Fair Synchronous Transition Systems and their Liveness Proofs *

A. Pnueli[1] N. Shankar[2] E. Singerman[2]

[1] Weizmann Institute of Science, Rehovot, Israel
[2] SRI International, Menlo Park, California, USA

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 αSTS 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.

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".

* This research was supported in part by the Minerva Foundation, by an infrastructure grant from the Israeli Ministry of Sciences and Art, 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.

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 present paper presents a unifying compositional 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. The semantics is given in terms of the model of fair synchronous transitions systems (FSTS), which is based on the αSTS model [KP96]. The αSTS 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) [PSiS98]. 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 αSTS 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 of synchronous specification?
- How can linear time temporal logic be adapted to synchronous specifications?
- What is a sound compositional proof system for proving temporal safety and liveness properties of synchronous specifications?

The FSTS model is designed to be a simple and general model. 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

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

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 these rules by formalizing the intuitive arguments used in the example of Section 3. In Section 5 we outline 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 Σ 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

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

consisting of the following components:

- V : A finite set of typed *system variables*.
- Θ : The *initial condition*. A satisfiable assertion characterizing the initial states.
- ρ : 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 $\rho(s[V], s'[V]) = \mathrm{T}$, we say that state s' is a *ρ-successor* of state s.

Remark: As implied by the notation $\rho(V, V')$, ρ can only refer to the variables of Φ, and therefore cannot distinguish between two possible Φ-successors that agree on the values of all the variables of Φ. 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 $L = V - E$ 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 $C = V - S$ 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 φ 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 Φ, a state s' is called a *Φ-successor* of the state s if $\langle s, s' \rangle \models \rho$.

Definition 1. *An* FSTS *Φ is called* realizable *if every state $s \in \Sigma$ has a Φ-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 Φ 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 Φ 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 σ 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 Φ-successor of s_j.
- *Justice*: We say that a signal variable $x \in C$ is *enabled with respect to S* at state s_j of σ if there exists a Φ-successor s of s_j, s.t. $s \models (x \neq \perp \wedge S = \perp)$. 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 σ, 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. Φ" instead of "enabled w.r.t. S_Φ".

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 Φ-successor s such that $s[x] \neq \bot \wedge S = \bot$.

A *run* of a system Φ 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 Φ 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 Φ 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 σ by repeated transformations where a state s in σ is replaced with s, s' in $\widehat{\sigma}$, or a pair of adjacent states s, s' in σ 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 σ, $\sigma \in S$ iff all stuttering variants of σ belong to S. An FSTS Φ is called *stuttering robust* if the set of Φ-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 Φ_1: $\langle V_1, \Theta_1, \rho_1, E_1, S_1 \rangle$ and Φ_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

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

That is, only synchronization variables can be common to both systems. We define the *parallel composition* of syntactically compatible Φ_1 and Φ_2, denoted $\Phi = \Phi_1 \parallel \Phi_2$, to be the FSTS Φ: $\langle V, \Theta, \rho, E, S \rangle$, where

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

The following theorem (whose proof can be found in [PShS98]) indicates the relation between computations of a composed system and computations of its constituents.

Theorem 1. *Let* Φ_1: $\langle V_1, \Theta_1, \rho_1, E_1, S_1\rangle$ *and* Φ_2: $\langle V_2, \Theta_2, \rho_2, E_2, S_2\rangle$ *be two compatible* FSTS *modules, and* σ: $s_0, s_1, s_2, \ldots$ *a sequence of states.*

σ *is a computation of* $\Phi_1 \parallel \Phi_2 \iff \sigma$ *is both a computation of* Φ_1 *and of* Φ_2.

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 Φ: $\langle V, \Theta, \rho, E, S\rangle$. We define the *$W$-restriction* of Φ, denoted $[\mathbf{own}\, W : \Phi]$, to be the FSTS $\widetilde{\Phi}$: $\langle \widetilde{V}, \widetilde{\Theta}, \widetilde{\rho}, \widetilde{E}, \widetilde{S}\rangle$, where $\widetilde{V} = V$, $\widetilde{\Theta} = \Theta$, $\widetilde{\rho} = \rho$, $\widetilde{E} = E$, and $\widetilde{S} = S \setminus W$.

Thus the effect of W-restricting the system Φ 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 Φ, where

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

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

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

is a computation of Φ but is not a computation of $\widetilde{\Phi}$. Note that x assumes a non-$\bot$ value only at s_0. The sequence σ is a computation of Φ since x is not in C ($x \in S$), and therefore is not enabled w.r.t. S in any state of σ. On the other hand, x is no longer a synchronization variable in $\widetilde{\Phi}$, since it belongs to $\widetilde{C}$. The sequence σ 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 σ, 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 Φ.

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

Lemma 1. *The infinite sequence* σ: $s_0, s_1, s_2, \ldots$ *is a computation of* $[\mathbf{own}\, W.\Phi]$ *iff*

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

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

Example 2 Consider the FSTS Φ_1 defined by

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

and the FSTS Φ_2 defined by

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

Both of these FSTS modules have computations satisfying $\Box\Diamond(y = 3)$. There exists, however, a computation of both Φ_1 and Φ_1 which by Theorem 1 is therefore also a computation of $\Phi_1 \parallel \Phi_2$, which does not satisfy $\Box\Diamond(y = 3)$. This computation is

$$\begin{aligned} \sigma\colon \quad & s_0\colon \langle \overset{x}{\perp}, \overset{y}{\perp} \rangle, \ \{\langle \perp, \perp \rangle, \langle 4, 3 \rangle, \ \langle 5, \perp \rangle\} \\ & s_1\colon \langle \perp, \perp \rangle, \ \{\langle \perp, \perp \rangle, \langle 4, 3 \rangle, \ \langle 5, \perp \rangle\} \\ & s_2\colon \langle \perp, \perp \rangle, \qquad\qquad \ldots \end{aligned}$$

In σ, 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 σ. This is because $y' \neq \perp \wedge x' = \perp$ is false in every state of σ. Now, suppose we close off Φ with respect to x, to get the FSTS module $\widetilde{\Phi}$: [**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. $S_{\widetilde{\Phi}}$ in σ (in every state, in fact). However, it is obviously not the case that $y \neq \perp$ is satisfied infinitely often in σ, and hence σ 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 temporal logic. Let ELTL be LTL extended with the unary predicate *ready*. An ELTL *model* is a pair $M = (L, \sigma)$, where σ is an infinite sequences of the form

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

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 $M = (L, \sigma)$, where σ is not given explicitly, we denote by $\sigma(i)$, the $i+1$-th element of σ.

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

1. σ is a computation of Φ, and
2. For every $j = 0, 1, 2, \ldots,\ L(\sigma(j))$ is the set of all possible Φ-successors of $\sigma(j)$, i.e., $L(\sigma(j)) = \mathrm{image}(\rho, \{\sigma(j)\})$.

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

Let us begin, by noting that

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

can be verified independently for Φ_1 and for Φ_2. From (1) and the temporal tautology $q \rightarrow \Diamond q$, we can derive

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

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

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

From the latter, we can easily derive

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

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

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

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

$$\frac{\begin{array}{l}1.\ \Phi_1 \models \Diamond\Box \mathit{ready}(z_1 = c_1 \wedge \ldots \wedge z_n = c_n),\ \text{where } \{z_1, \ldots, z_n\} \supseteq S_{\Phi_1} \cap S_{\Phi_2} \\ 2.\ \Phi_2 \models \Diamond\Box \mathit{ready}(z_1 = c_1 \wedge \ldots \wedge z_n = c_n)\end{array}}{\Phi_1 \parallel \Phi_2 \models \Diamond\Box \mathit{ready}(z_1 = c_1 \wedge \ldots \wedge z_n = c_n)}$$

Fig. 1. Rule READY.

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

Fig. 2. Rule OWN.

derive

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

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 $\Phi \models \Box\Diamond(y = 3)$ which completes the proof.

The soundness of the deductive rules we have introduced is stated in the following theorem whose proof is omitted due to lack of space. A detailed proof can be found in [PShS98].

Theorem 2. *The rules* READY, OWN, CONT *and* COMP *are sound.*

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 these are just the possible successors, so they are computed while the model is constructed. The predicate $\mathit{ready}(p)$ can be computed symbolically as $\mathit{ready}(p)$: $\exists V'.\rho(V, V') \wedge p'$.

5 The FSTS semantics of SIGNAL

As mentioned in the introduction, the FSTS model is a significant extension of the previous, more basic αSTS model [KP96] obtained by introducing operations

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

Fig. 3. Axiom CONT.

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

Fig. 4. Rule COMP.

(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 αSTS 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 sketch below an outline of the translation given by Kesten and Pnueli [KP96].

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

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

System Variables

The system variables of Φ 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. The value of $x.v$ is intended to represent the value of v at the previous instance (present excluded) that v was different from $\bot$.

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 Φ 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 ρ will be a conjunction of assertions, where each SIGNAL statement gives rise to a conjunct in ρ. We list below several representative statements of SIGNAL and, for each statement S, we present the conjunct contributed to ρ by S. For a complete discussion see [KP96] and [PShS98].

- Consider the SIGNAL statement $y := f(v_1, \ldots, v_n)$, where f is a state-function. This statement requires that the signals $y, v_1, \ldots, v_n$ are present at precisely the same time instants, and that at these instants $y = f(v_1, \ldots, v_n)$. Its contribution to ρ is given by:

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

- The contribution of $y := v\$$ **init** v_0, stating that the value of y is that of v shifted by one time unit (initialized with v_0) and that both variables have the same clock, is given by:

$$\begin{array}{rl} & x'.v = \textbf{if } \mathit{clocked}(v') \textbf{ then } v' \textbf{ else } x.v \\ \wedge\;\; y' \;=\; & \begin{pmatrix} \textbf{if} & \neg\mathit{clocked}(v') & \textbf{then } \bot \\ \textbf{else if} & x.v = \bot & \textbf{then } v_0 \\ \textbf{else} & x.v & \end{pmatrix} \end{array}$$

- The contribution of $y := v$ **when** b, stating that y is present and has the value of v only when v is present and b is true, is given by:

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

- The contribution of $y := u$ **default** v, stating that the value of y is that of v if v is present, or else it is the value of v if v is present and otherwise it is $\bot$, is given by:

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

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 αSTS-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.

[PSiS98] A. Pnueli, M. Siegel, and E. Singerman. Translation Validation. *TACAS'98*, LNCS 1384, pages 151–166, 1998.

[PShS98] A. Pnueli, N. Shankar, and E. Singerman. Fair Synchronous Transition Systems and their Liveness Proofs. Technical Report SRI-CSL-98-02, http: www.csl.sri.com ????, 1998.