# Fair Synchronous Transition Systems and their Liveness Proofs *

Amir Pnueli
Dept. of Applied Math. and CS
The Weizmann Institute of Science
Rehovot, ISRAEL

Natarajan Shankar  Eli Singerman
Computer Science Laboratory
SRI International
Menlo Park CA, USA

Technical Report SRI-CSL-98-02

**Abstract**

We present a compositional semantics of synchronous systems that captures both safety and progress properties of such systems. The *fair synchronous transitions systems* (FSTS) model we introduce in this paper extends the basic $\alpha$STS model [KP96] by introducing operations for parallel composition, for the restriction of variables, and by addressing fairness. We introduce a weak fairness (justice) condition which ensures that any communication deadlock in a system can only occur through the need for external synchronization. We present an extended version of linear time temporal logic (ELTL) for expressing and proving safety and liveness properties of synchronous specifications, and provide a sound and compositional proof system for it.

*This research was supported in part by the Minerva Foundation, by an infrastructure grant from the Israeli Ministry of Science, by US National Science Foundation grants CCR-9509931 and CCR-9712383, and by US Air Force Office of Scientific Research Contract No. F49620-95-C0044. Part of this research was done as part of the European Community project SACRES (EP 20897). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF, AFOSR, the European Union, or the U.S. Government. We are grateful to Sam Owre for lending assistance with PVS.

# 1 Introduction

Synchronous languages are rapidly gaining popularity as a high-level programming paradigm for a variety of safety-critical and real-time control applications and for hardware/software co-design. Synchronous languages are used to define systems that are:

- Reactive: Engaged in an ongoing interaction with an environment.
- Event-triggered: System computation is triggered by the arrival and absence of external inputs.
- Concurrent: A system is composed of subsystems that are computing and interacting concurrently.
- Synchronous: The system response to input stimuli is "instantaneous".

The present paper presents a unifying transition system model for synchronous languages based on *fair synchronous transition systems* (FSTS) that can be used as a semantic basis for synchronous languages. Such a unifying semantic model can be used to specify the temporal behavior of synchronous systems and to relate different synchronous system descriptions.

There are two major classes of synchronous languages [H93]. The *imperative* languages like ESTEREL [BG92] and STATECHARTS [Har87], and *declarative* languages like LUSTRE [CHPP87] and SIGNAL [BGJ91]. The language ESTEREL has statements like:

- **present** *signal* **then** *statement* **else** *statement*: Execute **then** statement when *signal* is present, and **else** statement otherwise.
- **emit** *signal* : broadcast signal.
- **trap** *id* **in ... exit** *id* **...**: A catch-throw mechanism for exception signaling and handling.
- **do** *statement* **watching** *signal*: preempt execution of *statement* when *signal* occurs.

These are combined with the usual control constructs like conditional branching and parallel composition.

Declarative languages like LUSTRE and SIGNAL express constraints on signal flows or infinite streams. In LUSTRE, each signal is defined in a mutually recursive way in terms of other signals. LUSTRE applies the various logical and arithmetic operators in a pointwise manner to signals so that `X + Y` is the signal that is the pointwise sum of signals `X` and `Y`. The basic operators for constructing expressions that define signals in LUSTRE are:

- **pre**`(X)` which is nil, $x_0, x_1, \ldots$, where `X` is $x_0, x_1, \ldots$.

- `X -> Y` which is the signal $x_0, y_1, y_2, \ldots$ where `X` is of the form $x_0, x_1, \ldots$, and `Y` is of the form $y_0, y_1, \ldots$.
- `X` **when** `B` which is the signal $\langle x_i \mid b_i \rangle$, i.e., the sequences $x_{i_1}, x_{i_2}, \ldots,$, where $i_1, i_2, \ldots,$ are the positions at which the boolean signal `B` is true. The resulting clock of `X when B` is defined by the boolean signal `B`.
- **current** `X` which speeds up `X` to the next faster clock in terms of which `X` is defined. This is done by padding `X` with the previous $x_i$ wherever `X` is undefined with respect to the faster clock.

For example,

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| `B` | T | F | T | F | F | T | T | F | F ... |
| `X` | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ... |
| `X` **when** `B` | 1 | | 3 | | | 6 | 7 | | ... |
| **current** `(X` **when** `B)` | 1 | 1 | 3 | 3 | 3 | 6 | 7 | 8 | 8 ... |

LUSTRE programs are functional with respect to the input streams. All LUSTRE signals in a program are defined with respect to sub-clocks of a single master clock.

The language SIGNAL is the most liberal of the synchronous languages and is the primary motivation for the fair synchronous transition systems model presented here. SIGNAL is similar to LUSTRE but admits the specification of nondeterministic constraints and the signals are not all derived from a single master clock.

SIGNAL programs may also contain mutually recursive definitions of signals. Apart from the usual pointwise arithmetic, logical, and other data operators, the basic operators in SIGNAL are:

- `X $` **init** $y_0$ which is the signal having the same clock as `X` and the same signal as `X` but delayed by one clock tick and with initial value $y_0$.
- `X` **when** `B` is similar to the corresponding operator in LUSTRE but `X` and `B` can have different clocks. The clock of the resulting signal is the intersection of the two clocks and takes on the value of `X` whenever the boolean signal `B` is true.
- `X` **default** `Y` is the signal whose clock is the union of the clocks of `X` and `Y` and whose value is equal to the value of `X` when `X` is defined, and the value of `Y` otherwise.

SIGNAL programs can be combined by parallel composition `P||Q` which is the conjunction of the equations in `P` and `Q`. Hiding `P/X` is another operation where an externally visible signal `X` in `P` can be made local to a program `P/X`.

The features common to all these synchronous languages are that

- Signals are not persistent and can be present or absent in a computation state.
- State transitions can be governed by the presence of a signal in a state, its value when it is present, and its absence.
- A module specifies constraints on the signal values in each transition.
- Composition yields the conjunction of such constraints since in synchronous systems, the synchronized transitions occur simultaneously.

In this paper, we present a compositional semantics of synchronous systems that captures both safety and progress properties of such systems. The semantics is given in terms of the model of fair synchronous transitions systems (FSTS), which is based on the $\alpha$STS model [KP96]. The $\alpha$STS model has been used as a common semantic domain for both SIGNAL programs and the C-code generated by compiling them, for proving the correctness of the translation (compilation) [PSS98]. It has also been used in [BGA97] as a semantics which is "...fully general to capture the essence of the synchronous paradigm."

The FSTS model presented here extends $\alpha$STS by introducing operations for parallel composition, for the restriction of variables, and by addressing fairness. It can be used to answer questions such as:

- What is the valid set of runs corresponding to a given synchronous specification?
- How can we characterize a set of fair computations corresponding to a given synchronous specification?
- How can linear time temporal logic be adapted to reasoning about fair synchronous specifications?
- What is a sound compositional proof system for proving temporal safety and liveness properties of synchronous specifications?

The key characteristics of the FSTS model that make it suitable for capturing the semantics of synchronous systems are:

- Signals are modeled by variables whose value might be undefined in a state indicating the absence of the signal.
- The absence of a signal is indicated by having the values of variables range over a lifted domain that contains an undefined $\perp$ value in addition to defined data values.
- The variables of a transition system module are partitioned into
  - Synchronization variables that are used for interaction through input and output with other modules.

  - Controlled variables which are entirely controlled by the system consisting of variables that are externally visible and those that are local.

- Transitions can be taken on the basis of whether a signal is present or absent in a state.

- The crucial compositionality constraints on a FSTS module are :

  - A module transition cannot force the synchronization variables to take on defined values. It should always be possible for the system to enter a communication deadlock where the synchronization variables all remain undefined. This is because the module and its environment need to cooperate in order to synchronize on a defined value for a synchronization variable, and it is always possible for the environment to not cooperate with the module.
  - A module transition is invariant with respect to the part of the state that is unobservable by it. The behavior of the module is only affected by another module through the values (defined or undefined) of the synchronization variables that they share.
  - If any signal that is controlled by the module is continuously disabled, i.e., undefined, it must be because its definedness depends on the definedness of some subset of synchronization variables. This restriction is captured by means of a justice (weak fairness) condition that ensures that a variable that is continuously enabled to take on a defined value even when the synchronization variables are deadlocked, does eventually do so. The justice condition requires controlled variables to not deadlock, i.e., remain undefined, unless their definedness depends on the values of synchronization variables.

The FSTS model is designed to be simple and general. It extends the classical notion of transition systems with *signals* that can be present or absent in a given state, communication as *synchronization* by means of signals, *stuttering* as the absence of signals, and local progress through weak fairness constraints. The fairness condition ensures that any communication deadlock in a module can only occur through the need for external synchronization. Except for the fairness condition, the presence or absence of a signal is treated in a symmetrical manner as is the case in synchronous languages. The use of weak fairness constraints ensures that a module can satisfy these constraints without the cooperation of the environment, i.e., the module is *receptive* [AL95].

The FSTS model, the compositionality proofs, the extended linear temporal logic, and the accompanying soundness proofs have all been formally verified using PVS [Ow95]. The PVS verification pointed out a number of gaps in our earlier formalization and led to sharper definitions of the basic concepts, and

elegant and rigorous proofs.[1]

The paper is organized as follows. In Section 2 we introduce the FSTS computational model. In Section 3 we define two important operations on FSTS modules, *parallel composition* and *restriction*, and motivate the definitions by an intuitive example of progress (liveness). In Section 4 we present a formal method for proving temporal properties of synchronous systems by introducing an appropriate logic for expressing these properties and a system of deductive proof rules. We demonstrate the use of these rules by formalizing the intuitive arguments used in the example of Section 3. In Section 5 we present the FSTS semantics of the most expressive synchronous language – SIGNAL. (The FSTS semantics of ESTEREL and LUSTRE can be obtained in a similar way.)

# 2 Fair Synchronous Transition Systems

In this section, we introduce the notion of *fair synchronous transition systems*.

## The Computational Model

We assume a vocabulary $\mathcal{V}$ which is a set of typed variables. Variables which are intended to represent signals are identified as *volatile*, and their domain contains a distinguished element $\perp$ used to denote the absence of the respective signal. In the translation of synchronous programs to FSTS specifications (see Section 5), we shall also use *persistent* variables to simulate the "memorization" operators of the synchronous languages (e.g., **current** in LUSTRE, "$" in SIGNAL).

Some of the types we consider are the *booleans* with domain $\mathcal{B} = \{\mathrm{T}, \mathrm{F}\}$, the type of *integers* whose domain $Z$ consists of all the integers, the type of *pure signals* with domain $\mathrm{S}_\perp = \{\mathrm{T}, \perp\}$, the type of *extended booleans* with domain $\mathcal{B}_\perp = \{\mathrm{T}, \mathrm{F}, \perp\}$, and the type of *extended integers* with domain $\mathrm{Z}_\perp = Z \cup \{\perp\}$.

We define a *state* $s$ to be a type-consistent interpretation of $\mathcal{V}$, assigning to each variable $u \in \mathcal{V}$ a value $s[u]$ over its domain. We denote by $\Sigma$ the set of all states. For a subset of variables $V \subseteq \mathcal{V}$, we define a *$V$-state* to be a type-consistent interpretation of $V$.

Following [MP91] and [KP96], we define a *fair synchronous transition system* (FSTS) to be a system

$$\Phi: \quad \langle V, \Theta, \rho, E, S \rangle,$$

consisting of the following components:

- $V$ : A finite set of typed *system variables*.
- $\Theta$ : The *initial condition*. A satisfiable assertion characterizing the initial states.

[1] The PVS formalization and proofs can be obtained from the URL `www.csl.sri.com/~singermn/fsts/`.

- $\rho$ : A *transition relation*. This is an assertion $\rho(V, V')$, which relates a state $s \in \Sigma$ to its possible successors $s' \in \Sigma$ by referring to both unprimed and primed versions of the system variables. An unprimed version of a system variable refers to its value in $s$, while a primed version of the same variable refers to its value in $s'$. For example, the assertion $x' = x + 1$ states that the value of $x$ in $s'$ is greater by 1 than its value in $s$. If $\rho(s[V], s'[V]) = \text{T}$, we say that state $s'$ is a *$\rho$-successor* of state $s$.

  **Remark:** As implied by the notation $\rho(V, V')$, $\rho$ can only refer to the variables of $\Phi$, and therefore cannot distinguish between two possible $\Phi$-successors that agree on the values of all the variables of $\Phi$. That is, for all $s, s_1, s_2 \in \Sigma$,

  $$s_1|V = s_2|V \ \rightarrow \ (\rho(s, s_1) \Leftrightarrow \rho(s, s_2)) \, .$$

- $E \subseteq V$ : The set of *externally observable variables*. These are the variables that can be observed outside the module. We refer to $L = V \perp E$ as the *local variables*. Local variables cannot be observed outside the module.

- $S \subseteq E$ : The set of *synchronization variables*. These are the signal variables on which the module may (and needs to) synchronize with its environment. We refer to $C = V \perp S$ as the *controllable variables*. These are the variables on whose values the module has full control.

For states $s, s' \in \Sigma$ and assertion $\varphi(V, V')$, we say that $\varphi$ holds over the *joint interpretation* $\langle s, s' \rangle$, denoted

$$\langle s, s' \rangle \models \varphi,$$

if $\varphi(V, V')$ evaluates to T over the interpretation which interprets every $x \in V$ as $s[x]$ and every $x'$ as $s'[x]$.

For an FSTS $\Phi$, a state $s'$ is called a *$\Phi$-successor* of the state $s$ if $\langle s, s' \rangle \models \rho$.

**Definition 1** *An* FSTS $\Phi$ *is called* realizable *if every state $s \in \Sigma$ has a $\Phi$-successor in which all the synchronization variables assume the value $\perp$. This requirement expresses the possibility that the environment might not be ready to co-operate with the module $\Phi$ in the current state.*

From now on, we restrict our attention to realizable FSTS specifications.

## Computations of an FSTS

Let $\Phi = \langle V, \Theta, \rho, E, S \rangle$ be an FSTS. A *computation* of $\Phi$ is an infinite sequence

$$\sigma: \quad s_0, s_1, s_2, \ldots ,$$

where, for each $i = 0, 1, \ldots$, $s_i \in \Sigma$, and $\sigma$ satisfies the following requirements:

- *Initiation*: $s_0$ is initial, i.e., $s_0 \models \Theta$.
- *Consecution*: For each $j = 0, 1, \ldots,\ s_{j+1}$ is a $\Phi$-successor of $s_j$.
- *Justice* (weak fairness): We say that a signal variable $x \in C$ is *enabled with respect to* $S$ at state $s_j$ of $\sigma$ if there exists a $\Phi$-successor $s$ of $s_j$, such that $s[x] \neq \perp \wedge s[S] = \{\perp\}$.

  The *justice* requirement is that for every signal variable $x \in C$, it is not the case that $x$ is enabled w.r.t. $S$ in all but a finite number of states along $\sigma$, while $s_j[x] \neq \perp$, for only finitely many states $s_j$ in the computation.

**Remark:** In the sequel, we shall sometimes use the term "enabled w.r.t. $\Phi$" instead of "enabled w.r.t. $S_\Phi$".

The requirement of justice with respect to a controllable signal variable $x$ demands that if $x$ is continuously enabled to assume a non-$\perp$ value, from a certain point on, without the need to synchronize with the environment, it will eventually assume such a value. The fact that a variable $x$ is enabled to become non-$\perp$ without 'external assistance' is evident from the existence of a $\Phi$-successor $s$ such that $s[x] \neq \perp \wedge s[S] = \{\perp\}$.

A *run* of a system $\Phi$ is a finite or an infinite sequence of states satisfying the requirements of initiation and consecution, but not necessarily the requirement of justice.

An FSTS is called *viable* if every finite run can be extended into a computation. From now on, we restrict our attention to viable FSTS specifications.

For the case that $\Phi$ is a finite-state system, i.e., all system variables range over finite domains, there exist (reasonably) efficient symbolic model-checking algorithms for testing whether $\Phi$ is viable.

A state $s'$ is a *stuttering variant* of a state $s$ if all volatile (i.e., signal) variables are undefined in $s'$ and $s'$ agrees with $s$ on all persistent variables. A state sequence $\widehat{\sigma}$ is said to be a *stuttering variant* of the state sequence $\sigma : s_0, s_1, \ldots$, if $\widehat{\sigma}$ can be obtained from $\sigma$ by repeated transformations where a state $s$ in $\sigma$ is replaced with $s, s'$ in $\widehat{\sigma}$, or a pair of adjacent states $s, s'$ in $\sigma$ is replaced with $s$ in $\widehat{\sigma}$ where $s'$ is a stuttering variant of $s$. A set of state sequences $S$ is called *closed under stuttering* if, for every state sequence $\sigma$, $\sigma \in S$ iff all stuttering variants of $\sigma$ belong to $S$. An FSTS $\Phi$ is called *stuttering robust* if the set of $\Phi$-computations is closed under stuttering.

# 3 Operations on FSTS Modules

There are two important operations on FSTS modules: *parallel composition* and *restriction*.

## 3.1 Parallel Composition

Let $\Phi_1$: $\langle V_1, \Theta_1, \rho_1, E_1, S_1\rangle$ and $\Phi_2$: $\langle V_2, \Theta_2, \rho_2, E_2, S_2\rangle$ be two FSTS modules. These systems are called *syntactically compatible* if they satisfy

$$C_1 \cap V_2 = V_1 \cap C_2 = \emptyset \text{ (or equivalently } V_1 \cap V_2 = S_1 \cap S_2).$$

That is, only synchronization variables can be common to both systems. We define the *parallel composition* of syntactically compatible $\Phi_1$ and $\Phi_2$, denoted $\Phi = \Phi_1 \parallel \Phi_2$, to be the FSTS $\Phi$: $\langle V, \Theta, \rho, E, S\rangle$, where

$$\begin{array}{rcl} V & = & V_1 \cup V_2 \\ \Theta & = & \Theta_1 \wedge \Theta_2 \\ \rho & = & \rho_1 \wedge \rho_2 \\ E & = & E_1 \cup E_2 \\ S & = & S_1 \cup S_2 \end{array}$$

To indicate the relation between computations of a composed system and computations of its constituents, we first prove the following lemma:

**Lemma 1** *Let $\Phi_1$: $\langle V_1, \Theta_1, \rho_1, E_1, S_1\rangle$ and $\Phi_2$: $\langle V_2, \Theta_2, \rho_2, E_2, S_2\rangle$ be two compatible* FSTS *modules, $x$ be a controlled signal variable of $\Phi_1$, and $\sigma$: $s_0, s_1, s_2, \ldots$ be a sequence of states. For every $j \geq 0$:*

$$x \text{ is enabled in } s_j \text{ w.r.t. } \Phi_1 \iff x \text{ is enabled in } s_j \text{ w.r.t. } \Phi_1 \parallel \Phi_2.$$

**Proof:** ($\Longrightarrow$) First, note that since $x$ is a controlled signal variable of $\Phi_1$, it is also a controlled signal variable of $\Phi_1 \parallel \Phi_2$. Now, assume that $s$ is a $\Phi_1$-successor of $s_j$, s.t. $s[x] \neq \perp$ and $s[S_1] = \perp$. By the realizability requirement applied to $\Phi_2$ (see Definition 1), there exists a state $s'$, s.t. $\rho_2(s, s')$ and $s'[S_2] = \perp$. Let $s''$ be the state that valuates each variable of $\Phi_1$ as $s$ and all other variables as $s'$. Since $\Phi_1$ and $\Phi_2$ are syntactically compatible, we have $s''[x] \neq \perp$ and $s''[S_1 \cup S_2] = \perp$. Hence, by recalling that $\rho_1$ and $\rho_2$ can only refer to variables in $V_1$ and $V_2$, respectively, we have that $\rho_1(s, s'')$ and $\rho_2(s, s'')$; i.e., $s''$ is a $\Phi_1 \parallel \Phi_2$-successor of $s_j$, and we are done.

The argument in the other direction is similar. ▟

We can now prove the following theorem:

**Theorem 2** *Let $\Phi_1$: $\langle V_1, \Theta_1, \rho_1, E_1, S_1\rangle$ and $\Phi_2$: $\langle V_2, \Theta_2, \rho_2, E_2, S_2\rangle$ be two compatible* FSTS *modules, and $\sigma$: $s_0, s_1, s_2, \ldots$ be a sequence of states.*

*$\sigma$ is a computation of $\Phi_1 \parallel \Phi_2$ $\iff$ $\sigma$ is both a computation of $\Phi_1$ and of $\Phi_2$.*

**Proof:** ($\Longrightarrow$) Due to symmetry, it suffices to prove that $\sigma$ is a computation of $\Phi_1$. The fact that $\sigma$ satisfies the initiation and consecution requirements for $\Phi_1$ follows easily from the definition of $\Phi_1 \parallel \Phi_2$. What remains is to show that $\sigma$ is fair (just) w.r.t. $\Phi_1$. Let $x$ be a controlled variable of $\Phi_1$ which is continuously

enabled w.r.t. $\Phi_1$. We have to prove that $x \neq \bot$ infinitely often. By Lemma 1, we conclude that $x$ is continuously enabled w.r.t. $\Phi_1 \parallel \Phi_2$. So, by the assumption that $\sigma$ is a computation of $\Phi_1 \parallel \Phi_2$, and is therefore just w.r.t. $\Phi_1 \parallel \Phi_2$, we have $x \neq \bot$ infinitely often.

($\Longleftarrow$) Again, it is easy to see that $\sigma$ satisfies the the initiation and consecution requirements for $\Phi_1 \parallel \Phi_2$. To prove that $\sigma$ is fair w.r.t. $\Phi_1 \parallel \Phi_2$, let $x$ be a controlled variable of $\Phi_1 \parallel \Phi_2$ which is continuously enabled w.r.t. $\Phi_1 \parallel \Phi_2$. By the definition of $\Phi_1 \parallel \Phi_2$, $x \in (V_1 \setminus S_1) \cup (V_2 \setminus S_2)$. Without loss of generality, assume $x \in (V_1 \setminus S_1)$, i.e., $x$ is a controlled variable of $\Phi_1$. Using Lemma 1, we see that $x$ is continuously enabled w.r.t. $\Phi_1$, so that by the fact that $\sigma$ is fair w.r.t. to $\Phi_1$, we have $x \neq \bot$ infinitely often. ▟

## 3.2 Restriction

In the operation of restriction, we identify a set of synchronization variables and close off the system for external synchronization on these variables.

Let $W \subseteq S$ be a set of synchronization variables of the FSTS $\Phi$: $\langle V, \Theta, \rho, E, S \rangle$. We define the *$W$-restriction* of $\Phi$, denoted $[\mathbf{own}\ W \colon \Phi]$, to be the FSTS $\widetilde{\Phi}$: $\langle \widetilde{V}, \widetilde{\Theta}, \widetilde{\rho}, \widetilde{E}, \widetilde{S} \rangle$, where

$$\begin{array}{rcl} \widetilde{V} & = & V \\ \widetilde{\Theta} & = & \Theta \\ \widetilde{\rho} & = & \rho \\ \widetilde{E} & = & E \\ \widetilde{S} & = & S \setminus W. \end{array}$$

Thus the effect of $W$-restricting the system $\Phi$ amounts to moving the variables in $W$ from $S$ to $C$. This movement may have a considerable effect on the computations of the system.

**Example 1** Consider the FSTS $\Phi$, where

$$\begin{array}{rcl} V = E = S & : & \{x : \{\mathrm{T}, \bot\}\} \\ \Theta & : & x = \mathrm{T} \\ \rho & : & x' = \mathrm{T} \ \vee \ x' = \bot. \end{array}$$

Let $\widetilde{\Phi}$: $[\mathbf{own}\ x.\ \Phi]$ denote the $x$-restriction of $\Phi$. The sequence below, in which next to each state appears the set of all possible successors,

$$\begin{array}{lll} \sigma\colon & s_0\colon \langle x\colon \mathrm{T} \rangle, & \{\langle x\colon \bot \rangle,\ \langle x\colon \mathrm{T} \rangle\} \\ & s_1\colon \langle x\colon \bot \rangle, & \{\langle x\colon \bot \rangle,\ \langle x\colon \mathrm{T} \rangle\} \\ & s_2\colon \langle x\colon \bot \rangle, & \ldots \end{array}$$

is a computation of $\Phi$ but is not a computation of $\widetilde{\Phi}$. Note that $x$ assumes a non-$\bot$ value only at $s_0$. The sequence $\sigma$ is a computation of $\Phi$ since $x$ is not in

$C$ ($x \in S$), and therefore is not enabled w.r.t. $S$ in any state of $\sigma$. On the other hand, $x$ is no longer a synchronization variable in $\widetilde{\Phi}$, since it belongs to $\widetilde{C}$. The sequence $\sigma$ is not a computation of $\widetilde{\Phi}$ since it is unfair to $x$. This is because now, $x$ is enabled w.r.t. $S_{\widetilde{\Phi}}$ in every state of $\sigma$, but $s[x] \neq \bot$ only in $s_0$. From this we can deduce that all computations of $\widetilde{\Phi}$ contain infinitely many states in which $x \neq \bot$, but this is not necessarily the case for computations of $\Phi$.

The following lemma (whose proof is left for the final version) describes the relation between computations of unrestricted and restricted systems.

**Lemma 3** *The infinite sequence*

$$\sigma: \quad s_0, s_1, s_2, \ldots$$

*is a computation of* [**own** $W.\,\Phi$] *iff*

1. *$\sigma$ is a computation of $\Phi$, and*
2. *$\sigma$ satisfies the justice requirement w.r.t. $S \setminus W$.*

Thus, the computations of [**own** $W.\,\Phi$] can be obtained from those of $\Phi$.

**Example 2** Consider the FSTS $\Phi_1$ defined by

$$\begin{array}{rcl} V_1 = E_1 & : & \{x, y : \mathrm{Z}_\bot\} \\ S_1 & : & \{x\} \\ \Theta_1 & : & x = y = \bot \\ \rho_1 & : & (y' = 3) \wedge (x' = 4) \quad \vee \quad (y' = \bot) \wedge (x' \neq 4) \end{array}$$

and the FSTS $\Phi_2$ defined by

$$\begin{array}{rcl} V_2 = E_2 = S_2 & : & x : \mathrm{Z}_\bot \\ \Theta_2 & : & x = \bot \\ \rho_2 & : & (x' = 4) \vee (x' = 5) \vee (x' = \bot). \end{array}$$

Both of these FSTS modules have computations satisfying $\Box \Diamond (y = 3)$. There exists, however, a computation of both $\Phi_1$ and $\Phi_2$ which by Theorem 2 is therefore also a computation of $\Phi_1 \parallel \Phi_2$, which does not satisfy $\Box \Diamond (y = 3)$. This computation is

$$\begin{array}{lll} \sigma: & s_0: \langle \overset{x}{\bot}, \overset{y}{\bot} \rangle, & \{\langle \bot, \bot \rangle, \langle 4, 3 \rangle, \langle 5, \bot \rangle\} \\ & s_1: \langle \bot, \bot \rangle, & \{\langle \bot, \bot \rangle, \langle 4, 3 \rangle, \langle 5, \bot \rangle\} \\ & s_2: \langle \bot, \bot \rangle, & \ldots \end{array}$$

In $\sigma$, the variable $y$ does not get the value 3 even once. This is fair, since $y$ is not enabled w.r.t. $S_{\Phi_1 \| \Phi_2}$ in any state of $\sigma$. This is because $y' \neq \bot \wedge x' = \bot$ is

false in every state of $\sigma$. Now, suppose we close off $\Phi$ with respect to $x$, to get the FSTS module

$$\widetilde{\Phi}: \quad [\mathbf{own}\ x.\,[\Phi_1 \parallel \Phi_2]].$$

In $\widetilde{\Phi}$, $x$ is no longer a synchronization variable, and therefore $y$ is continuously enabled w.r.t. $S^{\sim}_{\Phi}$ in $\sigma$ (in every state, in fact). However, it is obviously not the case that $y \overset{\sim}{\neq} \perp$ is satisfied infinitely often in $\sigma$, and hence $\sigma$ is not a computation of $\widetilde{\Phi}$. In conclusion, we see that only the restriction of $x$ guarantees $\Box \Diamond (y = 3)$.

# 4 Compositional Verification of FSTS Modules

In this section we show how to construct compositional verification of the temporal properties of FSTS's, concentrating on liveness properties.
First, we define an appropriate logic. Let ELTL be LTL extended with the unary predicate *ready*. An ELTL *model* is a pair $M = (L, \sigma)$, where $\sigma$ is an infinite sequences of the form

$$\sigma: \quad s_0, s_1, s_2, \ldots, \text{ where } s_j \in \Sigma, \text{ for every } j \geq 0,$$

and $L : \Sigma \to 2^{\Sigma}$. ELTL formulas are interpreted as follows.

- For a state formula $p$, $(M, j) \models p \ \leftrightarrow\ s_j \models p$.
- $(M, j) \models \neg p \ \leftrightarrow\ (M, j) \not\models p$.
- $(M, j) \models p \vee q \ \leftrightarrow\ (M, j) \models p$ or $(M, j) \models q$.
- $(M, j) \models \bigcirc p \ \leftrightarrow\ (M, j + 1) \models p$.
- $(M, j) \models p \mathcal{U} q \ \leftrightarrow\ (M, k) \models q$ for some $k \geq j$ and $(M, i) \models p$ for all $i$, $j \leq i < k$.
- For a state formula $p$, $(M, j) \models \mathit{ready}(p) \ \leftrightarrow\ \exists s \in L(s_j)$ s.t. $s \models p$.

As usual, we use the abbreviations $\Diamond p$ for $\mathrm{T} \mathcal{U} p$ and $\Box p$ for $\neg \Diamond \neg p$.

We say that a model $M$ *satisfies* an ELTL formula $p$, written $M \models p$, if $(M, 0) \models p$.

**Notation:** For an ELTL model $M = (L, \sigma)$, we denote by $\sigma(i)$, the $i + 1$-th element of $\sigma$.

For an FSTS $\Phi$, the ELTL model $M = (L, \sigma)$ is called a $\Phi$-*model*, if

1. $\sigma$ is a computation of $\Phi$, and

| | |
|---|---|
| 1. | $\Phi_1 \models \diamondsuit\,\Box\, ready(z_1 = c_1 \wedge \ldots \wedge z_n = c_n)$, where $\{z_1, \ldots, z_n\} \supseteq S_{\Phi_1} \cap S_{\Phi_2}$ |
| 2. | $\Phi_2 \models \diamondsuit\,\Box\, ready(z_1 = c_1 \wedge \ldots \wedge z_n = c_n)$ |
| | $\Phi_1 \parallel \Phi_2 \models \diamondsuit\,\Box\, ready(z_1 = c_1 \wedge \ldots \wedge z_n = c_n)$ |

Figure 1: Rule READY.

2. For every $j = 0, 1, 2, \ldots$, $L(\sigma(j))$ is the set of all possible $\Phi$-successors of $\sigma(j)$, i.e., $L(\sigma(j)) = \text{image}(\rho, \{\sigma(j)\})$.

We write $\Phi \models p$, and say that $p$ *is valid over* $\Phi$, if $M \models p$, for every $\Phi$-*model* $M$.

**Definition 2** *An* ELTL *formula is called* universal*, if it does not contain* ready *or if each occurrence of* ready *appears under an* odd *number of negations.*

We present our method by formalizing the intuitive arguments used in Example 2. In the proof, we introduce several deductive rules that we believe may be useful in typical liveness proofs. The soundness of these rules is proved in the sequel.

Let us begin by noting that

$$\Phi_1,\ \Phi_2 \models \Box\, ready(y = 3 \wedge x = 4), \tag{1}$$

can be verified independently for $\Phi_1$ and for $\Phi_2$. From (1) and the temporal tautology $q \rightarrow \diamondsuit\, q$, we can derive

$$\Phi_1,\ \Phi_2 \models \diamondsuit\,\Box\, ready(y = 3 \wedge x = 4). \tag{2}$$

Applying rule READY, presented in Fig. 1, to (2) yields

$$\Phi_1 \parallel \Phi_2 \models \diamondsuit\,\Box\, ready(y = 3 \wedge x = 4). \tag{3}$$

From the latter, we can easily derive

$$\Phi_1 \parallel \Phi_2 \models \diamondsuit\,\Box\, ready(y \neq \bot). \tag{4}$$

By applying rule OWN (Fig. 2) to (4), with $W = \{x\}$, we get

$$\underbrace{[\mathbf{own}\ x.\,[\Phi_1 \parallel \Phi_2]]}_{\Phi} \models \diamondsuit\,\Box\, ready(y \neq \bot). \tag{5}$$

Now, since $S_\Phi = \emptyset$, we can use axiom CONT (Fig. 3), and (5) with $z = y$, to derive

$$\Phi \models \Box\,\diamondsuit(y \neq \bot). \tag{6}$$

$$\frac{\Phi \models p, \text{ where } p \in \text{ELTL}}{[\mathbf{own}\ W.\,\Phi] \models p}$$

Figure 2: Rule OWN.

$$\Phi \models \diamondsuit\,\Box\, \mathit{ready}(z \neq \bot \wedge S_\Phi = \bot) \rightarrow \Box\,\diamondsuit(z \neq \bot), \text{ where } z \in C_\Phi$$

Figure 3: Axiom CONT.

It is not difficult to prove

$$\Phi_1 \models \Box(y = 3 \vee y = \bot). \tag{7}$$

By applying rule COMP (Fig. 4) to the latter, we get

$$\Phi_1 \parallel \Phi_2 \models \Box(y = 3 \vee y = \bot). \tag{8}$$

We now apply rule OWN (Fig. 2) to (8), to get

$$\Phi \models \Box(y = 3 \vee y = \bot). \tag{9}$$

The latter, together with (6) implies $\Phi \models \Box\,\diamondsuit(y = 3)$ which completes the proof.

Now, as promised, we prove the soundness of the deductive rules. We start with two lemmas that follow from Theorem 2, and characterize the relation between ELTL models of a composed system and those of its constituents.

**Lemma 4** *Let $\Phi_1$ and $\Phi_2$ be two compatible fsts modules, $M_1 = (L_1, \sigma)$ be a $\Phi_1$-model, and $M_1 = (L_2, \sigma)$ be a $\Phi_2$-model. Then*

*$(L_\cap, \sigma)$ is a $\Phi_1 \parallel \Phi_2$ -model, where $L_\cap(s) = L_1(s) \cap L_2(s)$, for every $s \in \Sigma$.*

**Lemma 5** *Let $\Phi_1$ and $\Phi_2$ be two compatible fsts modules, and $M = (L, \sigma)$ be a $\Phi_1 \parallel \Phi_2$-model. Then*

*$(L_{\Phi_1}, \sigma)$ is a $\Phi_1$ -model, where $L_{\Phi_1}(s) = \mathit{image}(\rho_{\Phi_1}, \{s\})$, for every $s \in \Sigma$.*

**Theorem 6** *The rule* READY *is sound. That is, let $\Phi_1$: $\langle V_1, \Theta_1, \rho_1, E_1, S_1\rangle$ and $\Phi_2$: $\langle V_2, \Theta_2, \rho_2, E_2, S_2\rangle$ be two compatible* FSTS*'s, and $\{z_1, \ldots, z_n\}$ be a set of*

$$\frac{\Phi_1 \models p, \text{ where } p \in \text{ELTL is universal}}{\Phi_1 \parallel \Phi_2 \models p}$$

Figure 4: Rule COMP.

*variables s.t.* $\{z_1, \ldots, z_n\} \supseteq S_1 \cap S_2$*:*

$$\begin{array}{l}\big(\Phi_1 \models \diamondsuit\,\square\,\text{ready}(z_1 = c_1 \wedge \ldots \wedge z_n = c_n)) \wedge \\ (\Phi_2 \models \diamondsuit\,\square\,\text{ready}(z_1 = c_1 \wedge \ldots \wedge z_n = c_n))\big) \;\rightarrow \\ (\Phi_1 \parallel \Phi_2 \models \diamondsuit\,\square\,\text{ready}(z_1 = c_1 \wedge \ldots \wedge z_n = c_n)).\end{array}$$

**Proof:** [(sketch)] Let $M = (L, \sigma)$ be a $\Phi_1 \parallel \Phi_2$-model. Use Lemma 5 to construct a $\Phi_1$-model, $M_1 = (L_{\Phi_1}, \sigma)$, and a $\Phi_2$-model, $M_2 = (L_{\Phi_2}, \sigma)$. By the assumption,

$$\begin{array}{lcl} M_1 & \models & \diamondsuit\,\square\, \mathit{ready}(z_1 = c_1 \wedge \ldots \wedge z_n = c_n) \text{ and} \\ M_2 & \models & \diamondsuit\,\square\, \mathit{ready}(z_1 = c_1 \wedge \ldots \wedge z_n = c_n). \end{array}$$

Hence, there exists an $i_0$ s.t. for all $j \geq i_0$,

$$\begin{array}{lcl} (M_1, j) & \models & \mathit{ready}(z_1 = c_1 \wedge \ldots \wedge z_n = c_n) \text{ and} \\ (M_2, j) & \models & \mathit{ready}(z_1 = c_1 \wedge \ldots \wedge z_n = c_n). \end{array}$$

Note that

$$\{z_1, \ldots, z_n\} \supseteq S_1 \cap S_2 \supseteq V_1 \cap V_2,$$

so that using the realizability requirement (see Def. 1), it is not too difficult to prove that for every $j \geq i_0$:

$$(((L_{\Phi_1} \cap L_{\Phi_2}, \sigma), j) \models \mathit{ready}(z_1 = c_1 \wedge \ldots \wedge z_n = c_n).$$

That is

$$M \models \diamondsuit\,\square\, \mathit{ready}(z_1 = c_1 \wedge \ldots \wedge z_n = c_n).$$

▟

**Theorem 7** *The rule* OWN *(Fig. 2) is sound. That is, let* $\Phi$*:* $\langle V, \Theta, \rho, E, S\rangle$ *be an* FSTS*,* $W \subseteq S$ *be a set of variables, and* $p$ *be an* ELTL *formula. Then*

$$(\Phi \models p) \;\rightarrow\; ([\textbf{own}\ W.\, \Phi] \models p)\,.$$

**Proof:** Let $M_{\text{own}} = (L, \sigma)$ be a [**own** $W.\,\Phi$]-model. First, note that $\sigma$ is a computation of [**own** $W.\,\Phi$], so by Lemma 3 it is also a computation of $\Phi$. Next, recall that for every $j \geq 0$, $L(\sigma(j))$ is the set of all possible [**own** $W.\,\Phi$]-successors of $\sigma(j)$; but since $\Phi$ and [**own** $W.\,\Phi$] have the same transition relation, $L(\sigma(j))$ is also the set of all possible $\Phi$-successors of $\sigma(j)$. Thus, $M_{\text{own}}$ is a $\Phi$-model, and by the assumption $M_{\text{own}} \models p$. ▟

**Theorem 8** *The axiom* CONT *(Fig. 3) is valid. That is, let* $\Phi$: $\langle V, \Theta, \rho, E, S\rangle$ *be an* FSTS*, and* $z$ *be a controlled variable of* $\Phi$*. Then*

$$\Phi \models \Diamond\Box\,\text{ready}(z \neq \bot \wedge S = \bot) \;\rightarrow\; \Box\Diamond(z \neq \bot).$$

**Proof:** (sketch) Suppose that $M = (L, \sigma)$ is an $\Phi$-model, and $z \in C_\Phi$. $\sigma$ is a computation of $\Phi$, and is therefore fair w.r.t. $z$. The observant reader would notice that

$$\Diamond\Box\,\mathit{ready}(z \neq \bot \wedge S = \bot) \;\rightarrow\; \Box\Diamond(z \neq \bot)$$

is simply a reformulation of the justice requirement w.r.t. $z$. From this it is not difficult to complete the proof. ▟

To prove the soundness of the COMP, we need some preparation.

**Definition 3** *An* ELTL *formula is called* existential*, if it does not contain* ready *or if each occurrence of* ready *appears under an* even *number of negations.*

**Lemma 9** *Let* $M_1 = (\sigma, L_1)$ *and* $M_2 = (\sigma, L_2)$ *be two* ELTL *models, s.t.* $L_2(\sigma(j)) \subseteq L_1(\sigma(j))$*, for each* $j = 0, 1, 2, \ldots$*. Then, for every* ELTL *formula* $p$*, and every* $i = 0, 1, 2, \ldots$*:*

1. $((M_1, i) \models p \wedge (p \textit{ is universal}) \;\rightarrow\; (M_2, i) \models p)$*, and*
2. $((M_2, i) \models p \wedge (p \textit{ is existential}) \;\rightarrow\; (M_1, i) \models p)$.

**Proof:** [(sketch)] The proof is carried out by mutual induction on the structure of $p$. Let us only mention that for the $p = \neg(q)$ case, observe that if $\neg(q)$ is universal then $q$ is existential, and if $\neg(q)$ is existential then $q$ is universal. Hence, (1) follows from the induction hypothesis for (2), and vice-versa. ▟

**Theorem 10** *The rule* COMP *(Fig. 4) is sound. That is, let* $\Phi_1$ *and* $\Phi_2$ *be two compatible fsts modules, and* $p$ *be a universal* ELTL *formula. Then*

$$(\Phi \models p) \;\;\rightarrow\;\; (\Phi_1 \parallel \Phi_2 \models p).$$

**Proof:** Suppose $M_\parallel = (L, \sigma)$ is a $\Phi_1 \parallel \Phi_2$-model. By Lemma 5, the ELTL model $M_1 = (L_\Phi, \sigma)$, where $L_{\Phi_1}(s) = \text{image}(\rho_{\Phi_1}, \{s\})$, for every $s \in \Sigma$, is a $\Phi_1$-model. So, by the assumption, $M_1 \models p$. Now, for every state $s \in \Sigma$:

$$L(s) \;=\; \text{image}(\rho_{\Phi_1 \parallel \Phi_2}, \{s\}) \;\subseteq\; \text{image}(\rho_{\Phi_1}, \{s\}) \;=\; L_{\Phi_1}(s).$$

Hence, recall that $p$ is universal, so that by using Lemma 9 (with $M_2 = M_{\|}$ and $i = 0$), we can conclude $M_{\|} \models p$. ∎

We conclude with a practical remark. Verification of FSTS specification can be done by using existing symbolic model-checking algorithms. Computing the $L$-sets comes at no extra cost, since the predicate $\mathit{ready}(p)$ is equivalent to the CTL formula $EXp$ which every model checker knows how to compute very efficiently.

# 5 The FSTS semantics of SIGNAL

As mentioned in the introduction, the FSTS model is a significant extension of the previous, more basic, $\alpha$STS model [KP96] obtained by introducing operations (parallel composition and restriction) and by addressing fairness. The translation from SIGNAL programs to corresponding FSTS specifications, however, is not affected by these extensions, and can be carried out exactly as with $\alpha$STS simply by taking the input/output variables as the *externally observable* variables and also as the *synchronization* variables. Nevertheless, to make the paper more self-contained, we present below the translation given by Kesten and Pnueli [KP96].

For a variable $v$, $\mathit{clocked}(v)$ denotes the assertion:

$$\mathit{clocked}(v) : v \neq \perp$$

In the following, we describe how to construct an FSTS $\Phi_P$ corresponding to a given SIGNAL program $P$.

## System Variables

The system variables of $\Phi$ are given by $V = U \cup X$, where $U$ are the SIGNAL variables explicitly declared and manipulated in $P$, and $X$ is a set of auxiliary variables. An auxiliary variable by the name of $x.v$ is included in $X$ for each expression of the form $v$ \$ appearing in $P$. For simplicity, we assume that the \$ operator is only applied to variables and not to more general expressions. The value of $x.v$ is intended to represent the value of $v$ at the previous instance (present excluded) that $v$ was different from $\perp$.

## Externally observable and synchronization variables

The externally observable variables $E$ and also the synchronization variables $S$ are those explicitly declared in $P$ as input/output variables.

## Initial Condition

The initial condition for $\Phi$ is given by

$$\Theta\colon \quad \bigwedge_{u \in U} u = \bot \quad \wedge \quad \bigwedge_{x.v \in U} x.v = \bot$$

As will result from our FSTS translation of SIGNAL programs, they are all stuttering robust. Consequently, we can simplify things by assuming that the first state in each run of the system is a stuttering state.

## Transition Relation

The transition relation $\rho$ will be a conjunction of assertions, where each SIGNAL statement gives rise to a conjunct in $\rho$.

We list the statements of SIGNAL and, for each statement $S$, we present the conjunct contributed to $\rho$ by $S$.

## Basic Instructions

- Consider the SIGNAL statement $y := f(v_1, \ldots, v_n)$, where $f$ is a state-function. Its contribution to $\rho$ is given by:

$$\begin{array}{ll} & \mathit{clocked}(y') \equiv \mathit{clocked}(v'_1) \equiv \ldots \equiv \mathit{clocked}(v'_n) \\ \wedge & (\mathit{clocked}(y') \rightarrow y' = f(v'_1, \ldots, v'_n)) \end{array}$$

  This formula requires that the signals $y, v_1, \ldots, v_n$ are present at precisely the same time instants, and that at these instants $y = f(v_1, \ldots, v_n)$.

- The contribution of the statement

$$y := v\$ \ \ \textbf{init}\ v_0$$

  is given by:

$$\begin{array}{lll} & x'.v & = \ \textbf{if}\ \mathit{clocked}(v')\ \textbf{then}\ v'\ \textbf{else}\ x.v \\ \wedge & y' & = \left( \begin{array}{llll} \textbf{if} & \neg \mathit{clocked}(v') & \textbf{then} & \bot \\ \textbf{else if} & x.v = \bot & \textbf{then} & v_0 \\ \textbf{else} & x.v & & \end{array} \right) \end{array}$$

  The first conjunct of this formula defines the new value of $x.v$. If the new value of $v$ is different from $\bot$, then the new value of $x.v$ is the new value of $v$. Otherwise, $x.v$ retains its current value.

  The second conjunct of the formula defines the new value of $y$ by considering three cases. The first case requires that $y' = \bot$ whenever $v' = \bot$. This together with the other two cases implies that the clocks of $v$ and

$y$ are identical. The second considered case is the first position at which $v' \neq \bot$. Observe that the fact that we are at the *else* clause of the test $\neg \mathit{clocked}(v')$ implies that $v' \neq \bot$, and that $x.v = \bot$ implies that there was no previous position at which $v \neq \bot$. In this case, we take the new value of $y$ ($y'$) to be $v_0$. The last case considers subsequent positions at which $v' \neq \bot$. At all of these positions, $y'$ is taken to be the value of $x.v$, i.e., the value of $v$ as it were at the previous ($v' \neq \bot$)-position and as memorized in $x.v$.

- The contribution of the statement

$$y := v \textbf{ when } b$$

is given by:

$$y' = \textbf{ if } (b' = \mathrm{T}) \textbf{ then } v' \textbf{ else } \bot.$$

- The contribution of the statement

$$y := u \textbf{ default } v$$

is given by:

$$y' = \textbf{ if } \mathit{clocked}(u') \textbf{ then } u' \textbf{ else } v'.$$

## Shorthand Instructions

- The contribution of the statement

$$\varphi(\textbf{synchro } v, y),$$

which states that $v$ and $y$ have the same clock, is given by:

$$\mathit{clocked}(v') \equiv \mathit{clocked}(y')$$

- The contribution of the statement

$$y := \textbf{when}(v),$$

which is an abbreviation for: $y := v$ when $v$, for a boolean variable $v$, is given by:

$$y' = \textbf{ if } (v' = \mathrm{T}) \textbf{ then } \mathrm{T} \textbf{ else } \bot.$$

- The contribution of the statement

$$y := \textbf{ event}(v),$$

which defines $y$ to be a pure signal representing the clock of $v$, is given by:

$$y' = \textbf{ if } \mathit{clocked}(v') \textbf{ then } \mathrm{T} \textbf{ else } \bot.$$

GUARD_COUNT {**input event** *fill*, **output boolean** *empty*}

=

**synchro** (**when**$(zn = 0)$), *fill*

| $n :=$ (10 **when** *fill*) **default** $(zn \perp 1)$

| $zn :=$ $n$ \$ **init** 0

| *empty* := **when**$(n = 0)$ **default** (**not** *fill*)

Figure 5: A sample SIGNAL program.

## Example

In Fig. 5, we present a SIGNAL program example, taken from [BGJ91]. We simplified the program to make it more self-contained. This small program models a system with a replenishable resource, for example, a water reservoir. The input event *fill* signals that that the reservoir is filled to the top. The local integer variable $n$ measures the current water level. At each *fill* signal, the level is set to 10 (assumed maximal capacity). Then the level gradually decreases until it reaches 0. The output signal *empty* will register T when the water level drops to 0, and will register F when the reservoir is next filled.

## The program as an FSTS

The FSTS translation of the SIGNAL program of Fig. 5 is defined as follows:

The system variables are given by:

$$V: \quad \{\underbrace{\mathit{fill}, \mathit{empty}, zn, n}_{U}, \underbrace{x.n}_{X}\}.$$

The externally observable and synchronization variables are given by:

$$E = S: \quad \{\mathit{fill}, \mathit{empty}\}.$$

The initial condition is given by:

$$\Theta: \quad \mathit{fill} = \mathit{empty} = zn = n = x.n = \perp.$$

The transition relation $\rho$ is given by:

$$
\begin{array}{lrcl}
 & (zn' = 0) & \equiv & clocked(fill') \\
\wedge & n' & = & \left( \begin{array}{llll} \textbf{if} & fill' = \mathrm{T} & \textbf{then} & 10 \\ \textbf{else if} & clocked(zn') & \textbf{then} & zn' \perp 1 \\ \textbf{else} & \perp & & \end{array} \right) \\
\wedge & x.n' & = & \textbf{if } clocked(n') \textbf{ then } n' \textbf{ else } x.n \\
\wedge & zn' & = & \left( \begin{array}{llll} \textbf{if} & \neg clocked(n') & \textbf{then} & \perp \\ \textbf{else if} & x.n = \perp & \textbf{then} & 0 \\ \textbf{else} & x.n & & \end{array} \right) \\
\wedge & empty' & = & \left( \begin{array}{llll} \textbf{if} & n' = 0 & \textbf{then} & \mathrm{T} \\ \textbf{else if} & fill' = \mathrm{T} & \textbf{then} & \mathrm{F} \\ \textbf{else} & \perp & & \end{array} \right)
\end{array}
$$

# 6 Conclusions and Future Work

We have presented FSTS, a compositional semantics of synchronous systems that captures both safety and progress properties. We have motivated the fairness requirement and the operations of parallel composition and of restriction of variables by means of intuitive examples.

We have then introduced an extended version of linear temporal logic (ELTL), in which it is convenient to express safety and liveness properties of synchronous specifications, and have presented (and demonstrated) a sound compositional proof system for it.

We have concluded by specifying how to translate programs written in an expressive representative of the synchronous school, namely SIGNAL, to FSTS.

Directions in future work which we intend to pursue are

- Specifying in detail the FSTS semantics of LUSTRE, ESTEREL and STATECHARTS.
- Apply the deductive proof system developed here together with existing symbolic model-checking algorithms to the verification of FSTS specifications that result from actual synchronous programs.

# References


[AL95] M. Abadi and L. Lamport. Conjoining Specifications. *TOPLAS*, 17(3), pages 507–534, 1995.

[BGA97] A. Benveniste, P. Le Guernic, and P. Aubry. Compositionality in dataflow synchronous languages: specification & code generation. *Proceedings of COMPOS'97.*

[BGJ91] A. Benveniste, P. Le Guernic, and C. Jacquemot. Synchronous programming with event and relations: the SIGNAL language and its semantics. *Science of Computer Programming*, 16, pages 103–149, 1991.

[BG92] G. Berry and G. Gonthier. The ESTEREL Synchronous Programming Language: Design, semantics, implementation. *Science of Computer Programming*, 19(2), 1992.

[CHPP87] P. Caspi, N. Halbwachs, D. Pilaud, and J. Plaice. LUSTRE, a Declarative Language for Programming Synchronous Systems. *POPL'87*, ACM Press, pages 178–188, 1987.

[H93] N. Halbwachs. *Synchronous Programming of Reactive Systems*. Kluwer, Dordrecht, The Netherlands, 1993.

[Har87] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8, pages 231–274, 1987.

[KP96] Y. Kesten and A. Pnueli. An $\alpha$STS-based common semantics for SIGNAL and STATECHARTs, March 1996. Sacres Manuscript.

[MP91] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1991.

[Ow95] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS. IEEE trans. on software eng., 21(2), pages 107–125, 1995.

[PSS98] A. Pnueli, M. Siegel, and E. Singerman. Translation Validation. *TACAS'98*, LNCS, 1998.