e?^* \alpha_2 e?^* \ldots$ is a strongly fair trace of N `HIDE` $e?$. (Namely, all occurrences of $e?$ in σ are

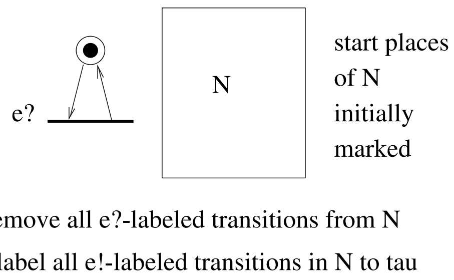

Figure 2: Net translation for N HIDE $e?$

restricted, all occurences of $e!$ in σ are hidden, and then finite sequences of $e?$ events are interspersed at arbitrary points in the middle.) For the second kind of strongly fair traces, let σ be a *finite* trace of N that does not contain any occurrences of $e?$, and let σ' and σ'' be as above (except that they are finite). Then $\sigma'' e?^\omega$ is a fair trace of N HIDE e whenever σf^ω is a fair trace of N. Thus, for hiding, the f event serves as a "quiescence detector" (much as $\surd$ serves as a termination detector). Infinite subsequences of this event predict when it is correct to append an infinite subsequence of $e?$ events to a finite trace. A completely analogous property holds for weakly fair traces.

Sequential composition. Figure 3 presents the (standard) intuition for the net construction for sequential composition (Definition A.4 of appendix); we merely note that the set of the fair places of N_1 ; N_2 is just the union of the fair places of N_1 and the fair places of N_2.

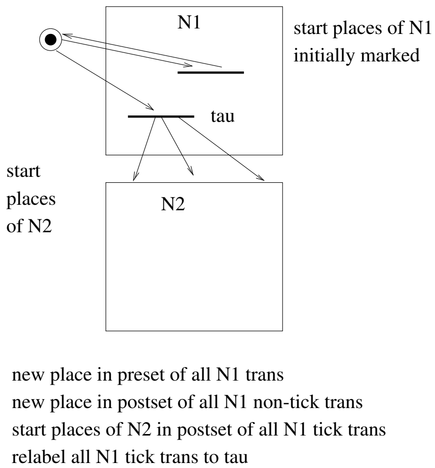

Figure 3: Net translation for N_1 ; N_2

The strongly fair traces of N_1 ; N_2 are of two kinds. Firstly, any strongly fair trace of N_1 is a strongly fair trace of N_1 ; N_2. Secondly, any finite terminated trace of N_1 concatenated with a strongly fair trace of N_2 is also a strongly fair trace of N_1 ; N_2. Similarly for weak fairness.

Loop. The definition of the net for LOOP N (Definition A.5 of appendix) follows standard intuitions (countable many copies of N connected by ;). The fair places of LOOP N are simply the union of the fair places of the countably many copies of N. Thus, the strongly fair traces of LOOP N are of two kinds. Firstly, any trace that involves infinite unwinding of the loop is a strongly fair trace of LOOP N. Secondly, any trace that involves finite unwinding of the loop and projects down to a strongly fair trace on the last unwinding is also a strongly fair trace of LOOP N. An analogous property holds for weakly fair traces.

The construction for LOOP shows that our definitions of fairness (both strong and weak) applies only to *event occurrences* rather than event names: in particular, loop unwinding does not preserve event occurrences. (This is in contrast to [Vaa91]).

Example 3.6 Consider a process $r = \texttt{LOOP}\ (\texttt{EMIT}\ e! + q)$. Our treatment considers an execution of r that continually chooses to execute the q process to be strongly fair, since the different enablings of $e!$ in the unwindings of the loop correspond to different event occurrences. Consequently, our treatment of fairness (both strong and weak) satisfies (in terms of fair traces): $\texttt{LOOP}\ p = p; \texttt{LOOP}\ p$

Watchdog. The watchdog combinator is described in Figure 4 (Definition A.6 of appendix). The fair places of the resulting net are given by the union of the fair places of N_1, N_2.

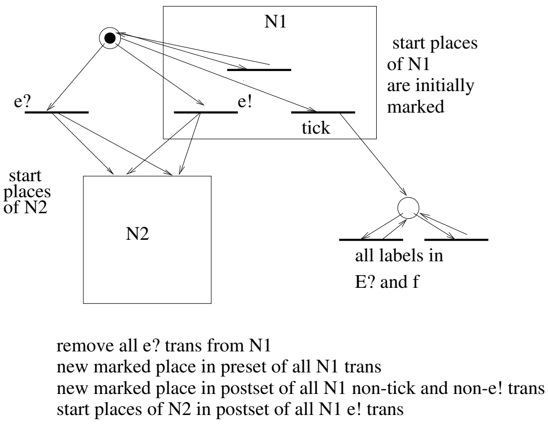

Figure 4: Net translation for $\texttt{DO}\ N_1\ \texttt{WATCH}\ e?\ \texttt{TOUT}\ N_2?$

The strongly fair traces of $\texttt{DO}\ N_1\ \texttt{WATCH}\ e?\ \texttt{TOUT}\ N_2?$ are of two kinds. Firstly, any strongly fair trace of N_1 that does not contain $e?$ or $e!$ is a strongly fair trace of $\texttt{DO}\ N_1\ \texttt{WATCH}\ e?\ \texttt{TOUT}\ N_2$. Secondly, any finite trace of N_1 that does not contain $e?$ or $e!$ except as its last element and is followed by a strongly fair trace of N_2 is also a strongly fair trace of $\texttt{DO}\ N_1\ \texttt{WATCH}\ e?\ \texttt{TOUT}\ N_2$. An analogous property holds for weakly fair traces.

Suspension–Activation. The definition for the supension-activation combinator follows the intuition described in Figure 5 on the next page (Definition A.7 of appendix). The fair places of the resulting net are given by the fair places of N.

The supension-activation combinator illustrates the difference between strong and weak fairness. Indeed, this combinator is the sole way in Triveni to have alternating enabling and disabling of event occurrences.

Example 3.7 Consider the process

$$[\,\texttt{SUSP}\ \ \texttt{EMIT}\ e!\ \text{on}\ a?\ \texttt{RES}\ b?] \mid\mid [\texttt{LOOP}\ \texttt{EMIT}\ a!] \mid\mid [\texttt{LOOP}\ \texttt{EMIT}\ b!]$$

Every strongly fair trace of this process must contain the event $e!$. However, there are some weakly fair traces that do not contain the event $e!$.

In general, any infinite trace that involves infinitely many suspensions and activations of N is a weakly fair trace. Such a trace is strongly fair only if its projection to N is strongly fair.

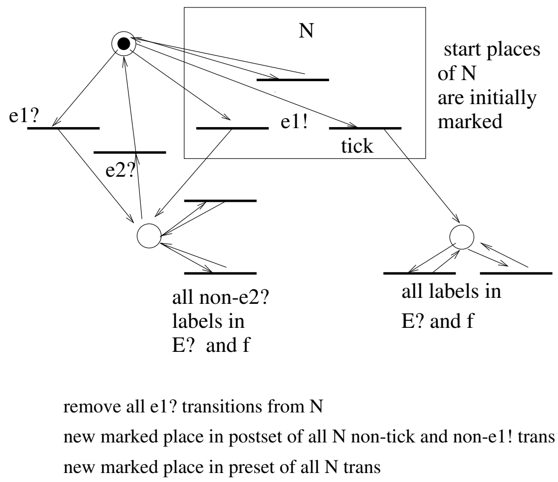

Figure 5: Net translation for $\texttt{SUSP}\ N$ on $e_1?\ \texttt{RES}\ e_2?$

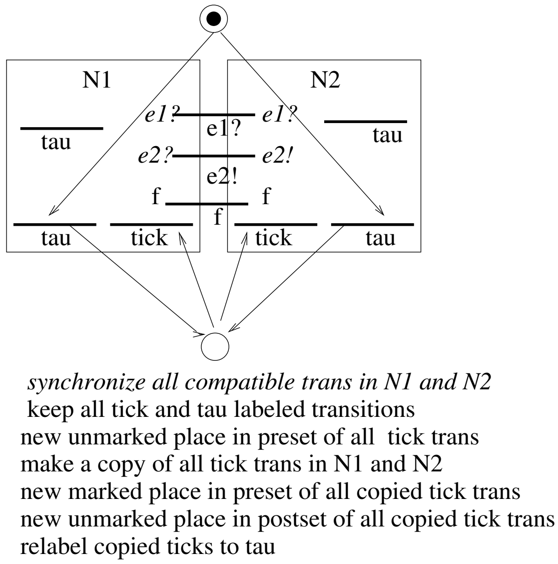

Figure 6: Net translation for $N_1 \mid\mid N_2$

Parallel composition. For parallel composition, it helps to formalize the idea of compatible labels. Let $\alpha_1, \alpha_2 \in \mathcal{E}? \cup \mathcal{E}! \cup \{\tau, \surd, f\}$. We say that α_1 and α_2 are *compatible* iff either $\alpha_1 = e? = \alpha_2$ for some $e \in \mathcal{E}$ (in this case, the compatible label is $e?$); or $\alpha_1 = e?$ and $\alpha_2 = e!$ (or vice-versa) for some $e \in \mathcal{E}$ (in this case, the compatible label is $e!$); or $\alpha_1 = f = \alpha_2$ (in this case, the compatible label is f). For compatible α_1, α_2, we write *compatible* (α_1, α_2) to denote their compatible label. Figure 6 describes the construction for parallel composition; note that the fair places of the resulting net is the union of the fair places of the individual nets. Indeed, the "disjoint-union" of places that naturally arises in the Petri net presentation—a reflection of its "truly concurrent nature"—is crucial to our theory. The definition (formally in Definition A.8 of appendix) also reflects that our parallel composition terminates when both components terminate; the first terminating component emits a τ rather than a $\surd$, and the later terminating component emits the required $\surd$ action.

The definition of parallel composition preserves intentional information about which parallel component emitted a particular transition with label in $\mathcal{E}! \cup \{\tau, \surd\}$. The theories of [Dil88, LT89] accomplish this by requiring the stringent restriction that parallel components have disjoint labels. The "disjoint-union" of places that naturally arise because of the "spatial true concurrency" nature of Petri nets, together with our condition (3) on fair places of RFT nets, allow us to remove this restriction while preserving compositionality. In particular, the emptiable condition on places preserves intentional information about which parallel component emitted an output action.

These intuitions are summarized in the following discussion. Any fair place s in an RFT net $N = N_1 \mid\mid N_2$ must be a fair place of exactly one of N_1 or N_2. We now argue informally that s is emptiable in a marking of N whenever s is emptiable in the induced marking (M_1) of N_1 or (M_2 of) N_2. Assume that s is a fair place of N_1 (the other case is symmetric). We argue informally as follows. Suppose that s is emptiable in marking M of N. Thus, there must be some transition t in N with label in $\mathcal{E}! \cup \{\tau, \surd\}$ such that $s \in \mathit{pre}_N(t) - \mathit{post}_N(t)$. The key case is when t is labeled with $e!$. In this case, t must be of the form $\langle t_1, t_2 \rangle$, where $e!$ is the compatible label of $l_{N_1}(t_1)$ and $l_{N_2}(t_2)$. Thus, exactly one of t_1, t_2 must be $e?$-labeled in their respective nets, and exactly one of them must be $e!$-labeled. Since $s \in \mathit{pre}_N(t) - \mathit{post}_N(t)$, the parallel composition operator implies that $s \in \mathit{pre}_{N_1}(t) - \mathit{post}_{N_1}(t)$; thus, condition (3) of RFT nets implies that t_1 cannot be $e?$-labeled in N_1. Hence, it must be $e!$-labeled, and so s is emptiable in marking M_1 of N_1.

3.5 Theorems

The net constructions are reasonable. For any process p, net(p) is defined inductively: the base processes are defined as the corresponding RFT nets, and the process operators are defined compositionally from the corresponding net operators in the obvious manner. The following lemma is simply a coherence check: ignoring the fairness information in RFT nets takes us back to a familiar transition system point of view.

Lemma 3.8

- RFT nets are closed under all of the net operators.
- The labeled transition systems of p and net(p) are strongly bisimilar.

Compositionality of Fair-Trace Semantics. The following theorem is the fruit of labor of the preceding pages. The formal proof is omitted for space reasons. However, the intuitions behind the proofs of the key cases of hiding and parallel composition have been described informally earlier.

Theorem 3.9 The finite trace semantics $[\![\cdot]\!]_{\text{Fin}}$, the strongly fair traces semantics $[\![\cdot]\!]_{\text{Strong}}$, and the weakly fair traces semantics $[\![\cdot]\!]_{\text{Weak}}$ are each compositional for all the process operators on nets.

Non–definability result. All process operators except suspend–activate are closed on nets that do not distinguish weak and strong fairness, i.e. nets N such that $[\![N]\!]_{\text{Strong}} = [\![N]\!]_{\text{Weak}}$. (This makes intuitive sense since these operators do not ever reenable a disabled transition.) As we have seen earlier (Example 3.7), the suspend–activate combinator does not satisfy this invariant. Thus,

Theorem 3.10 The suspend-activate combinator is not definable from the following algebra:

- (External) constants are RFT nets N satisfying $[\![N]\!]_{\text{Strong}} = [\![N]\!]_{\text{Weak}}$.
- All process combinators except the suspend-activate combinator are allowed.

Implementation notes In the implementation of the Java programming language, there are two queues[3], one each for active and suspended processes. The standard Java scheduler (in most non-Unix platforms) timeshares between the processes in the active queue. There are no guarantees provided about the position of a process when it moves to the active queue from the the suspended

[3]More precisely, two queues per priority level; we ignore priority issues here since all threads in the implementation of `Triveni` have the same priority

queue. Thus, a straightforward implementation that realizes event emission via threads implements weak fairness. Our current implementation implements strong fairness using the scheduler in the proof of the feasibility criterion for strong fairness. This scheduler essentially manipulates the above two queues by itself.

Notes on Equivalence robustness [AFK88] defines the criterion of *equivalence robust* as follows: reorderings of independent event occurrences do not affect the fairness of an execution. The following example suggests that a reasonable notion of fairness cannot be (non-trivially) equivalence robust.

$$\begin{array}{l}[\texttt{LOOP EMIT } a!] \mid\mid [\texttt{LOOP EMIT } b!] \mid\mid [\texttt{LOOP EMIT } c!] \mid\mid [\texttt{LOOP EMIT } d!] \mid\mid \\ (\texttt{ SUSP } [\texttt{ SUSP EMIT } e! \text{ on } a? \texttt{ RES } b?] \text{ on } c? \texttt{ RES } d?)\end{array}$$

Since there is no non-trivial causal relationship among the production of the $a!, b!, c!, d!$ events, equivalence robustness requires one to be able to reorder these events without affecting fairness. However, the trace $(a!c!d!b!)^\omega$ is strongly unfair whereas the trace $(a!c!b!d!)^\omega$ is strongly fair. Similar but more involved examples demonstrate this for the case of weak fairness.

4 Temporal Logic

We now describe the connections with linear-time temporal logic and model-checking. For the purposes of this section, we are interested in properties that have to do with events and fairness (rather than termination). In this section, we assume that the set of events $\mathcal{E}$ is finite. We use propositional linear-time logic without next and previous (i.e., propositional linear-time logic without immediate operators [MP91]), and where the propositions are output events.

$$\phi \ ::= \ e! \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \mid \Box\phi \mid \Diamond\phi \mid \phi\, \mathcal{U}\, \phi \mid \phi\, \mathcal{W}\, \phi \mid \boxminus\phi \mid \diamondminus\phi \mid \phi\, \mathcal{S}\, \phi \mid \phi\, \mathcal{B}\, \phi$$

Let `nil` be defined as $\Box\neg(\bigvee_{e\in\mathcal{E}} e!)$.

Instantaneous $\approx$ eventually + fairness. We use the standard definition [MP91] for when a possibly infinite sequence σ over $\mathcal{E}? \cup \mathcal{E}! \cup \{\surd, f\}$ satisfies a PLTL formula. In particular, (σ, j) satisfies $e!$ iff $e!$ is the j^{th} element of σ.

Definition 4.1 Let p be a `Triveni` process, ϕ a formula. Then *p strong (weak) fairly satisfies ϕ* iff all the infinite strong (weak) fair traces of p satisfy ϕ; namely, all sequences in $[\![p]\!]_{\text{Strong}}$ ($[\![p]\!]_{\text{Weak}}$) satisfy ϕ. A similar definition holds for RFT nets.

A sample of the properties follows. The following properties clarify the nature of the approximation to synchronous programming languages achieved in `Triveni`; in all cases, instantaneously is replaced by eventually in the presence of fairness. In particular, the liveness properties are guaranteed because of fairness: they would not hold otherwise.

$$\frac{p \models \Diamond\phi}{(\texttt{AWAIT } e? \rightarrow p \mid\mid \texttt{EMIT } e!) \models \Diamond\phi} \qquad \frac{q \models \Diamond\phi}{((\texttt{ DO } p \texttt{ WATCH } e? \texttt{ TOUT } q) \mid\mid \texttt{EMIT } e!) \models \Diamond(\phi \vee \texttt{nil})}$$

$$\texttt{EMIT } e! \models \Diamond e! \qquad ((\texttt{ SUSP } p \text{ on } e_1? \texttt{ RES } e_2?) \texttt{ HIDE } e_2) \mid\mid \texttt{EMIT } e_1! \models \Diamond\texttt{nil}$$

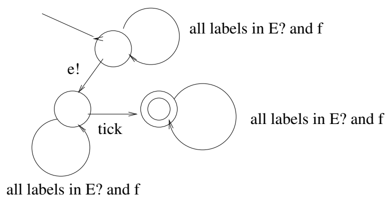

Figure 7: Buchi automaton for `EMIT` $e!$

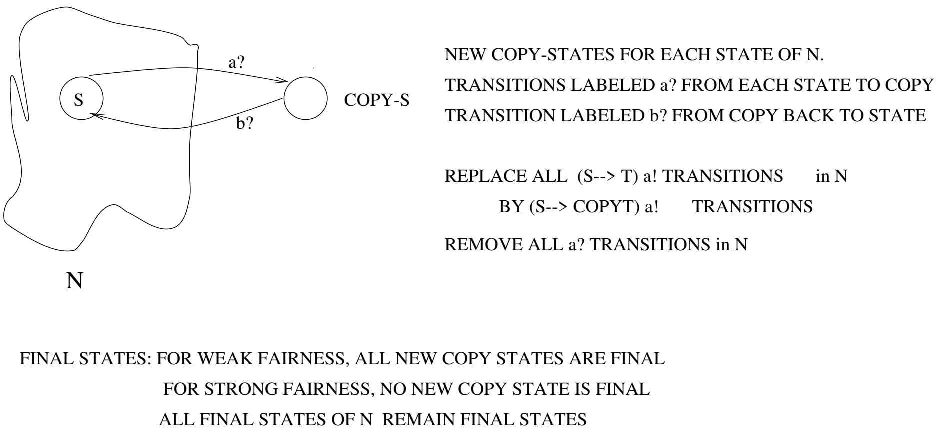

Figure 8: Buchi automaton for `SUSP` N on $a?$ `RES` $b?$

Alternate justification for our fairness notions We can associate constructions on Buchi automata with process combinators. The constructions are straightforward and follow the intuitions of the Petri net constructions; for example, the Buchi automaton for the process `EMIT` $e!$ and the suspend–activate combinator are in Figures 7 and 8, and the construction for parallel composition is a simple variant of the standard intersection construction for Buchi automata.

Theorem 4.2

- The process combinators are closed on the subclass of nets whose (strong/weak) fair traces form an ω-regular set.
- It is decidable if a `Triveni` process p (strong/weak) fairly satisfies a PLTL formula ϕ.

The decidability results follow standard results on model-checking for Buchi automata (c.f. [Eme90]).

Safety properties The finite trace semantics $[\![\cdot]\!]_{\text{Fin}}$ is closely related to the subclass of safety properties ψ [MP91]: modifying the Buchi automata discussed earlier, and using the results surveyed in [Eme90], we get specializations of the earlier results to finite traces and safety properties. In this case, using the fact that an infinite sequence violates a safety property only if some finite prefix of the sequence violates the safety property [MP91], we get a sharper full abstraction result in the spirit of the result for CSP [BHR84].

Theorem 4.3

- The process combinators are closed on the subclass of nets whose finite traces form a regular set.
- It is decidable if a `Triveni` process p satisfies a PLTL safety formula ϕ.
- The $[\![\cdot]\!]_{\text{Fin}}$ semantics is compositional, adequate and fully abstract (with respect to all the process operators) for observing safety properties: namely, for any RFT nets N_1, N_2 $[\![N_1]\!]_{\text{Fin}} = [\![N_2]\!]_{\text{Fin}}$ iff N_1 and N_2 satisfy the same set of safety properties in any process context.

References

[AFK88] K.R. Apt, N. Francez, and S. Katz. Appraising fairness in languages for distributed programming. *Distributed Programming*, 2:226–241, 1988.

[BB91] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. In *Special issue on Another Look at Real-time Systems*, Proceedings of the IEEE, September 1991.

[Ber93] G. Berry. Preemption in concurrent systems. In *Proc. of FSTTCS*. Springer-Verlag, 1993. LNCS 781.

[BG92] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: design, semantic s, implementation. *Science of Computer Programming*, 19:87–152, 1992.

[BHR84] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the Association of Computing machinery*, 31(3):560–599, 1984.

[BRS93] G. Berry, S. Ramesh, and R.K. Shyamsunder. Communicating reactive processes. In *Proceedings of Twentieth ACM Symposium on Principles of Programming Languages*, pages 85 – 98, 1993.

[BRV95] E. Brinksma, A. Rensink, and W. Vogler. Fair testing. In *Proceedings of International Conference on Concurrency Theory, Volume 962 of Lecture Notes in Computer Science*, pages 313–327, 1995.

[CJJ+98a] C. Colby, L. J. Jagadeesan, R. Jagadeesan, K. Lä ufer, and C. Puchol. Design and implementation of Triveni: A process-algebraic API for threads + events. In *Proceedinngs of the 1998 IEEE International Conference on Computer Languages*. IEEE Computer Press, 1998. To appear.

[CJJ+98b] C. Colby, L. J. Jagadeesan, R. Jagadeesan, K. Lä ufer, and C. Puchol. Objects and concurrency in `Triveni`: A telecommunication case study in Java. In *Proceedinngs of the Fourth USENIX Conference on Object Oriented Technologies and Systems*, 1998. To appear.

[CP91] Carol Critchlow and Prakash Panangaden. The expressive power of delay operators in SCCS. *Acta Informatica*, 28(5):447–452, 1991.

[Dil88] D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1988.

[Eme90] E. A. Emerson. *Handbook of Theoretical Computer Science*, chapter Temporal and modal logic, pages 995–1072. Elsevier/The MIT Press, 1990.

[FGL+96] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *7th International Conference on Concurrency Theory (CONCUR'96)*, pages 406–421, Pisa, Italy, August 1996. Springer-Verlag. LNCS 1119.

[GBGM91] P. Le Guernic, M. Le Borgne, T. Gauthier, and C. Le Maire. Programming real time applications with SIGNAL. In *Special Issue on Another Look at Real-time Systems*, Proceedings of the IEEE, September 1991.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GSSAL94] R. Gawlick, R. Segala, J.F. Søgaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. In *Proceedings 21st ICALP*, number 820 in Lecture Notes in Computer Science, 1994.

[Hal93] N. Halbwachs. *Synchronous programming of reactive systems*. The Kluwer international series in Engineering and Computer Science. Kluwer Academic publishers, 1993.

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

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data-flow programming language LUSTRE. *Proceedings of the IEEE*, 79:1305–1320, 1991.

[Hen87] M. Hennessy. An algebraic theory of fair asynchronous communicating processes. *Theoretical Computer Science*, 49:121–143, 1987.

[Jat93] Lalita Jategaonkar. *Observing "True" Concurrency*. PhD thesis, Massachusetts Institute of Technology, September 1993.

[Jos92] Mark B. Josephs. Receptive process theory. *Acta Informatica*, 29:17–31, 1992.

[LT89] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989. Also available as MIT Technical Memo MIT/LCS/TM-373.

[Mil91] Robin Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK, oct 1991. Appeared in *Proceedings of the International Summer School on Logic and Algebra of Specification*, Marktoberdorf, August 1991. Reprinted in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer, and H. Schwichtenberg, Springer-Verlag, 1993.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1991. 427 pp.

[MPW89] R. Milner, J. Parrow, and D. Walker. Mobile processes. Technical report, University of Edinburgh, 1989.

[NC95] V. Natarajan and R. Cleaveland. Divergence and fair testing. In *Proceedings of ICALP 95*, pages 648–659. Springer-Verlag, 1995.

[PS88a] P. Panangaden and V. Shanbhogue. Mccarthy's AMB cannot implement fair merge. In *Proceedings of the Eighth FSTTSC Conference*, pages 348–363, 1988. LNCS 338.

[PS88b] P. Panangaden and E. W. Stark. Computations, residuals and the power of indeterminacy. In Timo Lepisto and Arto Salomaa, editors, *Proceedings of the Fifteenth ICALP*, pages 439–454. Springer-Verlag, 1988. Lecture Notes in Computer Science 317.

[San92] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1992.

[Seg92] R. Segala. A process algebraic view of I/O automata. Technical Memo MIT/LCS/TR-557, Massachusetts Institute of Technology, Cambridge, MA 02139, October 1992.

[Sta90] E. W. Stark. On the relations computed by a class of concurrent automata. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 329–340. ACM, 1990.

[Vaa91] F.W. Vaandrager. On the relationship between process algebra and input/output automata. In *Proceedings* 6^{th} *Annual Symposium on Logic in Computer Science*, pages 387–398, 1991.

[Vog92] Walter Vogler. *Modular Construction and Partial Order Semantics of Petri Nets*, volume 625 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992. 252 pp.

[Wal91] D. J. Walker. π-calculus semantics of object oriented languages. In *Proceedings of the conference on Theoretical aspects of computer software*, volume 526 of *Lecture notes in computer science*, 1991.

A Formal definitions

Definition A.1

NIL is defined as the following net L:

$$\begin{array}{rclrclrcl} S_L & = & \{s\} & T_L & = & \{t_\alpha \mid \alpha \in \mathcal{E}? \cup \{f\}\} & & & \\ pre_L(t_\alpha) & = & \{s\} & post_L(t_\alpha) & = & \{s\} & l_L(t_\alpha) & = & \alpha \\ Start_L & = & \{s\} & FS_L & = & \emptyset & & & \end{array}$$

DONE is defined as the following net D:

$$\begin{array}{rclrclrcl} S_D & = & \{s_1, s_2\} & T_D & = & \{t_{\langle\alpha,i\rangle} \mid \alpha \in \mathcal{E}? \cup \{f\}, 1 \le i \le 2\} \cup \{t_{\surd}\} & & & \\ pre_D(t_{\langle\alpha,i\rangle}) & = & \{s_i\} & post_D(t_{\langle\alpha,i\rangle}) & = & \{s_i\} & l_D(t_{\langle\alpha,i\rangle}) & = & \alpha \\ pre_D(t_{\surd}) & = & \{s_1\} & post_D(t_{\surd}) & = & \{s_2\} & l_D(t_{\surd}) & = & \surd \\ Start_D & = & \{s_1\} & FS_D & = & \{s_1\} & & & \end{array}$$

EMIT $e!$ is defined as the following net E:

$$\begin{array}{rclrclrcl} S_E & = & \{s_1, s_2, s_3\} & T_E & = & \{t_{\langle\alpha,i\rangle} \mid \alpha \in \mathcal{E}? \cup \{f\}, 1 \le i \le 3\} \cup \{t_{\surd}, t_{e!}\} & & & \\ pre_E(t_{\langle\alpha,i\rangle}) & = & \{s_i\} & post_E(t_{\langle\alpha,i\rangle}) & = & \{s_i\} & l_E(t_{\langle\alpha,i\rangle}) & = & \alpha \\ pre_E(t_{e!}) & = & \{s_1\} & post_E(t_{e!}) & = & \{s_2\} & l_E(t_{e!}) & = & e! \\ pre_E(t_{\surd}) & = & \{s_2\} & post_E(t_{\surd}) & = & \{s_3\} & l_E(t_{\surd}) & = & \surd \\ Start_E & = & \{s_1\} & FS_E & = & \{s_1, s_2\} & & & \end{array}$$

Definition A.2 Let $\langle S_N, T_N, Start_N, FS_N\rangle$ be an RFT net, and let $e \in \mathcal{E}$. Then $R = e? {\rightarrow} N$ is defined as follows. Let DONE be as defined earlier, let *snew* be a new place and $T_{\text{new}} = \{t_\alpha \mid \alpha \in \mathcal{E}? \cup \{f\}\}$ a set of new transitions, where the transitions are labeled by their subscript, $\{snew\}$ is the preset of all the new transitions, $post_R(t_f) = \{snew\}$, $post_R(t_{e?}) = Start_N$, and $post_R(t_\alpha) = Start_{\texttt{DONE}}$ for all other α. Then $R = \langle S_N \cup \{snew\}, T_N \cup T_{\text{new}}, \{snew\}, FS_N \cup FS_{\texttt{DONE}}\rangle$, where the labels, presets and postsets of all transitions from N stay unchanged,

Definition A.3 Let $N = \langle S_N, T_N, Start_N, FS_N\rangle$ be an RFT net, and let $e \in E$. Then $R = N$ HIDE e is defined as follows. Let *snew* be a new place not in S_N and *tnew* a new transition not in T_N. Then R is defined as:

$$\begin{array}{rclrcll} S_R & = & S_N \cup \{snew\} & T_R & = & \{t \in T_N \mid l_N(t) \neq e?\} \cup \{tnew\} & \\ pre_R(tnew) & = & \{snew\} & post_R(tnew) & = & \{snew\} & \\ pre_R(t) & = & pre_N(t) & post_R(t) & = & post_N(t) & t \in T_R \cap T_N \\ l_R(tnew) & = & e? & l_R(t) & = & \begin{cases} \tau & \text{if } l_N(t) = e! \\ l_N(t) & \text{otherwise} \end{cases} & t \in T_R \cap T_N \\ Start_R & = & Start_N \cup \{snew\} & FS_R & = & FS_N & \end{array}$$

Definition A.4 Let $\langle S_{N_1}, T_{N_1}, Start_{N_1}, FS_{N_1}\rangle$ and $\langle S_{N_2}, T_{N_2}, Start_{N_2}, FS_{N_2}\rangle$ be RFT nets. Then $R = N_1\ ;\ N_2$ is defined as follows. Let *snew* be a new place. Then R is $\langle S_{N_1} \cup S_{N_2} \cup \{snew\}, T_{N_1} \cup T_{N_2}, Start_{N_1} \cup \{snew\}, FS_{N_1} \cup FS_{N_2}\rangle$, where the labels, presets and postsets of all transitions from N_2 stay unchanged. For all $t \in T_R \cap T_N$, $pre_R(t) = pre_N(t) \cup \{snew\}$, and if $l_N(t) \neq \surd$, then $l_R(t) = l_N(t)$ and $post_R(t) = post_N(t) \cup \{snew\}$. If $l_N(t) = \surd$, then $l_R(t) = \tau$ and $post_R(t) = post_N(t) \cup Start_{N_2}$.

Definition A.5 Let $\langle S_N, T_N, Start_N, FS_N\rangle$ be an RFT net, and for all $n \geq 1$, let $\langle snew, n\rangle$ be new places not in S_N. Then $R =$ LOOP N is defined as the following infinite net.

$$
\begin{array}{rcl}
S_R & = & \{\langle s,n\rangle \mid s \in S_N \cup \{\mathit{snew}\} \text{ and } n \geq 1\} \\
T_R & = & \{\langle t,n\rangle \mid t \in T_N \text{ and } n \geq 1\}
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{pre}_R(\langle t,n\rangle) & = & \{\langle s,n\rangle \mid s \in \mathit{pre}_N(t)\} \cup \{\langle \mathit{snew},n\rangle\} \\
l_R(\langle t,n\rangle) & = & \begin{cases} \tau & \text{if } l_N(t) = \surd \\ l_N(t) & \text{otherwise} \end{cases}
\end{array}
$$

$$
\mathit{post}_R(\langle t,n\rangle) = \begin{cases} \{\langle s,n\rangle \mid s \in \mathit{post}_N(t)\} \cup \{\langle \mathit{snew},n\rangle\} & \text{if } l_N(t) \neq \surd \\ \{\langle s,n\rangle \mid s \in \mathit{post}_N(t)\} & \\ \cup\{\langle s,n+1\rangle \mid s \in (\mathit{Start}_N \cup \{\mathit{snew}\})\} & \text{if } l_N(t) = \surd \end{cases}
$$

$$
\begin{array}{rcl}
\mathit{Start}_R & = & \{\langle s,1\rangle \mid s \in \mathit{Start}_N \cup \{\mathit{snew}\}\} \\
FS_R & = & \{\langle s,n\rangle \mid s \in FS_N \text{ and } n \geq 1\}
\end{array}
$$

Definition A.6 Let $\langle S_{N_1}, T_{N_1}, \mathit{Start}_{N_1}, FS_{N_1}\rangle$ and $\langle S_{N_2}, T_{N_2}, \mathit{Start}_{N_2}, FS_{N_2}\rangle$ be RFT nets, let $e \in \mathcal{E}$, let $\mathit{snew}_1, \mathit{snew}_2$ be new places, and let $T_{\text{new}} = \{\mathit{tnew}_\alpha \mid \alpha \in E? \cup \{f\}\} \cup \{t_{e?}\}$ be new transitions. Then $R = \texttt{DO}\ N_1\ \texttt{WATCH}\ e?\ \texttt{TOUT}\ N_2$ is defined as follows.

$$
\begin{array}{rcl}
S_R & = & S_{N_1} \cup S_{N_2} \cup \{\mathit{snew}_1, \mathit{snew}_2\} \\
T_R & = & \{t \in T_{N_1} \mid l_{N_1}(t) \neq e?\} \cup T_{N_2} \cup T_{\text{new}}
\end{array}
$$

$$
\begin{array}{rcl}
l_R(\mathit{tnew}_\alpha) & = & \alpha \\
l_R(t_{e?}) & = & e? \\
l_R(t) & = & l_{N_i}(t) \qquad \text{if } t \in T_R \cap T_{N_i}
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{post}_R(t_{e?}) & = & \mathit{Start}_{N_2} \\
\mathit{post}_R(\mathit{tnew}_\alpha) & = & \{\mathit{snew}_2\} \\
\mathit{post}_R(t) & = & \begin{cases} \mathit{post}_{N_1}(t) \cup \{\mathit{snew}_1\} & \text{if } t \in T_R \cap T_{N_1} \text{ and } l_{N_1}(t) \notin \{e!, \surd\} \\ \mathit{post}_{N_1}(t) \cup \{\mathit{snew}_2\} & \text{if } t \in T_R \cap T_{N_1} \text{ and } l_{N_1}(t) = \surd \\ \mathit{post}_{N_1}(t) \cup \mathit{Start}_{N_2} & \text{if } t \in T_R \cap T_{N_1} \text{ and } l_{N_2}(t) = e! \\ \mathit{post}_{N_2}(t) & \text{if } t \in T_R \cap T_{N_2} \end{cases}
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{pre}_R(\mathit{tnew}_\alpha) & = & \{\mathit{snew}_2\} \qquad\qquad \mathit{pre}_R(t_{e?}) = \{\mathit{snew}_1\} \\
\mathit{pre}_R(t) & = & \begin{cases} \mathit{pre}_{N_1}(t) \cup \{\mathit{snew}_1\} & \text{if } t \in T_R \cap T_{N_1} \\ \mathit{pre}_{N_2}(t) & \text{if } t \in T_R \cap T_{N_2} \end{cases}
\end{array}
$$

$$
\begin{array}{rcl}
\mathit{Start}_R & = & \mathit{Start}_{N_1} \cup \{\mathit{snew}_1\} \\
FS_R & = & FS_{N_1} \cup FS_{N_2}
\end{array}
$$

Definition A.7 Let $\langle S_{N_1}, T_{N_1}, \mathit{Start}_{N_1}, FS_{N_1}\rangle$ be a RFT net, and let $e_1, e_2 \in \mathcal{E}$. Then $R = \texttt{SUSP}\ N_1$ on $e_1?\ \texttt{RES}\ e_2?$ is defined as follows.

Let $\mathit{snew}_1, \mathit{snew}_2, \mathit{snew}_3$ be new places and let $T_{\text{new}} = \{\mathit{tnew}_\alpha \mid \alpha \in E? \cup \{f\} \cup \{\mathit{tnew}'_\alpha \mid \alpha \in E? \cup \{f\} \cup \{\mathit{tnew}''_{e_1}\}$ be new transitions labeled with their subscripts.

- Exactly snew_3 is the preset and postset of all the tnew_α.
- Exactly snew_2 is the preset and postset of all the tnew'_α such that $\alpha \neq e_2?$.
- $\mathit{pre}_R(\mathit{tnew}_{e_2?}) = \{\mathit{snew}_2\}$ and $\mathit{post}_R(\mathit{tnew}_{e_2}) = \{\mathit{snew}_1\}$.
- $\mathit{pre}_R(\mathit{tnew}''_{e_1}) = \{\mathit{snew}_1\}$ and $\mathit{post}_R(\mathit{tnew}''_{e_1}) = \{\mathit{snew}_2\}$.

Then $R = \langle S_N \cup \{snew_1, snew_2, snew_3\}, \{t \in T_N \mid l_N(t) \neq e_1?\} \cup T_{\text{new}}, Start_N \cup \{snew_1\}, FS_N\rangle$, where the labels of all transitions from N stay the same.

For all transitions $t \in T_R \cap T_N$, $pre_R(t) = pre_{N_1}(t) \cup \{snew_1\}$ and

- If $l_N(t) \notin \{e_1!, \surd\}$, then $post_R(t) = post_{N_1}(t) \cup \{snew_1\}$.
- If $l_N(t) = \surd$, then $post_R(t) = post_{N_1}(t) \cup \{snew_3\}$.
- If $l_N(t) = e_1!$, then $post_R(t) = post_{N_1}(t) \cup \{snew_2\}$.

Definition A.8 Let $\langle S_{N_1}, T_{N_1}, Start_{N_1}, FS_{N_1}\rangle$ and $\langle S_{N_2}, T_{N_2}, Start_{N_2}, FS_{N_2}\rangle$ be RFT nets, and let $\{snew_1, snew_2\}$ be new places. Then $R = N_1 \mid\mid N_2$ is defined as follows.

$$
\begin{array}{rcll}
S_R & = & S_{N_1} \cup S_{N_2} & \\
T_R & = & T \cup T' \cup T'' & \\
\\
T & = & \{\langle t_1, t_2\rangle \mid l_{N_1}(t_1) \text{ and } l_{N_2}(t_2) \text{ compatible } \} & \\
l_R(\langle t_1, t_2\rangle) & = & \textit{compatible}\ (l_{N_1}(t_1), l_{N_2}(t_2)) & \\
pre_R(\langle t_1, t_2\rangle) & = & pre_{N_1}(t_1) \cup pre_{N_2}(t_2) & \\
post_R(\langle t_1, t_2\rangle) & = & post_{N_1}(t_1) \cup post_{N_2}(t_2) & \\
\\
T' & = & \{\langle t_{\text{copy}}, i\rangle \mid t \in T_{N_i} \text{ and } l_{N_i} = \surd\} & \\
l_R(\langle t_{\text{copy}}, i\rangle) & = & \tau & \\
pre_R(\langle t_{\text{copy}}, i\rangle) & = & pre_{N_i}(t) \cup \{snew_1\} & \\
post_R(\langle t_{\text{copy}}, i\rangle) & = & post_{N_i}(t) \cup \{snew_2\} & \\
\\
T'' & = & \{t \in T_{N_i} \mid l_{N_i}(t) \in \{\tau, \surd\}\} & \\
l_R(t) & = & l_{N_i}(t) & t \in T'' \cap T_{N_i} \\
pre_R(t) & = & \begin{cases} pre_{N_i}(t) & \text{if } l_{N_i}(t) = \tau\} \\ pre_{N_i}(t) \cup \{snew_2\} & \text{if } l_{N_i}(t) = \surd\} \end{cases} & \\
post_R(t) & = & post_{N_i}(t) & t \in T'' \cap T_{N_i} \\
\\
Start_R & = & Start_{N_1} \cup Start_{N_2} \cup \{snew_1\} & \\
FS_R & = & FS_{N_1} \cup FS_{N_2} &
\end{array}
$$