Front. Comput. Sci., 2013, 7(5): 673–693
DOI 10.1007/s11704-013-3908-2

REVIEW ARTICLE

A comparative study of two formal semantics of the SIGNAL language

Zhibin YANG (✉)[1,2], Jean-Paul BODEVEIX (✉)[2], Mamoun FILALI (✉)[2]

1 School of Computer Science and Engineering, Beihang University, Beijing 100191, China
2 IRIT-CNRS, Université de Toulouse, Toulouse 31062, France

Abstract SIGNAL is a part of the synchronous languages family, which are broadly used in the design of safety-critical real-time systems such as avionics, space systems, and nuclear power plants. There exist several semantics for SIGNAL, such as denotational semantics based on traces (called trace semantics), denotational semantics based on tags (called tagged model semantics), operational semantics presented by structural style through an inductive definition of the set of possible transitions, operational semantics defined by synchronous transition systems (STS), etc. However, there is little research about the equivalence between these semantics.

In this work, we would like to prove the equivalence between the trace semantics and the tagged model semantics, to get a determined and precise semantics of the SIGNAL language. These two semantics have several different definitions respectively, we select appropriate ones and mechanize them in the Coq platform, the Coq expressions of the abstract syntax of SIGNAL and the two semantics domains, i.e., the trace model and the tagged model, are also given. The distance between these two semantics discourages a direct proof of equivalence. Instead, we transform them to an intermediate model, which mixes the features of both the trace semantics and the tagged model semantics. Finally, we get a determined and precise semantics of SIGNAL.

Keywords synchronous language, SIGNAL, trace semantics, tagged model semantics, semantics equivalence, Coq

Received February 20, 2013; accepted June 3, 2013

E-mail: {Zhibin.Yang, bodeveix, filali}@irit.fr

1 Introduction

Safety-critical real-time systems such as avionics, space systems, and nuclear power plants, are also considered as reactive systems [1], because they always interact with their environments continuously. The environment can be some physical devices to be controlled, a human operator, or other reactive systems. These systems receive from the environment input events, and compute the output information, which are finally returned to the environment. The arrival time of events may be different, and the computation needs time. Synchronous method is an important choice to design these systems, which relies on the synchronous hypothesis [2]. Firstly, the computation time is abstracted as zero, that lets system behaviors be divided into a discrete sequence of instants. At each instant, the system does input-computation-output, which takes zero time. Secondly, the different arrival time of events are abstracted as the relative order between events. Even of the physical time is abstracted, the inherent functional properties are not changed, so we can say this method focuses on functional behaviors at a platform-independent level.

There are several synchronous languages, such as ESTEREL [3], LUSTRE [4], SIGNAL [5], and QUARTZ [6]. Synchronous languages can be considered as different implementations of the synchronous hypothesis. As a main difference from other synchronous languages, SIGNAL naturally considers a mathematical time model, in term of a partial order relation, to describe multi-clocked systems without the necessity of a global clock. This feature permits the description of globally asynchronous locally synchronous systems (GALS) [7, 8] conveniently.

There exist several semantics for SIGNAL, such as denotational semantics based on traces (called trace semantics) [9–11], denotational semantics based on tags which are elements of a partially ordered dense set (called tagged model semantics) [10, 12], operational semantics presented by structural style through an inductive definition of the set of possible transitions [5, 10], operational semantics defined by synchronous transition systems (STS) [13]. The differences between the trace semantics and the tagged model semantics are: logical time is represented by a totally ordered set (the set of natural integers **N**) or a partially ordered set; absence of events is explicitly specified (by the $\perp$ symbol) or not. Additionally, Nowak proposes a co-inductive semantics for modeling SIGNAL in the Coq proof assistant [14, 15]. However, there is little research about the equivalence between these semantics. The trace semantics and the tagged model semantics are more commonly used, so we would like to prove the equivalence between them, to get a determined and precise semantics of the SIGNAL language.

The rest of the paper is organized as follows. Section 2 introduces the basic concepts of the SIGNAL language. The abstract syntax of SIGNAL and its Coq expression is given in Section 3. Section 4 presents the definitions of the two semantics domains, i.e., the trace model and the tagged model. Section 5 gives the two formal semantics and their Coq specifications. The proof of the semantics equivalence is presented in Section 6. Section 7 discusses the related work, and Section 8 gives some concluding remarks.

2 An introduction to SIGNAL

Signals As declared in the synchronous hypothesis, the behaviors of a reactive system are divided into a discrete sequence of instants. At each instant, the system does input-computation-output, which takes zero time. So, the inputs and outputs are sequences of values, each value of the sequence being present at some instants. Such a sequence is called a signal. Consequently, at each instant, a signal may be present or absent (denoted by $\perp$). In SIGNAL, signals must be declared before being used, with an identifer (i.e., signal variable or the name of signal) and an associated type for their values such as integer, real, complex, boolean, event, string, etc.

Example 1 Three signals named input_1, input_2, output are shown as follows:

$$\begin{array}{lllll} \text{input}_1 & 1 & \perp & 3 & \perp \cdots \\ \text{input}_2 & \perp & 5 & 7 & 9 \cdots \\ \text{output} & \perp & \perp & 10 & \perp \cdots \end{array}$$

Abstract clock The set of instants where a signal takes a value is the abstract clock of the signal. Two signals are synchronous if they are always present or absent at the same instants, which means they have the same abstract clock.

In the example given above, the abstract clock of input_1, input_2 and output, denoted respectively $\hat{}\text{input}_1$, $\hat{}\text{input}_2$ and $\hat{}\text{output}$, are defined by different set of logical instants.

Moreover, SIGNAL can specify the relations between the abstract clocks of signals in two ways: implicitly or explicitly.

Primitive constructs SIGNAL uses several primitive constructs to express the relations between signals, including relations between values and relations between abstract clocks. Moreover, the primitive constructs can be classified into two families: monoclock operators (for which all signals involved have the same abstract clock) and multiclock operators (for which the signals involved may have different clocks).

- Monoclock operators, including instantaneous function and delay. The instantaneous function $x := f(x_1, x_2, \ldots, x_n)$ applied on a set of inputs $x_1, x_2, \ldots, x_n$ will produce the output x, while the delay operator $x := x_1\ \$\ \text{init}\ c$ sends a previous value of the input to the output with an initial value c.
- Multiclock operators, including undersampling and deterministic merging. The undersampling operator $x := x_1\ \text{when}\ x_2$ is used to check the output of an input at the true occurrence of another input, while the deterministic merging operator $x := x_1\ \text{default}\ x_2$ is used to select between two inputs to be sent as the output, with a higher priority to the first input.

Notice that, these operators specify the relations between the abstract clocks of the signals in an implicit way.

In the SIGNAL language, the relations between values and the relations between abstract clocks, of the signals, are defined as equations, and a process consists of a set of equations. Two basic operators apply to processes, the first one is the composition of different processes, and the other one is the local declaration in which the scope of a signal is restricted to a process.

Example 2 Let us consider a simple process Count [12].

It accepts an input signal reset and delivers the integer output signal val. The local variable counter is initialized to 0 and stores the previous value of the signal val. When an input reset occurs, the signal val is reset to 0. Otherwise, the signal val takes an increment of the variable counter. The process ParallelCount is the composition of two Count processes. Here, the program is not deterministic.

```
process ParallelCount = (! integer x1, x2; )
(| x1 := Count(r)
 | x2 := Count(r)
 |) where event r;
  process Count = (? event reset; ! integer val; )
  (| counter := val $1 init 0
   | val := (0 when reset) default (counter + 1)
   |) where integer counter;
   end;
 end;
```

Extended constructs SIGNAL also provides some operators to express control-related properties by specifying clock relations explicitly, such as clock synchronization, set operators on clocks (union, intersection, difference) and clock comparison.

- Clock synchronization, the equation x_1 ^= x_2 ^= $\cdots$ ^=x_n specifies that signals $x_1, x_2, \ldots, x_n$ are synchronous.
- Set operators on clocks, such as the equation x:= x_1 ^ + x_2 defines the clock of x as the union of the clocks of signals x_1 and x_2, the equation x:= x_1 ^ * x_2 defines the clock of x as the intersection of the clocks of signals x_1 and x_2, the equation x := x_1 ^ − x_2 defines the clock of x as the difference of the clocks of signals x_1 and x_2.
- Clock comparison, such as the statement x_1 ^ < x_2 specifies a set inclusion relation between the clocks of signals x_1 and x_2, the statement x_1 ^ > x_2 specifies a set containment relation between the clocks of signals x_1 and x_2.

3 Abstract syntax of SIGNAL and its Coq expression

In this section, we first give a brief introduction of the theorem prover Coq, then, we give the abstract syntax of SIGNAL and its Coq expression.

3.1 A brief introduction of Coq

Coq [16] is a theorem prover based on the calculus of inductive constructions which is a variant of type theory, following the "Curry-Howard Isomorphism" paradigm, enriched with support for inductive and co-inductive definitions of data types and predicates. From the specification perspective, Coq offers a rich specification language to define problems and state theorems. From the proof perspective, proofs are developed interactively using tactics, which can reduce the workload of the users. Moreover, the type-checking performed by Coq is the key point of proof verification.

Here, we try to give an intuitive introduction to the Coq terminologies which are used in this paper. In the spirit of "Curry-Howard Isomorphism" paradigm, types may represent programming data-types or logical propositions. So, the Coq objects used in this paper can be sorted into two categories: the Type sort and the Prop sort:

- Type is the sort for data types and mathematical structures, i.e., well-formed types or structures are of type Type. Data types can be basic types such as nat, bool, nat → nat, etc., and can be inductive structures, record and co-inductive structures (for infinite objects, as for example infinite sequences). We use Fixpoint and CoFixpoint definitions to define functions over inductive and to co-inductive data types.
- Prop is the sort for propositions, i.e., well-formed propositions are of type Prop. We can define new predicates using inductive, record (for conjunctions of properties) or co-inductive definitions.

3.2 The abstract syntax of SIGNAL

The semantics of each of the extended constructs is defined in term of the primitive constructs, so we just consider the primitive constructs, that is core-SIGNAL. Its abstract syntax is presented as follows.

$$
\begin{array}{lll}
P ::= & x := f(x_1, x_2, \ldots, x_n) & \text{instantaneous function} \\
 & |x := x_1 \ \$ \ \text{init}\ c & \text{delay} \\
 & |x := x_1 \ \text{when}\ x_2 & \text{undersampling} \\
 & |x := x_1 \ \text{default}\ x_2 & \text{deterministic merging} \\
 & |P|P' & \text{composition} \\
 & |P/x & \text{local declaration}
\end{array}
$$

To express more complex SIGNAL programs, all the right-side signal variables of the equations can be replaced by an expression on signal variables.

Here we give the Coq expression of the abstract syntax of SIGNAL. It is parameterized by the set XVar of signal variables, and the set Value of values that can be taken by the variables. isTrue checks that a value is considered to be true. *mkBool* is used to coerce Bool(s) to Value(s). The type Process is defined using five constructors corresponding to the constructs of the core-SIGNAL. We give a very abstract expression of an instantaneous function. The function Pass takes three parameters: a function f of type ((Index → Value) → Value) having an indexed set of input parameters, a variable name of type XVar which contains the left-side variable and an indexed set of variable names of type (Index → XVar) which denotes the actual parameters of f. Index, for example $1, 2, \ldots, n$, represents a set used to index the parameters. Similarly, Pdelay, Pwhen, Pdefault, and Ppar build the corresponding SIGNAL constructs. However, the local declaration is ignored, to get a simplest criterion for the proof of semantics equivalence (see Section 5 and Section 6).

Parameter XVar : **Type**.
Parameter Value : **Type**.
Parameter isTrue : Value → **Prop**.
Parameter mkBool : Bool → Value.
Inductive Process : **Type** :=
Pass : ∀ Index, ((Index → Valuse) → Value)
→ XVar → (Index → XVar) → Process
| Pdefault : XVar → XVar → XVar → Process
| Pwhen : XVar → XVar → XVar → Process
| Pdelay : XVar → XVar → Value → Process
| Ppar : Process → Process → Process.

4 Semantics domains

Semantics domains such as the trace model and the tagged model are introduced in this section. To avoid confusion, we will treat signal variables and signals (sequence of values) separately. The naming convention is given as follows:

- $\{ x, x_1, x_2, \ldots, x_n, y, \ldots \}$ are signal variables.
- $\{ v, v_1, v_2, \ldots, v_n, vv, c, \ldots \}$ are values, and c represents a constant value.
- $\{ s, s_1, s_2, \ldots, s_n, \ldots \}$ are signals.
- $\{ i, i_1, i_2, \ldots, i_n, j, k, \ldots \}$ are indexes.
- $\{ \mathrm{tr}, \mathrm{tr}_1, \mathrm{tr}_2, \ldots, \mathrm{tr}_n, \mathrm{tr}', \mathrm{trs}, \ldots \}$ are traces.
- $\{ t, t_0, t_1, \ldots, t_n, tt, \ldots \}$ are tags.
- $\{ b, b_1, b_2, \ldots, b_n, b', tb, \ldots \}$ are the behaviors on tag structures.

The SIGNAL language specifies a system behavior as a platform-independent model at first. However, it is finally needed to guarantee a correct physical implementation from it (i.e., need to deal with physical time). A formal support for allowing time scalability in design is given in the modeling environment Polychrony [17] by the so-called stretch-closure property. This property can be defined both on the trace model and on the tagged model.

4.1 Trace model

Let $\boldsymbol{X}$ be a set of signal variables, and let $\boldsymbol{V}$ be the set of values that can be taken by the variables. The symbol $\perp$ ($\perp \notin \boldsymbol{V}$) is introduced to express the absence of valuation of a variable. Then we denote:

$$\boldsymbol{V}^{\perp} = \boldsymbol{V} \cup \{\perp\}$$

The corresponding Coq expression is given as follows:
Inductive EValue : **Type** :=
Val : Value → EValue
| Absence : EValue.

Definition 1 (VSignal) [10] A signal s is a sequence $(s_i)_{i \in I}$ of typed values (of $\boldsymbol{V}^{\perp}$), where I is the set of natural integers $\mathbf{N}$ or an initial segment of $\mathbf{N}$, including the empty segment.

A signal can be finite. However, we can extend the finite signal with infinite absences, to get an infinite one. So, in the Coq expression, a signal is defined as an infinite object.
CoInductive VSignal : **Type** :=
Vs : EValue → VSignal → VSignal.

In the following paragraphs, the definition of traces is given. Notice that, a signal is just a sequence of values corresponding to a signal variable, while a trace defines the synchronized sequences of values of a set of signal variables.

Definition 2 (Event) [9] Considering X a non-empty subset of $\boldsymbol{X}$, we call event on X any application

$$e : \boldsymbol{X} \to \boldsymbol{V}_X^{\perp}$$

- $e(x) = \perp$ indicates that variable x has no value in the event.
- $e(x) = v$ indicates, for $v \in \boldsymbol{V}_x$, that variable x takes the value v in the event.

The absent event on $X(X \to \{\perp\})$, where all the signals are absent at a logical instant, is denoted $\perp_e(X)$. Moreover, the set of events on $X(X \to \boldsymbol{V}_X^{\perp})$ is denoted $\boldsymbol{\varepsilon}_X$.

A trace is a sequence of events. For any subset X of $\boldsymbol{X}$, we consider the following definition of the set $\boldsymbol{\tau}_X$ of traces on X.

Definition 3 (Traces) $\boldsymbol{\tau}_X$ is the set of traces on $\boldsymbol{X}$, defined as the set of applications $\mathbf{N} \to \varepsilon_X$ where $\mathbf{N}$ is the set of natural integers.

The absent trace on $X(\mathbf{N} \to \{\perp_e(X)\})$, i.e., the infinite sequence formed by the infinite repetition of $\perp_e(X)$, is denoted $\perp_X$.

Similarly, a trace can be finite. However, we can extend the finite sequence with infinite absent events, to get an infinite trace.

Example 3 Let us consider the following equation: $x_3 := x_1 * x_2$. The set of signal variables is $X = \{x_1,\ x_2,\ x_3\}$. A possible trace is given as follow:

$$\begin{array}{lllllll} x_1 & \perp & 3 & 3 & \perp & \perp & 0\cdots \\ x_2 & \perp & 5 & 7 & \perp & \perp & 9\cdots \\ x_3 & \perp & 15 & 21 & \perp & \perp & 0\cdots \end{array}$$

The trace can be seen as a sequence of events:

$$\{e_0 : \begin{pmatrix} x_1 \mapsto \perp \\ x_2 \mapsto \perp \\ x_3 \mapsto \perp \end{pmatrix}, e_1 : \begin{pmatrix} x_1 \mapsto 3 \\ x_2 \mapsto 5 \\ x_3 \mapsto 15 \end{pmatrix}, \ldots\}$$

The Coq expression of the definition of traces is given as follows.

CoInductive Trace : **Type** :=
  Tr : (XVar → EValue) → Trace → Trace.

As mentioned before, the set of instants where a signal takes a value is the abstract clock of the signal. Its Coq expression is given as follows.

CoFixpoint AClock (x : XVar)(tr : Trace)
  : VSignal :=
  match tr **with**
    Tr st tr′ ⇒
      match tr **with**
        Absence ⇒ Vs Absence (AClock x tr′)
        |_ ⇒ Vs (Val (mkBool true))
              (AClock x tr′)
      end
  end.

Definition 4 (Sprocess) Given a SIGNAL process, its trace semantics, denoted as Sprocess, includes a set of signal variables defining the domain of the process and a set of traces.

The Coq expression is given as follows:

Record Sprocess : **Type** := {
  sdom : XVar → **Prop**;
  straces : Trace → **Prop**
}.

Additionally, we give the definition of the stretch-closure property on the trace model as the definition of compression of a trace given in [18]. The intuition is to consider a trace as an elastic with ordered marks on it. If it is stretched, the marks remain in the same order but have more space (time) between each other by adding columns of $\perp$ (see Fig. 1). The same holds for a set of traces (a behavior), so stretching gives rise to an equivalence between behaviors (stretch equivalence).

$$\begin{array}{lllllll} x_1 & \perp & 3 & 3 & \perp & \perp & 0 & \cdots \\ x_2 & \perp & 5 & 7 & \perp & \perp & 0 & \cdots \\ x_3 & \perp & 15 & 21 & \perp & \perp & 0 & \cdots \end{array}$$

$$\big\downarrow f$$

$$\begin{array}{llllllllll} x_1 & \perp & \perp & 3 & \perp & 3 & \perp & \perp & \perp & 0 & \cdots \\ x_2 & \perp & \perp & 5 & \perp & 7 & \perp & \perp & \perp & 9 & \cdots \\ x_3 & \perp & \perp & 15 & \perp & 21 & \perp & \perp & \perp & 0 & \cdots \end{array}$$

Fig. 1 Stretching of a trace following f

Definition 5 (Stretching) For a given subset X of $\boldsymbol{X}$, a trace tr_1 is less stretched than another trace tr_2, noted $\mathrm{tr}_1 \leqslant \tau_X \mathrm{tr}_2$, iff there exists a mapping $f : \mathbf{N} \to \mathbf{N}$ such as:

- $\forall x \in X\ \forall i \in \mathbf{N}, \mathrm{tr}_2(f(i))(x) = \mathrm{tr}_1(i)(x)$
- $\forall x \in X\ \forall j \in \mathbf{N}, \mathrm{tr}_2(j)(x) = \perp,\ \text{if}\ j \notin \mathrm{range}(f)$
- $\forall i\ j \in \mathbf{N}, i < j \Rightarrow f(i) < f(j)$

The Coq expression is given as follows. trGetEV is used to get the value (including $\perp$) of each signal at each instant of a trace.

Fixpoint trGetEV tr i x : EValue :=
  match i, tr **with**
    O, (Tr st tr′) ⇒ st x
  | (S j), (Tr st tr′) ⇒ trGetEV tr′ j x
  end.

Record Stretching(tr$_1$: Trace)(tr$_2$: Trace)
  : **Prop** := {
  Stretch_f : nat!nat;
  Stretch_val : ∀ x i, trGetEV tr$_1$ i x
          = trGetEV tr$_2$ (Stretch_f i) x;
  Stretch_bot : ∀ x j, (∀ i, Stretch_f $j \neq i$)
          → trGetEV tr$_2$ j x = Absence;
  Stretch_mono : ∀ i j, $i < j$

$\rightarrow$ Stretch_f $i <$ Stretch_f j

}.

Definition 6 (Stretch equivalence) For a given subset X of $\boldsymbol{X}$, two traces tr_1 and tr_2 are stretch-equivalent, noted $\mathrm{tr}_1 \gtrless \mathrm{tr}_2$, iff there exists another behavior tr_3 less stretched than both tr_1 and tr_2, i.e., $\mathrm{tr}_1 \gtrless \mathrm{tr}_2$ iff $\exists \mathrm{tr}_3\ \mathrm{tr}_3 \leqslant \tau_X \mathrm{tr}_1$ and $\mathrm{tr}_3 \leqslant \tau_X \mathrm{tr}_2$.

The Coq expression is given as follows:

Inductive Stretch_Equivalence(tr_1 : Trace)
(tr_2 : Trace) : **Prop** :=
$\mathrm{Str_EqPrf}$: $\forall$ tr_3 : Trace, Stretching tr_3 tr_1
$\rightarrow$ Stretching tr_3 tr_2
$\rightarrow$ Stretch_Equivalence tr_1 tr_2.

Definition 7 (Stretch closure) For a given trace tr, the set of all traces that are stretch-equivalent to tr, defines its stretch closure, noted tr^*.

The stretch closure of a set of traces $\boldsymbol{\tau}_X$, includes all the traces resulting from the stretch closure of each trace $\mathrm{tr} \in \tau_X$, i.e., $\bigcup_{\mathrm{tr}\in\tau_X} \mathrm{tr}^*$.

The Coq expression is given as follows:

Inductive Stretch_Closure(trs : Trace $\rightarrow$ **Prop**)
: Trace $\rightarrow$ **Prop** :=
Stretch_cl : $\forall$ tr_1 tr_2 : Trace, trs tr_1
$\rightarrow$ Stretch_Equivalence tr_1 tr_2
$\rightarrow$ Stretch_Closure trs tr_2.

Definition 8 (Stretch-closed) A SIGNAL process is stretch-closed, iff, for all $\mathrm{tr}' \in$ Sprocess.straces and for all $\mathrm{tr} \in \tau_X$, $\mathrm{tr} \gtrless \mathrm{tr}' \Rightarrow \mathrm{tr} \in$ Sprocess.straces

4.2 Tagged model

Lee and Sangiovanni-Vincentelli proposed the tagged-signal model [19] to compare various models of computation. It is a denotational approach where a system is modeled as a set of behaviors. Behaviors are sets of events. Each event is a value-tag pair. Complex systems are derived through the parallel composition of sub-systems, by taking the intersection of the sets of behaviors. After that, the tagged-signal model is also used to express the semantics of the SIGNAL language [10, 12], because this model can represent the feature of multi-clock naturally.

We reuse the sets $\boldsymbol{X}$ and $\boldsymbol{V}$ defined in Section 4.1.

Definition 9 (Tag structure) A tag structure is a tuple ($\boldsymbol{T}$, $\leqslant$), where:

- $\boldsymbol{T}$ is the set of tags.
- $\leqslant$ is a partial order on $\boldsymbol{T}$.

The Coq expression is given as follows. Tag represents a set of tags, tle is a partial order, and tlt is defined as a strict partial order.

Record TAG : **Type** := {
Tag : **Type**;
tle : Tag $\rightarrow$ Tag $\rightarrow$ **Prop**;
tpo : order Tag tle;
tlt t_1 t_2 := tle t_1 $t_2 \wedge t_1 \neq t_2$;
}.

Definition 10 (Tagged event) [10] A tagged event e on a given tag structure ($\boldsymbol{T}$, $\leqslant$) is a pair $(t, v) \in \boldsymbol{T} \times \boldsymbol{V}$.

Example 4 A tag structure associated with events is given in Fig. 2. Sharing the same tag among different events represents the events are synchronous at that logical instant.

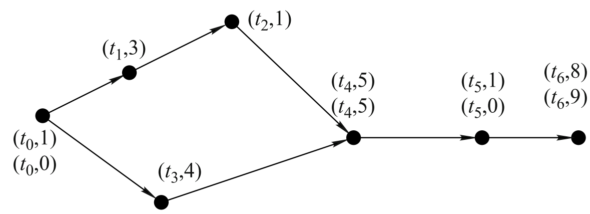

Fig. 2 A tag structure with events

A totally ordered set of tags $C \in \boldsymbol{T}$ is called a chain, and $\min\{C\}$ denotes the minimum element of C. In addition, we denote by C_T the set of all chains on ($\boldsymbol{T}$, $\leqslant$).

Definition 11 (TSignal) A signal on a tag structure ($\boldsymbol{T}$, $\leqslant$) is a partial function $s \in C \rightharpoonup \boldsymbol{V}$ which associates values with the tags that belong to a chain C.

Let the set of signals on ($\boldsymbol{T}$, $\leqslant$) be noted S_T. Here, we give two signals as an example (see Fig. 3).

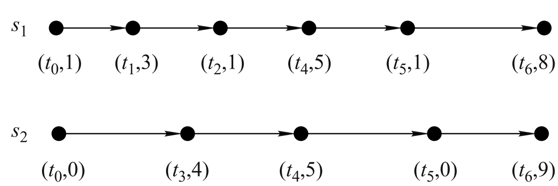

Fig. 3 Two signals of the tag structure in Fig. 2

The Coq expression is given as follows. The type Tsignal_from is used to construct a chain from a tag t. Tsignal represents the set of signals. "@<" is the notation for the strict partial order tlt.

CoInductive Tsignal_from{G : TAG}(t : Tag G) : **Type** :=
  Tend : Tsignal_from t
| Tnext : $\forall\ t_n,\ t@ < t_n$ → Value
  → Tsignal_from t_n → Tsignal_from t.
Inductive Tsignal G : Type :=
  Tempty : Tsignal G
| Tfrom : ∀ (t : Tag G), Value
  → Tsignal_from t → Tsignal G.

Definition 12 (Behavior) Given a tag structure $(\boldsymbol{T}, \leqslant)$, a behavior b on $\boldsymbol{X} \subseteq X$ is a function $b \in X \to S_T$ that associates each variable $x \in X$ with a signal s on $(\boldsymbol{T}, \leqslant)$.

Notice that, here signal variables and signals are treated separately, and the behaviors on tag structures give the mapping between them.

The Coq expression is given as follows. In the type Tbehavior, each variable is associated with a signal.

Definition Tbehavior (G : TAG) :=
  XVar → Tsignal G.

We denote by $B_{|X}$ the set of behaviors of domain $X \subseteq \boldsymbol{X}$ on $(\boldsymbol{T}, \leqslant)$. Given a behavior $b \in B_{|X}$, we write vars(b) and tags($b(x)$) ($x \in$ vars(b)) to denote the signal variables considered in b and the set of tags associated with the signal variable x. $0_{|X}$ expresses the association of X with empty signal.

Definition 13 (Tprocess) Given a SIGNAL process, its tagged model semantics, denoted as Tprocess, includes a set of signal variables and a set of behaviors on tag structures.

The Coq expression is given as follows:

Record Tprocess (G : TAG) := {
  tdom : XVar → **Prop**;
  tbehaviors : Tbehavior G → **Prop**
}.

Remark 1 The logical time used in the trace model is a totally ordered set, and the absence of events is explicitly specified, while the logical time used in the tagged model is a partially ordered set, and the absence of events is not specified. Moreover, a tag structure may correspond to a set of traces.

Additionally, we give the definition of the stretch-closure property on the tagged model [10,12]. The intuition is to consider a signal as an elastic with tags on it. If it is stretched, tags remain in the same order but have more space (time) between each other (see Fig. 4). The same holds for a set of elastics: a behavior. If elastics are equally stretched, the partial order between tags is unchanged.

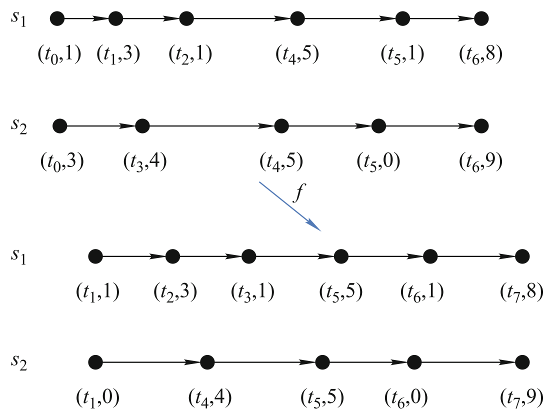

Fig. 4 Stretching of a behavior composed of two signals following f

Definition 14 (Stretching) For a given domain $X \subseteq \boldsymbol{X}$, a behavior b_1 is less stretched than another behavior b_2, noted $b_1 \leqslant_{B_{|X}} b_2$, iff there exists a mapping f : tags(b_1) → tags(b_2) following b_1 and b_2 are isomorphic:

- $\forall x \in \text{vars}(b_1), f(\text{tags}(b_1(x))) = \text{tags}(b_2(x))$
- $\forall x \in \text{vars}(b_1)\ \forall t \in \text{tags}(b_1(x)), b_1(x)(t) = b_2(x)(f(t))$
- $\forall t_1, t_2 \in \text{tags}(b_1), t_1 < t_2 \Rightarrow f(t_1) < f(t_2)$
- $\forall C \in C_T, \forall t \in C, t \leqslant f(t)$

The Coq expression is given as follows. tags_from and tags are used to get the tags of a given signal, btags represents the tags of all the signals in a given behavior, while tval_from and tval are used to get the value at each tag of a signal. "@<=" is the notation of tle.

Inductive tags_from {G}(t t_0 : Tag G)
  : Tsignal_from t_0 → **Prop** :=
  in_curr : ∀ t_i h v_i s', $t = t_i$
    → tags_from t t_0 (Tnext t_0 t_i h v_i s')
| in_next : ∀ t_i h v_i s', tags_from t t_i s'
    → tags_from t t_0 (Tnext t_0 t_i h v_i s').

Inductive tags {G} t : Tsignal G → **Prop** :=
  in_first : ∀ t_0 v_0 s', $t_0 = t$
    → tags t (Tfrom G t_0 v_0 s')
| in_from : ∀ t_0 v_0 s', tags_from t t_0 s'
    → tags t (Tfrom G t_0 v_0 s').

Inductive btags {G}(b : Tbehavior G)
 (dom : XVar → **Prop**) t : **Prop** :=
  btagsPrf : ∀ x, dom x → tags t (b x)
    → btags b dom t.

Record tStretching $\{G_1\ G_2 : \text{TAG}\}$
$(b_1 : \text{Tbehavior } G_1)(b_2 : \text{Tbehavior} G_2)$
$(\text{dom} : \text{XVar} \rightarrow$ **Prop**$)$: **Prop** := {
tStretch_f : Tag $G_1 \rightarrow$ Tag G_2;
tStretch_tags : $\forall\ t_2\ x,$ dom x
$\rightarrow$ tags t_2 $(b_2\ x)$
$\rightarrow \exists\ t_1,$ tags t_1 $(b_1\ x)$
$\wedge t_2 =$ tStretch_f t_1;
tStretch_val : $\forall\ t\ x\ v,$ dom x
$\rightarrow$ tval $(b_1\ x)\ t\ v$
$\rightarrow$ tval $(b_2\ x)$(tStretch_f t) v;
tStretch_mono : $\forall\ t_1\ t_2$: Tag G_1,
btags b_1 dom t_1
$\rightarrow$ btags b_1 dom $t_2 \rightarrow t_1 @ < t_2$
$\rightarrow$ tStretch_f $t_1 @ <$ tStretch_f t_2;
tStretch_incr : $\forall\ t,\ t @ <=$ tStretch_f t
}.

Definition 15 (Stretch equivalence) For a given domain $X \subseteq \boldsymbol{X}$, two behaviors b_1 and b_2 are stretch-equivalent, noted $b_1 \gtrless b_2$, iff there exists another behavior b_3 less stretched than both b_1 and b_2, i.e., $b_1 \gtrless b_2$ iff $\exists b_3\ b_3 \leqslant_{B_{|X}} b_1$ and $b_3 \leqslant_{B_{|X}} b_2$.

The Coq expression is given as follows.

Inductive tStretch_Equivalence $\{G_1\ G_2 : \text{TAG}\}$
$(b_1 : \text{Tbehavior } G_1)(b_2 : \text{Tbehavior } G_2)$
$(\text{dom} : \text{XVar} \rightarrow$ **Prop**$)$: **Prop** :=
tStrEq : $\forall\ G_3\ (b_3 : \text{Tbehavior } G_3)$,
tStretching $b_3\ b_1$ dom
$\rightarrow$ tStretching $b_3\ b_2$ dom
$\rightarrow$ tStretch_Equivalence $b_1\ b_2$ dom.

Definition 16 (Stretch closure) For a given behavior b, the set of all behaviors that are stretch-equivalent to b, defines its stretch closure, noted b^*.

The stretch closure of a set of behaviors $B_{|X}$ includes all the behaviors resulting from the stretch closure of each behavior $b \in B_{|X}$, i.e., $\bigcup_{b \in B_{|X}} b^*$.

The Coq expression is given as follows.

Inductive tStretch_Closure $\{G : \text{TAG}\}$
(tb : Tbehavior $G \rightarrow$ **Prop**)(dom : XVar
$\rightarrow$ **Prop**) : Tbehavior $G \rightarrow$ **Prop** :=
tStretch_cl : $\forall\ b_1\ b_2,\ tb\ b_1$
$\rightarrow$ tStretch_Equivalence $b_1\ b_2$ dom
$\rightarrow$ tStretch_Closure tb dom b_2.

Definition 17 (Stretch-closed) A SIGNAL process is stretch-closed, iff, for all $b' \in$ Tprocess.tbehaviors and for all $b \in B_{|X}$, $b \gtrless b' \Rightarrow b \in$ Tprocess.tbehaviors

5 Two formal semantics

Primitive constructs of the SIGNAL language specify the relations between signals at the syntax level. The trace semantics and the tagged model semantics are both denotational style. They interpret and define precisely the relations between values and the relations between clocks of signals in their semantics domains. In this paper, the semantics ignores the local declaration of signal variables to get a simplest criterion for the proof of semantics equivalence.

5.1 Trace semantics

There are several definitions of the trace semantics of SIGNAL [9–11], we select [10] as the reference paper semantics and mechanize it in Coq. Most of the Coq expressions are close to the paper semantics, but some expressions are not, so we need to justify the equivalence between them. We also refer to the Coq expressions of Nowak [14, 15].

Here, each single signal is observed in the reference paper semantics, while the corresponding trace with signal variables $x, x_1, \ldots, x_n$ is directly used in the Coq expressions. The difference between them has been given in Section 4.1. The mapping between them is done at the end (i.e., the definition Process2Sprocess).

Trace Semantics 1 (Instantaneous function) The trace semantics of the instantaneous function is defined as follows:

$$\forall \tau \in \mathbf{N},$$
$$s_\tau = \begin{cases} \perp, & \text{if } s_{1\tau} = s_{2\tau} = \cdots = s_{n\tau} = \perp, \\ f(s_{1\tau}, s_{2\tau}, \ldots, s_{n\tau}), & \text{if } s_{1\tau} \neq \perp \wedge \cdots \wedge s_{n\tau} \neq \perp. \end{cases}$$

At each instant τ, the signals are either all present or all absent, i.e., they are synchronous, denoted as $s\,\hat{}= s_1\,\hat{}= \cdots\,\hat{}= s_n$. s_τ gets the value of $f(s_{1\tau}, s_{2\tau}, \ldots, s_{n\tau})$ when the signals are all present. The function f includes different mathematical operations, such as arithmetic operations, boolean operations, etc.

The corresponding Coq expression is given as follows.

CoInductive Sassignment x Index (f : (Index $\rightarrow$
Value) $\rightarrow$ Value)(x_i : Index $\rightarrow$ Var)
: Trace $\rightarrow$ **Prop** :=
SassU : $\forall\ st$ tr, $(\forall\ i,\ st\ (x_i\ i) = \text{Absence})$

$\rightarrow st\ x = \text{Absence}$
$\rightarrow \text{Sassignment}\ x\ \text{Index}\ f\ x_i\ \text{tr}$
$\rightarrow \text{Sassignment}\ x\ \text{Index}\ f\ x_i\ (\text{Tr}\ st\ \text{tr})$
$|\ \ \text{SassP} : \forall\ v\ st\ \text{tr}, (\forall\ i, st\ (x_i\ i) = \text{Val}(v\ i))$
$\rightarrow st\ x = \text{Val}\ (f\ v)$
$\rightarrow \text{Sassignment}\ x\ \text{Index}\ f\ x_i\ \text{tr}$
$\rightarrow \text{Sassignment}\ x\ \text{Index}\ f\ x_i\ (\text{Tr}\ st\ \text{tr}).$

Trace Semantics 2 (Delay) The trace semantics of the delay construct is defined as follows:

– $(\forall \tau \in \mathbf{N})\ s_{1\tau} = \perp \Leftrightarrow s_\tau = \perp$
– $\{k \mid s_{1k} \neq \perp\} \neq \emptyset \Rightarrow s_{\min\{k \mid s_{1k} \neq \perp\}} = c$
– $(\forall \tau \in \mathbf{N})\ s_{1\tau} \neq \perp \wedge \{k > \tau \mid s_{1k} \neq \perp\} \neq \emptyset$
$\Rightarrow s_{\min\{k > \tau \mid s_{1k} \neq \perp\}} = s_{1\tau}$

Here, we make the definition of the trace semantics of Delay in [10] more precise. $\min(S)$ denotes the minimum of a non-empty set of naturals. Similarly to the instantaneous function, the delay construct also requires signals s and s_1 have the same clock, denoted as $s\ \hat{=}\ s_1$. Given a logical instant τ, s takes the most recent value of s_1 except the one at τ. Initially, s takes the value c.

The Coq expression is given as follows.

CoInductive Sdelay $x\ x_1\ c$: Trace $\rightarrow$ **Prop** :=
$\text{SdelayU} : \forall\ st\ \text{tr},\ st\ x_1 = \text{Absence}$
$\rightarrow st\ x = \text{Absence}$
$\rightarrow \text{Sdelay}\ x\ x_1\ c\ \text{tr}$
$\rightarrow \text{Sdelay}\ x\ x_1\ c\ (\text{Tr}\ st\ \text{tr})$
$|\ \ \text{SdelayP} : \forall\ st\ v\ \text{tr},\ st\ x_1 = \text{Val}\ v$
$\rightarrow st\ x = \text{Val}\ c$
$\rightarrow \text{Sdelay}\ x\ x_1\ v\ \text{tr}$
$\rightarrow \text{Sdelay}\ x\ x_1\ c\ (\text{Tr}\ st\ \text{tr}).$

Trace Semantics 3 (Undersampling) The trace semantics of the undersampling construct is defined as follows:

$$\forall \tau \in \mathbf{N},\quad s_\tau = \begin{cases} s_{1\tau} & \text{if } s_{2\tau} = \text{true}, \\ \perp & \text{otherwise.} \end{cases}$$

Here, s and s_1 have the same type and s_2 is a boolean signal. The clock of s is the intersection of the clock of s_1 and the clock of s_2, denoted as $s = s_1\ \hat{}*\ [s_2]$, while $[s_2]$ represents the true occurrences of s_2. Given a logical instant τ, s_τ gets the value of $s_{1\tau}$ when $s_{2\tau}$ is true, else gets the value $\perp$.

The Coq expression is given as follows.

CoInductive Swhen($x\ x_1\ x_2$: XVar) : Trace $\rightarrow$ **Prop** :=
$\text{SwhenT} : \forall\ st\ v\ b\ \text{tr},\ \text{isTrue}\ b$
$\rightarrow st\ x = \text{Val}\ v \rightarrow st\ x_1 = \text{Val}\ v$
$\rightarrow st\ x_2 = \text{Val}\ b \rightarrow \text{Swhen}\ x\ x_1\ x_2\ \text{tr}$
$\rightarrow \text{Swhen}\ x\ x_1\ x_2\ (\text{Tr}\ st\ \text{tr})$
$|\ \ \text{SwhenF} : \forall\ st\ b\ \text{tr},\ \neg\text{isTrue}\ b$
$\rightarrow st\ x = \text{Absence} \rightarrow st\ x_2 = \text{Val}\ b$
$\rightarrow \text{Swhen}\ x\ x_1\ x_2\ \text{tr}$
$\rightarrow \text{Swhen}\ x\ x_1\ x_2\ (\text{Tr}\ st\ \text{tr})$
$|\ \ \text{SwhenU} : \forall\ st\ \text{tr},\ st\ x = \text{Absence}$
$\rightarrow st\ x_2 = \text{Absence}$
$\rightarrow \text{Swhen}\ x\ x_1\ x_2\ \text{tr}$
$\rightarrow \text{Swhen}\ x\ x_1\ x_2\ (\text{Tr}\ st\ \text{tr}).$

Trace Semantics 4 (Deterministic merging) The trace semantics of the deterministic merging construct is defined as follows:

$$\forall \tau \in \mathbf{N},\quad s_\tau = \begin{cases} s_{1\tau} & \text{if } s_{1\tau} \neq \perp, \\ s_{2\tau} & \text{otherwise.} \end{cases}$$

Here, signals s, s_1 and s_2 have the same type. The clock of s is the union of the clocks of s_1 and s_2, denoted as $s = s_1\ \hat{}+\ s_2$. Given a logical instant τ, s_τ gets the merge of the values of $s_{1\tau}$ and $s_{2\tau}$, and the value of $s_{1\tau}$ has a higher priority.

The Coq expression is given as follows.

CoInductive Sdefault($x\ x_1\ x_2$: Var) : Trace $\rightarrow$ **Prop** :=
$\text{SdefaultU} : \forall\ st\ \text{tr},\ st\ x = \text{Absence}$
$\rightarrow st\ x_1 = \text{Absence}$
$\rightarrow st\ x_2 = \text{Absence}$
$\rightarrow \text{Sdefault}\ x\ x_1\ x_2\ \text{tr}$
$\rightarrow \text{Sdefault}\ x\ x_1\ x_2\ (\text{Tr}\ st\ \text{tr})$
$|\ \ \text{Sdefault1} : \forall\ st\ v\ \text{tr},\ st\ x = \text{Val}\ v$
$\rightarrow st\ x_1 = \text{Val}\ v$
$\rightarrow \text{Sdefault}\ x\ x_1\ x_2\ \text{tr}$
$\rightarrow \text{Sdefault}\ x\ x_1\ x_2\ (\text{Tr}\ st\ \text{tr})$
$|\ \ \text{Sdefault2} : \forall\ st\ v\ \text{tr},\ st\ x = \text{Val} v$
$\rightarrow st\ x_1 = \text{Absence}$
$\rightarrow st\ x_2 = \text{Val}\ v$
$\rightarrow \text{Sdefault}\ x\ x_1\ x_2\ \text{tr}$
$\rightarrow \text{Sdefault}\ x\ x_1\ x_2\ (\text{Tr}\ st\ \text{tr}).$

Finally, we apply these semantics rules to a SIGNAL process, to get a complete semantics of the process, that is Sprocess (defined in Section 4.1). SPassignment, SPdelay, SPwhen and SPdefault, used to construct the corresponding Sprocess on the semantics rule Sassignment, Sdelay, Swhen and Sdefault respectively, while the function Process2Sprocess is used to combine them as one Sprocess. We also give the semantics of processes composition, that is SPprod.

Program **Definition** SPassignment x Ind f x_i :=
{|
sdom y := $y = x \ \vee \ \exists\, i,\ y = x_i\ i$;
straces tr := Sassignment x Ind f x_i tr
|}.
Program **Definition** SPdelay x x_1 c :=
{|
sdom y := $y = x \vee y = x_1$;
straces tr := Sdelay x x_1 c tr
|}.
Program **Definition** SPwhen x x_1 x_2 :=
{|
sdom y := $y = x \vee y = x_1 \vee y = x_2$;
straces tr := Swhen x x_1 x_2 tr
|}.
Program **Definition** SPdefault x x_1 x_2 :=
{|
sdom y := $y = x \vee y = x_1 \vee y = x_2$;
straces tr := Sdefault x x_1 x_2 tr
|}.
Program **Definition** SPprod p_1 p_2 :=
{|
sdom y := sdom p_1 y $\vee$ sdom p_2 y;
straces tr := straces p_1 tr
$\wedge$straces p_2 tr
|}.

Fixpoint Process2Sprocess(p : Process)
: Sprocess :=
match p **with**
Pass Ind f x $x_i \Rightarrow$ SPassignment x Ind f x_i
| Pwhen x x_1 $x_2 \Rightarrow$ SPwhen x x_1 x_2
| Pdelay x x_1 $c \Rightarrow$ SPdelay x x_1 c
| Pdefault x x_1 $x_2 \Rightarrow$ SPdefault x x_1 x_2
| Ppar p_1 p_2
$\Rightarrow$ SPprod(Process2Sprocess p_1)
(Process2Sprocess p_2)
end

Example 5 The trace semantics of the process ParallelCount (Example 2) is a set of traces, and two possible traces are given as follows. Here, we just consider the external visible signals (the local declarations are hidden).

$$\mathrm{tr}_1 : \begin{array}{l} x_1\ 1\ \perp 2\ \perp 0\ 1\ \perp 2\ \perp 3\ \perp 0 \perp \cdots \\ x_2 \perp 1\ \perp 2\ 0 \perp 1\ \perp 2\ \perp 3\ 0 \perp \cdots \end{array}$$

$$\mathrm{tr}_2 : \begin{array}{l} x_1\ 0\ 1\ 2\ \perp 0\ 1\ 2\ \perp 3\ 0 \perp \cdots \\ x_2\ 0 \perp \perp 1\ 0 \perp \perp 1\ \perp 0 \perp \cdots \end{array}$$

Property 1 For all SIGNAL processes, the trace semantics is stretch-closed.

5.2 Tagged model semantics

Similarly, there are several definitions of the tagged model semantics of SIGNAL [10, 12], we select [10] as the reference paper semantics and mechanize it in Coq.

Here, signal variables $x, x_1, \ldots, x_n$ are used in the reference paper semantics, while the tag structure with signals $s, s_1, \ldots, s_n$ is used in the Coq expressions. The relation between them has been shown in Section 4.2. The mapping between them is done at the end (i.e., the definition Process2Tprocess).

Tagged Model Semantics 1 (Instantaneous function) The tagged model semantics of the instantaneous function is defined as follows:

$$\begin{aligned} &[\![x := f(x_1, x_2, \ldots, x_n)]\!] = \\ &\{b \in B_{|x,x_1,\ldots,x_n}|\mathrm{tags}(b(x)) = \mathrm{tags}(b(x_1)) = \cdots = \mathrm{tags}(b(x_n)) \\ &= C \in C_T \text{ and } \forall t \in C, b(x)(t) \\ &= [\![f]\!](b(x_1)(t), b(x_2)(t), \ldots, b(x_n)(t))\} \end{aligned}$$

The semantics of the instantaneous function is the set of behaviors b. The tags of each signal involved in b represent the same chain C, i.e., all the signals are synchronous. When the signals are all present, at each tag of C, the output signal gets the corresponding value.

The corresponding Coq expression is given as follows. TSA_T is used to express the relation between values, while TSA_S represents all the signals are synchronous. tval_from and tval represent that, given a signal of a tag structure G and a tag of the signal, we can get the corresponding value. tsync means two signals are synchronous.

Inductive tval_from{G}(t_0 : Tag G) :
Tsignal_from t_0 → Tag G → Value → **Prop** :=
tv_curr : $\forall\ t\ h\ v\ s\ tt\ vv,\ t = tt \rightarrow v = vv$
$\rightarrow$ tval_from t_0 (Tnext t_0 t h v s) tt vv
| tv_next : $\forall\ t\ h\ v\ s\ tt\ vv$,
tval_from t s tt vv $\rightarrow$

tval_from t_0 (Tnext t_0 t h v s) tt vv.
Inductive tval $\{G\}$: Tsignal $G \rightarrow$ Tag $G \rightarrow$
Value $\rightarrow$ **Prop** :=
tv_first : $\forall$ t v s tt vv, $t = tt \rightarrow v = vv$
$\rightarrow$ tval (Tfrom G t v s) tt vv
| tv_from : $\forall$ t_0 v s tt vv,
tval_from t_0 s tt vv $\rightarrow$
tval (Tfrom G t_0 v s) tt vv.
Definition tsync $\{G\}(s_1\ s_2$: Tsignal $G)$: **Prop** :=
$\forall$ t, tags t $s_1 \leftrightarrow$ tags t s_2.

Record TSassignment $\{G\}$ s Index (f : (Index
$\rightarrow$ Value) $\rightarrow$ Value)(s_i : Index $\rightarrow$ Tsignal G)
: **Prop** := {
TSA_T : $\forall$ t d v, ($\forall$ i,
tval (s_i i) t (d i))
$\rightarrow$ tval s t $v \rightarrow v = f\ d$;
TSA_S : $\forall$ i, tsync (s_i i) s
}.

Tagged Model Semantics 2 (Delay) The tagged model semantics of the delay construct is defined as follows:

$$\begin{aligned}&[\![x := x_1\$ \text{ init } c]\!] = \\ &\{0_{|x,x_1}\}\cup \\ &\{b \in B_{x,x_1} |\ \text{tags}(b(x)) = \text{tags}(b(x_1)) = C \in C_T\backslash\{\emptyset\}; \\ &\qquad b(x)(\min(C)) = c; \\ &\qquad \forall t \in C\backslash \min(C), b(x)(t) = b(x_1)(\text{pred}_C(t))\}\end{aligned}$$

Similarly to the instantaneous function, the tags of each signal represent the same chain C. When the signals are both present, x gets the value c at the initial tag of C, and for all the other tags $t \in C$, x gets the value carried by x_1 at the predecessor of t.

The Coq expression is given as follows. TSY0 and TSYN are used to express the relation between values, while TSYL represents the signals are synchronous. tfirst s t represents that t is the first tag of a given signal s, and tnext s_1 t_1 t_2 means t_2 is the next tag of t_1 of a given signal s_1 (it has the same meaning as $t_1 = \text{pred}_C(t_2)$).

Inductive tfirst $\{G\}$: Tsignal $G \rightarrow$ Tag G
$\rightarrow$ **Prop** :=
tf_prf : $\forall$ t v s tt, $t = tt$
$\rightarrow$ tfirst (Tfrom G t v s) tt.
Inductive tnext_from $\{G\}(t_0$: Tag $G)$:
Tsignal_from $t_0 \rightarrow$ Tag $G \rightarrow$ Tag G
$\rightarrow$ **Prop** :=
tnf0 : $\forall$ t h v s t_1 t_2, $t_1 = t_0 \rightarrow t_2 = t$
$\rightarrow$ tnext_from t_0 (Tnext t_0 t h v s) t_1 t_2
| tnfi : $\forall$ t h v s t_1 t_2, tnext_from t s t_1 t_2
$\rightarrow$ tnext_from t_0 (Tnext t_0 t h v s) t_1 t_2.
Inductive tnext $\{G\}$: Tsignal $G \rightarrow$ Tag G
$\rightarrow$ Tag $G \rightarrow$ **Prop** :=
tnn : $\forall$ t v s t_1 t_2, tnext_from t s t_1 t_2
$\rightarrow$ tnext (Tfrom G t v s) t_1 t_2.

Record TSdelay $\{G\}(s\ s_1$: Tsignal $G)$ c : **Prop** := {
TSY0 : $\forall$ t, tfirst s $t \rightarrow$ tval s t c;
TSYN : $\forall$ t_1 t_2 v, tnext s_1 t_1 t_2
$\rightarrow$ tval s_1 t_1 $v \rightarrow$ tval s t_2 v;
TSYL : tsync s s_1
}.

Tagged Model Semantics 3 (Undersampling) The tagged model semantics of the undersampling construct is defined as follows:

$$\begin{aligned}&[\![x := x_1 \text{ when } x_2]\!] = \\ &\{b \in B_{|x,x_1,x_2}|\text{tags}(b(x)) = \{t \in \text{tags}(b(x_1)) \\ &\cap\text{tags}(b(x_2))|b(x_2)(t) = \text{true}\} = C \in C_T \\ &\text{and } \forall t \in C, b(x)(t) = b(x_1)(t)\}\end{aligned}$$

The set of tags of x is the intersection of the set of tags associated with x_1 and the set of tags at which x_2 carries the value true. Moreover, at each tag of x, the value held by x is the value of x_1.

The Coq expression is given as follows. Here, we give all the cases. tnval s t means it is absent at the tag t of a given signal s.

Definition tnval $\{G\}$ s (t : Tag G) : **Prop** :=
$\neg\exists$ v, tval s t v.

Record TSwhen $\{G\}(s\ s_1\ s_2$: Tsignal $G)$: **Prop** := {
TSW_T : $\forall$ t v b, tval s_1 t v
$\rightarrow$ tval s_2 t $b \rightarrow$ isTrue b
$\rightarrow$ tval s t v;
TSW_F : $\forall$ t b, tval s_2 t b
$\rightarrow \neg$isTrue $b \rightarrow$ tnval s t;
TSW_U1 : $\forall$ t, tnval s_1 $t \rightarrow$ tnval s t;
TSW_U2 : $\forall$ t, tnval s_2 $t \rightarrow$ tnval s t
}.

Tagged Model Semantics 4 (Deterministic merging) The tagged model semantics of the deterministic merging construct is defined as follows:

$$[\![x := x_1 \text{ default } x_2]\!] =$$
$$\{b \in B_{|x,x_1,x_2}|\text{tags}(b(x)) = \text{tags}(b(x_1)) \cup \text{tags}(b(x_2)) = C \in C_T$$
$$\text{and } \forall t \in C, b(x)(t) = b(x_1)(t) \text{ if } t \in \text{tags}(b(x_1)) \text{ else } b(x_2)(t)\}$$

The set of tags of x is the union of the tags of x_1 and x_2. The value taken by x is that of x_1 at any tag when x_1 is present. Otherwise, it takes the value of x_2 at its tags, which do not belong to the tags of x_1.

The Coq expression is given as follows.

Record TSdefault$\{G\}(s\ s_1\ s_2$: Tsignal $G)$: **Prop** := {
  TSD0 : $\forall\ t\ v,$ tval $s\ t\ v \rightarrow$
    (tval $s_1\ t\ v \vee$ tnval $s_1\ t \wedge$ tval $s_2\ t\ v$);
  TSD1 : $\forall\ t\ v,$ tval $s_1\ t\ v \rightarrow$ tval $s\ t\ v$;
  TSD2 : $\forall\ t\ v,$ tnval $s_1\ t \rightarrow$
    tval $s_2\ t\ v \rightarrow$ tval $s\ t\ v$
}.

Finally, we apply these semantics rules to a SIGNAL process, to get a complete semantics of the process, that is Tprocess (defined in Section 4.2). Tassignment, Tdelay, Twhen and Tdefault, used to construct the corresponding Tprocess on the semantics rule TSassignment, TSdelay, TSwhen and TSdefault respectively, while the function Process2Tprocess is used to combine them as one Tprocess. The semantics of processes composition is defined in Tpar.

Definition Tassignment $\{G\}$ x Index (f : (Index
  $\rightarrow$ Value) $\rightarrow$ Value)(x_i : Index $\rightarrow$ XVar)
  : Tprocess G :=
{|
  tdom $y := y = x \vee \exists i,\ y = x_i\ i$;
  tbehaviors b := TSassignment ($b\ x$) Index f
    (fun $i \Rightarrow (b\ (x_i\ i))$)
|}.

Definition Tdelay$\{G\}(x\ x_1$: XVar) c
  : Tprocess G :=
{|
  tdom $y := y = x \vee y = x_1$;
  tbehaviors b := TSdelay ($b\ x$)($b\ x_1$) c
|}.

Definition Twhen $\{G\}$ $x\ x_1\ x_2$: Tprocess G :=
{|
  tdom $y := y = x \vee y = x_1 \vee y = x_2$;
  tbehaviors b := TSwhen($b\ x$)($b\ x_1$)($b\ x_2$)
|}.

Definition Tdefault $\{G\}$ $x\ x_1\ x_2$: Tprocess G :=
{|
  tdom $y := y = x \vee y = x_1 \vee y = x_2$;
  tbehaviors b := TSdefault ($b\ x$)($b\ x_1$)($b\ x_2$)
|}.

Definition Tpar $\{G\}$ ($p_1\ p_2$: Tprocess G) :=
{|
  tdom y := tdom $p_1\ y \vee$ tdom $p_2\ y$;
  tbehaviors b := tbehaviors $p_1\ b$
    $\wedge$tbehaviors $p_2\ b$
|}.

Fixpoint Process2Tprocess G (p : Process)
  : Tprocess G :=
  match p **with**
   Pass Ind $f\ x\ x_i \Rightarrow$ Tassignment x Ind $f\ x_i$
  | Pdelay $x\ x_1\ c \Rightarrow$ Tdelay $x\ x_1\ c$
  | Pwhen $x\ x_1\ x_2 \Rightarrow$ Twhen $x\ x_1\ x_2$
  | Pdefault $x\ x_1\ x_2 \Rightarrow$ Tdefault $x\ x_1\ x_2$
  | Ppar $p_1\ p_2 \Rightarrow$ Tpar(Process2Tprocess $G\ p_1$)
    (Process2Tprocess $G\ p_2$)
end

Example 6 The tagged model semantics of the process ParallelCount (Example 2) is a set of behaviors, and two examples are shown in Fig. 5. Similarly, we just consider the external visible signals.

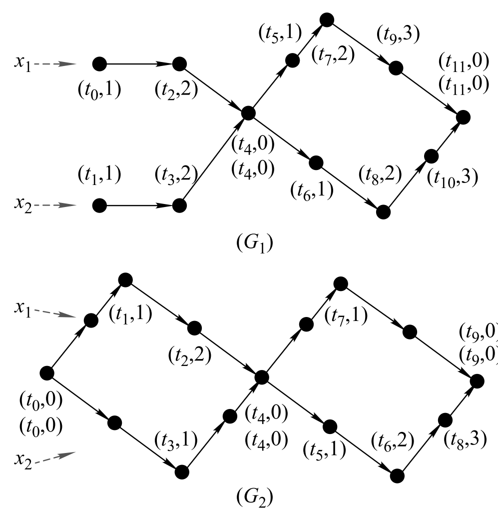

Fig. 5 The tag structures of two possible behaviors of the process ParallelCount

Property 2 [12] For all SIGNAL processes, the tagged model semantics is stretch-closed.

Property 1 and Property 2 represent that a SIGNAL process can be used at different time scales because its semantics is closed for the stretch-equivalence relation.

6 The proof of the semantics equivalence

The trace semantics and the tagged model semantics are very different models, so the equivalence between them (Theorems S2Teq and T2Seq) is established through an intermediate model. The global idea is sketched in Fig. 6.

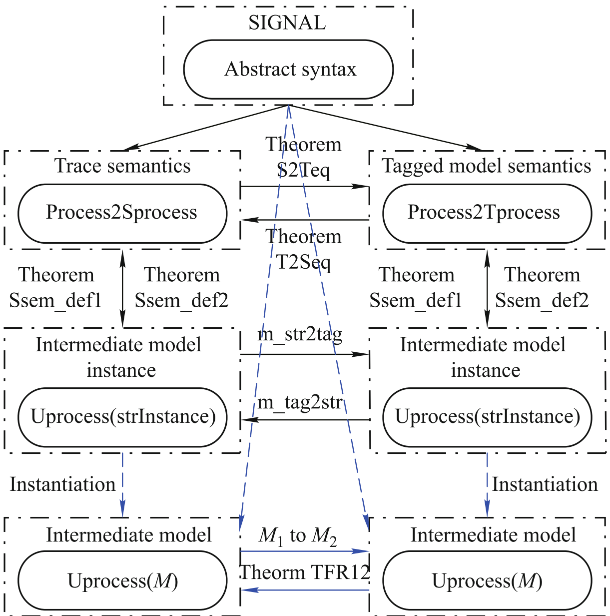

Fig. 6 Proof's plan

The intermediate model M is generic and parameterized by:

1) mdom, the domain of M, such as a set of traces, a set of behaviors on a tag structure;
2) mget $m\ x\ i\ v$, is true in domain m if variable x gets the i^{th} non-absent value v;
3) *msync* $m\ x_1\ x_2\ i_1\ i_2$, represents whether the variables x_1 and x_2 are synchronized or not at the i_1^{th} non-absent value and the i_2^{th} non-absent value respectively.

With these three functions, it is possible to give a semantics of SIGNAL, that is Uprocess(M). The difference between the trace semantics and the intermediate model is that the latter just considers non-absent values, while the difference between the tagged model semantics and the intermediate model is that the latter uses a totally ordered set to express logical time. In other words, the intermediate model mixes the features of both the trace semantics and the tagged model semantics. Here, Uprocess(M) is just a general expression, because the domain is unknown. However, we give a general mapping between two intermediate models (M_1 to M_2), and give a basic theorem to prove the equivalence between them (Theorem TFR12).

The trace semantics and the tagged model semantics are considered as instances of the intermediate model, so we transform them to their instance and prove the equivalence (Theorems Ssem_def1, Ssem_def2, Tsem_def1 and Tsem_def2).

Finally, we consider the relation between the two instances. The mapping M_1 to M_2 is refined as m_str2tag and m_tag2str, and the Theorem TFR12 is reused.

6.1 Intermediate model

Firstly, we give the definition of the intermediate model. mdom represents the domain of the model. In this model, we introduce two observers, mget which gives the (finite or infinite) sequence of values taken by each variable, and msync which defines the synchronization points of any couples of variables.

Record Model : **Type** := {
  mdom : **Type**;
  mget : mdom → XVar → nat → Value → **Prop**;
  msync : mdom → XVar → XVar → nat
    → nat → **Prop**
}.

Secondly, we define a semantics of SIGNAL using this model, which is a predicate over $m \in$ mdom. Here, signal variables $x, x_1, \dots, x_n$ are used both in the mathematical model and the Coq expressions.

Intermediate Model 1 (Instantaneous function) The intermediate model of the instantaneous function is defined as follows:

$$[\![x := f(x_1, x_2, \dots, x_n)]\!](m) =$$
$$-\ \forall i \in \mathbf{N}, \forall v_1, v_2, \dots, v_n\ v \in \boldsymbol{V},\ \text{mget}\ m\ x_1\ i\ v_1 \wedge \text{mget}\ m\ x_2\ i\ v_2$$
$$\wedge \cdots \wedge \text{mget}\ m\ x_n\ i\ v_n \wedge \text{mget}\ m\ x\ i\ v$$
$$\Rightarrow v = f(v_1, v_2, \dots, v_n)$$
$$-\ \forall i \in \mathbf{N},\ \text{msync}\ m\ x_1\ x\ i\ i \wedge \text{msync}\ m\ x_2\ x\ i\ i \wedge \cdots$$
$$\wedge\ \text{msync}\ m\ x_n\ x\ i\ i$$

All signals are synchronous and the i^{th} non-absent values of each signal satisfy the functional constant $v = f(v_1, v_2, \dots, v_n)$.

The Coq expression is given as follows, Uass_T represents the relation between values and Uass_S means all signals are synchronous.

Record Uassignment $\{M\}(m$: mdom $M)$ Index
  $(f$: (Index → Value) → Value$)(x$: XVar)
  $(vp$: Index → XVar) : **Prop** := {
  Uass_T : ∀ $d\ v\ i$,

$(\forall\ p, \text{mget}\ m\ (vp\ p)\ i\ (d\ p))$
$\rightarrow \text{mget}\ m\ x\ i\ v \rightarrow v = f\ d;$
$\text{Uass\_S} : \forall\ p\ i,\ \text{msync}\ m\ (vp\ p)\ x\ i\ i$
}.

Intermediate Model 2 (Delay) The intermediate model of the delay construct is defined as follows:

$$[\![x := x_1\$\ \text{init}\ c]\!](m) =$$
$$-\ \text{mget}\ m\ x\ 0\ c$$
$$-\ \forall i \in \mathbf{N}, \forall v_1\ v_2 \in \boldsymbol{V},\ \text{mget}\ m\ x_1\ i\ v_1 \wedge \text{mget}\ m\ x_1\ (i+1)\ v_2$$
$$\Rightarrow \text{mget}\ m\ x\ (i+1)\ v_1$$
$$-\ \forall i \in \mathbf{N},\ \text{msync}\ m\ x\ x_1\ i\ i$$

The two signals x and x_1 are synchronous. mget $m\ x\ 0\ c$ represents the first non-absent value of x is the initial value c, and the $(i+1)$th non-absent value of x is the ith non-absent value of x_1, provided it has an $(i+1)$th value.

The Coq expression is given as follows.

Record Udelay $\{M\}(m : \text{mdom}\ M)\ x\ x_1\ c$: **Prop** := {
$\text{Udelay\_0} : \forall\ v,\ \text{mget}\ m\ x\ 0\ v \rightarrow v = c;$
$\text{Udelay\_S} : \forall\ v_1\ v_2\ i,\ \text{mget}\ m\ x_1\ i\ v_1$
$\rightarrow \text{mget}\ m\ x_1\ (S\ \ i)\ v_2$
$\rightarrow \text{mget}\ m\ x\ (S\ \ i)\ v_1;$
$\text{Udelay\_s} : \forall\ i,\ \text{msync}\ m\ x\ x_1\ i\ i$
}.

Intermediate Model 3 (Undersampling) The intermediate model of the undersampling construct is defined as follows:

$$[\![x := x_1\ \text{when}\ x_2]\!](m) =$$
$$-\ \forall i \in \mathbf{N}, \forall v \in \boldsymbol{V},\ \text{mget}\ m\ x\ i\ v \Rightarrow$$
$$(\exists i_1\ i_2 \in \mathbf{N}, \text{msync}\ m\ x\ x_1\ i\ i_1 \wedge \text{msync}\ m\ x\ x_2\ i\ i_2$$
$$\wedge\ \text{mget}\ m\ x_1\ i_1\ v \wedge \text{mget}\ m\ x_2\ i_2\ \text{true})$$
$$-\ \forall i_1\ i_2 \in \mathbf{N}, \forall v \in \boldsymbol{V},\ \text{msync}\ m\ x_1\ x_2\ i_1\ i_2$$
$$\wedge\ \text{mget}\ m\ x_1\ i_1\ v \wedge \text{mget}\ m\ x_2\ i_2\ \text{true}$$
$$\Rightarrow (\exists i \in \mathbf{N}, \text{msync}\ m\ x\ x_1\ i\ i_1 \wedge \text{mget}\ m\ x\ i\ v)$$

Here, x is defined in the position i if and only if there are two synchronized positions i_1 and i_2 at which x_1 and x_2 are defined, and such as the value of x_2 is true. In such a case, the i^{th} non-absent value of x is the i_1^{th} non-absent value of x_1.

The Coq expression is given as follows.

Record Uwhen $\{M\}(m : \text{mdom}\ M)\ x\ x_1\ x_2$: **Prop** := {
$\text{Uwhen\_v} : \forall\ i\ v,\ \text{mget}\ m\ x\ i\ v \rightarrow$
$\exists\ i_1\ i_2,\ \text{msync}\ m\ x\ x_1\ i\ i_1$
$\wedge \text{msync}\ m\ x\ x_2\ i\ i_2$
$\wedge \text{mget}\ m\ x_1\ i_1\ v$
$\wedge \exists\ b,\ \text{mget}\ m\ x_2\ i_2\ b$
$\wedge \text{isTrue}\ b;$
$\text{Uwhen\_v12} : \forall\ i_1\ i_2\ b\ v,$
$\text{msync}\ m\ x_1\ x_2\ i_1\ i_2$
$\rightarrow \text{mget}\ m\ x_1\ i_1\ v \rightarrow \text{mget}\ m\ x_2\ i_2\ b$
$\rightarrow \text{isTrue}\ b$
$\rightarrow \exists\ i,\ \text{msync}\ m\ x\ x_1\ i\ i_1 \wedge \text{mget}\ m\ x\ i\ v$
}.

Intermediate Model 4 (Deterministic merging) The intermediate model of the deterministic merging construct is defined as follows:

$$[\![x := x_1\ \text{default}\ x_2]\!](m) =$$
$$-\ \forall i \in \mathbf{N}, \forall v \in \boldsymbol{V},\ \text{mget}\ m\ x\ i\ v \Rightarrow$$
$$((\exists i_1 \in \mathbf{N}, \text{msync}\ m\ x\ x_1\ i\ i_1 \wedge \text{mget}\ m\ x_1\ i_1\ v) \vee$$
$$(\neg(\exists i_1 \in \mathbf{N}, \text{msync}\ m\ x\ x_1\ i\ i_1) \wedge$$
$$(\exists i_2 \in \mathbf{N}, \text{msync}\ m\ x\ x_2\ i\ i_2 \wedge \text{mget}\ m\ x_2\ i_2\ v)))$$
$$-\ \forall i\ i_1 \in \mathbf{N}, \forall v \in \boldsymbol{V},\ \text{msync}\ m\ x\ x_1\ i\ i_1 \wedge \text{mget}\ m\ x_1\ i_1\ v$$
$$\Rightarrow \text{mget}\ m\ x\ i\ v$$
$$-\ \forall i\ i_2 \in \mathbf{N}, \forall v \in \boldsymbol{V},\ (\neg(\exists i_1 \in \mathbf{N}, \text{msync}\ m\ x\ x_1\ i\ i_1)$$
$$\wedge\ \text{msync}\ m\ x\ x_2\ i\ i_2 \wedge \text{mget}\ m\ x_2\ i_2\ v \Rightarrow \text{mget}\ m\ x\ i\ v$$

Here, either the i^{th} position of x is synchronized with some position of x_1, or else it is synchronized with some position of x_2. In both cases, the value of x at the i^{th} position is the value of the synchronized one.

The Coq expression is given as follows.

Record Udefault $\{M\}(m : \text{mdom}\ M)\ x\ x_1\ x_2$: **Prop** := {
$\text{Udefault\_v} : \forall\ i\ v,\ \text{mget}\ m\ x\ i\ v \rightarrow$
$((\exists\ i_1,\ \text{msync}\ m\ x\ x_1\ i\ i_1$
$\wedge \text{mget}\ m\ x_1\ i_1\ v) \vee$
$(\neg(\exists\ i_1,\ \text{msync}\ m\ x\ x_1\ i\ i_1)$
$\wedge \exists\ i_2,\ \text{msync}\ m\ x\ x_2\ i\ i_2$
$\wedge \text{mget}\ m\ x_2\ i_2\ v));$
$\text{Udefault\_v1} : \forall\ i\ i_1\ v,\ \text{msync}\ m\ x\ x_1\ i\ i_1$
$\rightarrow \text{mget}\ m\ x_1\ i_1\ v \rightarrow \text{mget}\ m\ x\ i\ v;$
$\text{Udefault\_v2} : \forall\ i\ i_2\ v,$
$(\neg(\exists\ i_1,\ \text{msync}\ m\ x\ x_1\ i\ i_1)$
$\rightarrow \text{msync}\ m\ x\ x_2\ i\ i_2$
$\rightarrow \text{mget}\ m\ x_2\ i_2\ v \rightarrow \text{mget}\ m\ x\ i\ v$
}.

In addition, we apply these semantics rules to a process to get a complete semantics, that is Uprocess. We also give the semantics of processes composition.

Fixpoint Uprocess $\{M\}(p : \text{Process})(m : \text{mdom } M)$
: **Prop** :=
match p **with**
Pass Ind $f\ x\ x_i \Rightarrow$ Uassignment m Ind $f\ x\ x_i$
| Pdelay $x\ x_1\ c \Rightarrow$ Udelay $m\ x\ x_1\ c$
| Pwhen $x\ x_1\ x_2 \Rightarrow$ Uwhen $m\ x\ x_1\ x_2$
| Pdefault $x\ x_1\ x_2 \Rightarrow$ Udefault $m\ x\ x_1\ x_2$
| Ppar $p_1\ p_2 \Rightarrow$ Uprocess $p_1\ m$
$\wedge$Uprocess $p_2\ m$
end

Thirdly, we give a general mapping between two intermediate models (M_1 to M_2). We use a function s_1 to s_2 to express the mapping from a set of elements of the domain of M_1 (denoted as S_1) to a set of elements of the domain of M_2. It relies on a function m_2 to m_1 mapping one element of the domain of M_2 to one element of the domain of M_1, such as from one trace to one behavior on a tag structure.

$$s_1 \text{ to } s_2(S_1) = \{e_2 \in \text{mdom}(M_2) | m_2 \text{ to } m_1(e_2) \in S_1\}$$

get12 and sync12 define the properties of m_2 to m_1, i.e., the same variable of two models has the same value at the same value index (same mget), and has the same synchronous relations (same msync).

Record M_1 to $M_2\ M_1\ M_2$: **Type** := {
m_2 to m_1 : mdom $M_2 \rightarrow$ mdom M_1;
get12 : $\forall\ m_2\ x\ i\ v$, mget $m_2\ x\ i\ v$
$\leftrightarrow$ mget (m_2 to $m_1\ m_2$) $x\ i\ v$;
sync12 : $\forall\ m_2\ x_1\ x_2\ i_1\ i_2$,
msync $m_2\ x_1\ x_2\ i_1\ i_2$
$\leftrightarrow$ msync (m_2 to $m_1\ m_2$) $x_1\ x_2\ i_1\ i_2$;
s1tos2 : (mdom $M_1 \rightarrow$ **Prop**) $\rightarrow$ (mdom $M_2 \rightarrow$ **Prop**)
:= fun $s_1 \Rightarrow$ fun $e_2 \Rightarrow s_1$ (m_2 to $m_1\ e_2$)
}.

Moreover, a basic theorem in which two intermediate models are equivalent is proven. This theorem states that the transformation of the M_2 semantics of a SIGNAL process p is the M_1 semantics of p.

Theorem TFR12 :
$\forall\ M_1\ M_2\ (p : \text{Process})(\text{t12} : M_1 \text{ to } M_2\ M_1\ M_2)$,
$\forall\ (m_2 : \text{mdom } M_2)$, Uprocess ($M := M_2$) $p\ m_2$
$\leftrightarrow s_1$ to s_2 t12 (Uprocess ($M := M_1$) p) m_2.

6.2 The relation between the trace semantics and the intermediate model

Notice that, the semantics defined by intermediate model (Uprocess) is generic, because mget and msync are abstract observers. Here, we focus on the relation between the trace semantics and the intermediate model, so we set the domain as a trace. The observers mget and msync also need to be refined, that are trGet and trSync.

The predicate trGet tr $i\ x\ v$ is satisfied if the i^{th} non-absent value of x is v.

Inductive trGet : Trace $\rightarrow$ nat $\rightarrow$ XVar
$\rightarrow$ Value $\rightarrow$ **Prop** :=
trg0 : $\forall\ x\ st$ tr v, $st\ x$ = Val v
$\rightarrow$ trGet (Tr st tr) 0 $x\ v$
| trgU : $\forall\ i\ x\ st$ tr v, $st\ x$ = Absence
$\rightarrow$ trGet tr $i\ x\ v$
$\rightarrow$ trGet (Tr st tr) $i\ x\ v$
| trgN : $\forall\ i\ x\ st$ tr v, $st\ x \neq$ Absence
$\rightarrow$ trGet tr $i\ x\ v$
$\rightarrow$ trGet (Tr st tr)($S\ i$) $x\ v$.

In order to define trSync, we introduce the auxiliary predicate trGetp. trGetp tr $i\ x\ j$ is satisfied if the i^{th} non-absent value of x is at the instant j of the trace tr.

Inductive trGetp : Trace $\rightarrow$ nat $\rightarrow$ XVar
$\rightarrow$ nat $\rightarrow$ **Prop** :=
trgp0 : $\exists\ x\ st$ tr, $st\ x \neq$ Absence
$\rightarrow$ trGetp (Tr st tr) 0 x 0
| trgpU : $\forall\ i\ x\ st$ tr j, $st\ x$ = Absence
$\rightarrow$ trGetp tr $i\ x\ j$
$\rightarrow$ trGetp (Tr st tr) $i\ x$ ($S\ j$)
| trgpN : $\forall\ i\ x\ st$ tr j, $st\ x \neq$ Absence
$\rightarrow$ trGetp tr $i\ x\ j$
$\rightarrow$ trGetp (Tr st tr)($S\ i$) x ($S\ j$).

Then, we say that x_1 and x_2 synchronize at value index i_1 and i_2 if the i_1^{th} non-absent value of x_1 and the i_2^{th} non-absent value of x_2 occur at the same instant.

Definition trSync $x_1\ x_2$ (tr : Trace)($i_1\ i_2$: nat)
: **Prop** :=
$\forall\ j$, trGetp tr $i_1\ x_1\ j \leftrightarrow$ trGetp tr $i_2\ x_2\ j$.

We construct the corresponding intermediate model instance using the observers trGet and trSync.

Definition strInstance : Model :=
{|
mdom := Trace;
mget tr $x\ i\ v$:= trGet tr $i\ x\ v$;
msync tr $x_1\ x_2\ i_1\ i_2$:= trSync $x_1\ x_2$ tr $i_1\ i_2$
|}.

Finally, we prove the equivalence between the trace semantics and its corresponding intermediate model instance.

Theorem Ssem_def1 : $\forall$ p tr,
straces (Process2Sprocess p) tr
$\rightarrow$ Uprocess (M := strInstance) p tr.

Theorem Ssem_def2 : $\forall$ p tr,
Uprocess (M := strInstance) p tr
$\rightarrow$ straces (Process2Sprocess p) tr.

Example 7 We construct the intermediate model instance of the trace tr_1 shown in the Example 5 (see Fig. 7).

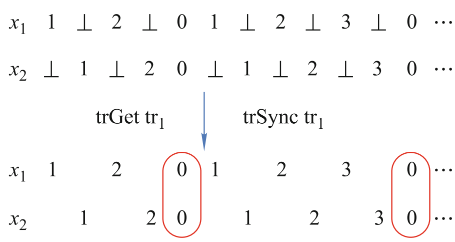

Fig. 7 The intermediate model instance of a trace

- trGet tr_1 = $\{(0, x_1, 1), (1, x_1, 2), (2, x_1, 0), (3, x_1, 1), \ldots, (0, x_2, 1), (1, x_2, 2), (2, x_2, 0), (3, x_2, 1), \ldots\}$
- trSync tr1 = $\{(x_1, x_2, 2, 2), (x_1, x_2, 6, 6), \ldots\}$

6.3 The relation between the tagged model semantics and the intermediate model

Here, we set the domain as a behavior on a tag structure. The observers mget and msync are refined as tGet and tSync.

In order to define tGet and tSync, we introduce the auxiliary predicates tGett_from and tGett. tGett s i t is satisfied if the i^{th} tag of the signal s is t.

Inductive tGett_from $\{G\}$(t_0 : Tag G) :
Tsignal_from t_0 $\rightarrow$ nat $\rightarrow$ Tag G $\rightarrow$ **Prop** :=
tgtn0 : $\forall$ t_1 h d s t, $t = t_1$
$\rightarrow$ tGett_from t_0 (Tnext t_0 t_1 h d s) 0 t
| tgtnS : $\forall$ t_1 h d s i t,
tGett_from t_1 s i t $\rightarrow$
tGett_from t_0 (Tnext t_0 t_1 h d s)(S i) t.

Inductive tGett $\{G\}$: Tsignal G $\rightarrow$ nat
$\rightarrow$ Tag G $\rightarrow$ **Prop** :=
tgt0 : $\forall$ d t s, tGett (Tfrom G t d s) 0 t
| tgtS : $\forall$ t_0 d s i t, tGett_from t_0 s i t $\rightarrow$
tGett (Tfrom G t_0 d s)(S i) t.

The predicate tGet s i v is satisfied if the value on the i^{th} tag of the signal s is v.

Inductive tGet $\{G\}$ s i v : **Prop** :=
tGet_prf : $\forall$ t : Tag G, tGett s i t
$\rightarrow$ tval s t v $\rightarrow$ tGet s i v.

Then, we say that x_1 and x_2 synchronize at tag index i_1 and i_2 if they share the same tag.

Inductive tSync $\{G\}$ x_1 x_2 (b : Tbehavior G)
i_1 i_2 : **Prop** :=
tSyncPrf : ($\forall$ t, tGett(b x_1) i_1 t
$\leftrightarrow$ tGett (b x_2) i_2 t)
$\rightarrow$ tSync x_1 x_2 b i_1 i_2.

We construct the corresponding intermediate model instance using the observers tGet and tSync.

Definition tagInstance G : Model :=
{|
mdom := Tbehavior G;
mget b x i v := tGet (b x) i v;
msync b x_1 x_2 i_1 i_2 := tSync x_1 x_2 b i_1 i_2
|}.

Finally, we prove the equivalence between the tagged model semantics and its corresponding intermediate model instance.

Theorem Tsem_def1 : $\forall$ G p b,
tbehaviors (Process2Tprocess G p) b
$\rightarrow$ Uprocess (M := tagInstance G) p b.

Theorem Tsem_def2 : $\forall$ G p b,
Uprocess (M := tagInstance G) p b
$\rightarrow$ tbehaviors (Process2Tprocess G p) b.

Example 8 We construct the intermediate model instance of the tag structure G_1 shown in the Example 6 (see Fig. 8).

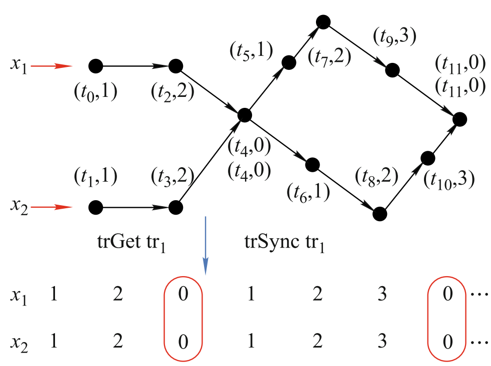

Fig. 8 The intermediate model instance of a tag structure

- tGet G_1 = $\{(x_1, 0, 1), (x_1, 1, 2), (x_1, 2, 0), (x_1, 3, 1), \ldots, (x_2, 0, 1), (x_2, 1, 2), (x_2, 2, 0), (x_2, 3, 1), \ldots\}$
- tSync G_1 = $\{(x_1, x_2, 2, 2), (x_1, x_2, 6, 6), \ldots\}$

6.4 The equivalence between the trace semantics and the tagged model semantics

We refine the definition of mapping (M_1 to M_2) as m_str2tag and m_tag2str. In other words, m_str2tag and m_tag2str are defined as instances of M_1 to M_2.

In m_str2tag, the function m_2 to m_1, i.e., from a behavior on a tag structure to a trace, is constructed by a mathematical transformation (Transformation 1) which is close to the topological sort algorithm [20], and it is used in the definition of the function s_1 to s_2, i.e., from the set of traces to a set of behaviors.

Lemma m_str2tag :
$\forall\, G,\ M_1$ to M_2 strInstance (tagInstance G).

Definition Sprocess2Tprocess G (p : Sprocess) :=
{|
tdom := sdom p;
tbehaviors := s_1 to s_2 (m_str2tag G)(straces p)
|}.

Transformation 1 Let us consider the mapping from a behavior on a tag structure to a trace. It must visit the tags of each signal following their chain order and must be fair(all the tags of all the signals must be eventually visited). For that, we use a variant of topological sort algorithm and the finiteness of the set signal variables.

- **Step 0**: consider the first tag of each signal, i.e., the tag index on each signal is 0, denoted as the vector of tag indexes: $\begin{bmatrix}0\\0\\\vdots\\0\end{bmatrix}$.
- **Step 1**: select any signal such as:
  – no other signal will synchronize in the strict future with its current position.
  – it has a minimal index compared to indexes of such signals.
- **Step 2**: get the current tag of the chosen signal.
- **Step 3**: add to the target trace the values of the signal variables for that tag, while the values of other signals variables are noted $\perp$.
- **Step 4**: increment the index of all the signals of which current tag is the chosen tag, namely their tag index will be added 1, for example $\begin{bmatrix}1\\0\\\vdots\\0\end{bmatrix}$.
- **Step 5**: repeat Step 1, Step 2, Step 3 and Step 4.

The transformation stops if there does not exist any variables with an associated tag at its current tag index. In this case, the resulting trace is finite. Otherwise, the transformation builds an infinite trace.

Example 9 According to Transformation 1, the tag structure G_1 in the Example 6 can be mapped to a set of traces (different arrangement of values), and the trace tr_1 shown in the Example 5 belongs to this set (see Fig. 9).

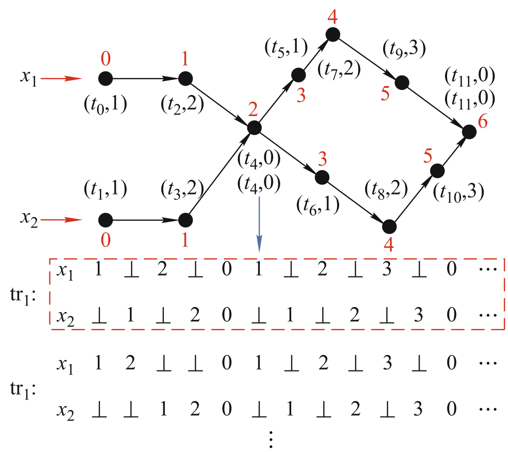

Fig. 9 Mapping from a tag structure to a trace

The tag index on each signal is noted on the tag structure explicitly. The transitions of the vector of tag indexes of tr_1 and tr_2 are given respectively as follows.

$$\begin{bmatrix}0\\0\end{bmatrix}\to\begin{bmatrix}1\\0\end{bmatrix}\to\begin{bmatrix}1\\1\end{bmatrix}\to\begin{bmatrix}2\\1\end{bmatrix}\to\begin{bmatrix}2\\2\end{bmatrix}\to\begin{bmatrix}3\\3\end{bmatrix}\to\begin{bmatrix}4\\3\end{bmatrix}\\\to\begin{bmatrix}4\\4\end{bmatrix}\to\begin{bmatrix}5\\4\end{bmatrix}\to\begin{bmatrix}5\\5\end{bmatrix}\to\begin{bmatrix}6\\5\end{bmatrix}\to\begin{bmatrix}6\\6\end{bmatrix}\to\emptyset$$

$$\begin{bmatrix}0\\0\end{bmatrix}\to\begin{bmatrix}1\\0\end{bmatrix}\to\begin{bmatrix}2\\0\end{bmatrix}\to\begin{bmatrix}2\\1\end{bmatrix}\to\begin{bmatrix}2\\2\end{bmatrix}\to\begin{bmatrix}3\\3\end{bmatrix}\to\begin{bmatrix}4\\3\end{bmatrix}\\\to\begin{bmatrix}4\\4\end{bmatrix}\to\begin{bmatrix}5\\4\end{bmatrix}\to\begin{bmatrix}5\\5\end{bmatrix}\to\begin{bmatrix}6\\5\end{bmatrix}\to\begin{bmatrix}6\\6\end{bmatrix}\to\emptyset$$

In m_tag2str, the function m_2 to m_1, i.e., from a trace to a behavior on a tag structure, is constructed by another mathematical transformation (Transformation 2), and it is used in the definition of the function s_1 to s_2, i.e., from a set of behaviors to a set of traces.

Lemma m_tag2str :
  $\forall\, G,\ M_1$ to M_2 (tagInstance G) strInstance.

Definition Tprocess2Sprocess G (p : Tprocess G) :=
{|
  sdom := tdom p;
  straces := s_1 to s_2 (m_tag2str G)(tbehaviors p)
|}.

In order to map the infinite traces on the tag structure, we must suppose that infinite chains exist, one of these chains will be chosen to map all the traces. So, we have the following hypothesis.

Hypothesis 1 A tag structure always has at least an infinite chain.

The Coq definition is given as follows.

CoInductive hasInfiniteChainFrom $\{G\}$
  (t : Tag G) : **Type** :=
  NextTag : $\forall\, t_1,\ t@<t_1$
    $\rightarrow$ hasInfiniteChainFrom t_1
    $\rightarrow$ hasInfiniteChainFrom t.
Inductive hasInfiniteChain G : **Type** :=
  FirstTag : $\forall$ (t : Tag G),
    hasInfiniteChainFrom t
    $\rightarrow$ hasInfiniteChain G.
Hypothesis infch : $\forall\, G$, hasInfiniteChain G.

Transformation 2 Let us consider the mapping from a trace to a behavior on a tag structure. An infinite chain of the target tag structure is noted by the tags $\{t_i \mid i = 0, 1, \ldots\}$ which correspond to instants ($j = 0, 1, \ldots$) of the trace.

- **Step 0**: start from the first instant of the trace, find the first position which has non-absent value, if the position cannot be found, then return an empty chain.
- **Step 1**: note the variable-value pair on the corresponding tag of the infinite chain.
- **Step 2**: from the current position, find the next position which has non-absent value, if the position cannot be found, then return the chain which is ended at the current position.
- **Step 3**: repeat Step 1 and Step 2.

Finally, each signal variable will get a sub-chain.

Example 10 According to Transformation 2, the trace tr_1 shown in the Example 5 is mapped to an infinite chain with non-absent values, which has the same observers tGet and tSync with the tag structure G_1 in the Example 6 (see Fig. 10).

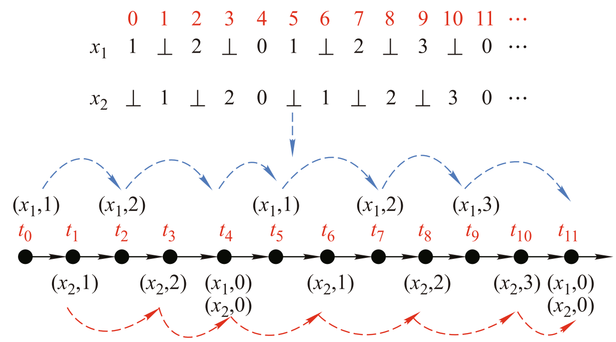

Fig. 10 Mapping from a trace to a tag structure

Finally, we prove the theorems S2Teq and T2Seq based on all the definitions and theorems as above.

In the direction from the trace semantics to the tagged model semantics, we can get a corresponding tag structure using the mapping Sprocess2Tprocess, that is Sprocess2Tprocess G (Process2Sprocess p), then we prove it is equivalent with the tagged model semantics Process2Tprocess, namely, they have the same observers tGet and tSync.

Record TPeq $\{G$: TAG$\}$(p_1 p_2 : Tprocess G) : **Type** := {
  TPd : $\forall\, y$: XVar, tdom p_1 $y \leftrightarrow$ tdom p_2 y;
  TPb : $\forall$ (b_1 : Tbehavior G)(b_2 : Tbehavior G),
    ($\forall\, y,\ b_1\ y = b_2\ y$)
    $\rightarrow$ (tbehaviors p_1 b_1
    $\leftrightarrow$ tbehaviors p_2 b_2)
}.

Theorem S2Teq : $\forall\, G$ (p : Process),
  TPeq (Sprocess2Tprocess G
    (Process2Sprocess p))
    (Process2Tprocess G p).

In the direction from the tagged model semantics to the trace semantics, we can get a corresponding trace using the mapping Tprocess2Sprocess, that is Tprocess2Sprocess G (Process2Tprocess G p), then we prove it is equivalent with the trace semantics Process2Sprocess, namely, they have the same observers trGet and trSync.

Record SPeq (p_1 p_2 : Sprocess) : **Prop** :=
{
  SPd : $\forall\, y$, sdom p_1 $y \leftrightarrow$ sdom p_2 y;
  SPs : $\forall$ tr, straces p_1 tr $\leftrightarrow$ straces p_2 tr

}.

Theorem T2Seq : ∀ G (p : Process),
SPeq (Tprocess2Sprocess G
(Process2Tprocess G p))
(Process2Sprocess p).

6.5 Discussion

As mentioned before, the observers mget and msync are used in the equivalence between two different semantic models. Moreover, local signal variables are ignored in the formal development to get a simplest criterion for comparing models. Here, we discuss the possible properties of mget and msync on the same semantics model, either on the trace semantics or on the tagged model semantics.

Remark 2 The SIGNAL semantics is not closed for mget/msync equivalence when the SIGNAL programs have local declarations, as explained in the following example.

Example 11 Let us consider another process Sampler:

```
process Sampler = (! integer x1, x2; )
 (| y  := not y $ init true
  | x1 := 1 when y
  | x2 := 2 when not y
  |) where boolean y;
end;
```

The trace model is considered here. Similarly, we just consider the external visible signals. We give two traces having the same observers mget and msync. However, tr_1 belongs to the trace semantics of Sampler, while tr_2 does not. The initial value of the local variable y is true, so x_1 should always get values at first.

$$\mathrm{tr}_1: \begin{array}{l} x_1 \ 1 \ \bot \ 1 \ \bot \ 1 \ \bot \cdots \\ x_2 \ \bot \ 2 \ \bot \ 2 \ \bot \ 2 \cdots \end{array}$$

$$\mathrm{tr}_2: \begin{array}{l} x_1 \ \bot \ 1 \ \bot \ 1 \ \bot \ 1 \cdots \\ x_2 \ 2 \ \bot \ 2 \ \bot \ 2 \ \bot \cdots \end{array}$$

Remark 3 The SIGNAL semantics is closed for mget/msync equivalence when the SIGNAL programs do not have local declarations, because the semantic constraints are expressed only through mget and msync.

So, we should not confuse the property of the observers mget and msync with the property of stretch closure.

7 Related work

The formal semantics of the SIGNAL language has a longtime research, and the contributors describe the semantics using different models. The reference manual of SIGNAL V4 [9] gives the definitions of event and trace, and defines the trace semantics. The trace model is a convenient one to be comprehended, so it is always used to interpret the basic concepts of SIGNAL [10, 11, 21]. Lee and Sangiovanni-Vincentelli proposes the tagged-signal model [19] to compare various models of computation, such as Kahn process networks, sequential processes, data flow, event structures, etc. It is a denotational approach where a system is modeled as a set of behaviors. Behaviors are set of events and each event is a value-tag pair. [10] and [12] refine the definitions of event, chain, behavior on tags, and give the tagged model semantics of SIGNAL. [22] introduces an algebra of tag structures, which is a variation of the tagged-signal model, to define parallel composition of heterogeneous reactive systems formally. Morphisms between tag structures can be used to represent design transformations from tightly-synchronized specifications to loosely-synchronized implementation architectures such as loosely time triggered architecture (LTTA) and globally asynchronous locally synchronous (GALS). In [10], they also give a structured operational semantics of SIGNAL through an inductive definition of the set of possible transitions. [13] proposes a synchronous transition systems (STS) model to present the operational semantics of SIGNAL, and presents the translation validation method to verify the compiler from SIGNAL to sequential C-code. [23] defines the properties of endochrony and isochrony on the STS semantics model, to guarantee correct-by-construction deployment from the synchronous programs to GALS.

Meanwhile, there are some work about mechanization of the semantics of the synchronous languages. Nowak proposes a co-inductive semantics for modeling SIGNAL in the Coq proof assistant [14, 15]. In [24], a semantics of Lucid-Synchrone, an extension of LUSTRE with higher-order stream functions, is given in Coq. [25] specifies the semantics of QUARTZ in HOL, and proves the equivalence between different semantics.

However, there has been little research about the equivalence between different semantics of SIGNAL. [14] defines a translation scheme of the trace semantics of SIGNAL to the logical framework of Coq, but they do not consider the semantics equivalence, the stretch-closure property is also excluded. They conduct some case studies to apply the approach SIGNAL-Coq, such as the steam-boiler problem [15], and the correctness of an implementation of SIGNAL protocol for LTTA [26].

8 Conclusion and future work

In this paper, we have studied the equivalence between two denotational semantics of SIGNAL, the trace semantics and the tagged model semantics. The former is easier to be comprehended, so it is often used to explain the basic concepts of SIGNAL. However, the latter can represent the multi-clock and distributed features more naturally. These two semantics have several different definitions respectively. We select appropriate ones as the reference paper semantics and mechanize them in the Coq platform. The distance between these two semantics discourages a direct proof of equivalence. Instead, we have transformed them to an intermediate model, which mixes the features of both the trace semantics and the tagged model semantics. Thus we have established the existence of a bijection between the trace and the tagged semantics domain such that the trace semantics of SIGNAL can be obtained from its tagged model semantics and vice versa. We prove the equivalence between the SIGNAL semantics by introducing two observers mget and msync, which introduces an equivalence relation weaker than the stretching relation. A feedback from our formal development, besides stretch-equivalence, the SIGNAL semantics satisfies the mget/msync equivalence if the SIGNAL programs do not have local declarations.

In the future, we plan to consider the local declarations in the intermediate model. Furthermore, we can use this framework to compare the definitions of SIGNAL properties such as endochrony, isochrony defined on variants of semantics models or on the syntax.

The synchronous hypothesis simplifies system specification and verification, however, the problem of deriving a correct physical implementation from it does remain. In particular, the target architecture has a distributed feature, such as multi-core systems. In order to exploit the emerging multi-core processors, thanks to the theory of weakly endochronous systems [27], there are several research to synthesize multi-threaded code from the synchronous specifications [28, 29]. However, one needs to prove the semantics preservation from the SIGNAL specifications to the multi-threaded code. The results of this paper will be useful for this challenging problem.

Acknowledgements This work was supported in part by the National Natural Science Foundation of China (Grant Nos. 61073013, 61003017), the Aviation Science Foundation of China (2012ZC51025), the TOPCASED Project, and the RTRA STAE Foundation in France.

References

1. Harel D, Pnueli A. On the development of reactive systems. Logics and Models of Concurrent Systems, 1989, F(13): 477–498
2. Potop-Butucaru D, De Simone R, Talpin J P. The synchronous hypothesis and synchronous languages. The Embedded Systems Handbook, 2005, 1–21
3. Boussinot F, De Simone R. The esterel language. Proceedings of the IEEE, 1991, 79(9): 1293–1304
4. Halbwachs N, Caspi P, Raymond P, Pilaud D. The synchronous dataflow programming language lustre. Proceedings of the IEEE, 1991, 79(9): 1305–1320
5. Benveniste A, Le Guernic P, Jacquemot C. Synchronous programming with events and relations: the signal language and its semantics. Science of Computer Programming, 1991, 16: 103–149
6. Schneider K. The synchronous programming language quartz. Internal Report, Department of Computer Science, University of Kaiserslautern, Germany, 2010
7. Teehan P, Greenstreet M, Lemieux G. A survey and taxonomy of gals design styles. IEEE Design and Test of Computers, 2007, 24: 418–428
8. Benveniste A, Caillaud B, Le Guernic P. From synchrony to asynchrony. In: Proceedings of CONCUR 99. 1999, 162–177
9. Besnard L, Gautier T, Le Guernic P. SIGNAL V4 Reference Manual, 2010
10. Gamatié A. Designing Embedded Systems With the SIGNAL Programming Language. Springer, 2010
11. Le Guernic P, Gautier T. Data-flow to von neumann: the signal approach. Advanced Topics in Data-Flow Computing, 1991, 413–438
12. Le Guernic P, Talpin J P, Le Lann J C. Polychrony for system design. Journal of Circuits Systems and Computers, 2002, 12: 261–304
13. Pnueli A, Siegel M, Singerman F. Tanslation validation. In: Proceedings of TACAS'98. 1998, 151–166
14. Nowak D, Beauvais J R, Talpin J P. Co-inductive axiomatization of a synchronous language. In: Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics. 1998, 387–399
15. Kerboeuf M, Nowak D, Talpin J P. Specification and verification of a stream-boiler with signal-coq. In: Proceedings of the 13th International Conference on Theorem Proving in Higher Order Logics. 2000, 356–371
16. Bertot Y, Casteran P. Interactive theorem proving and program development: Coq art: the calculus of inductive constructions. Springer, 2004
17. The polychrony toolset. http://www.irisa.fr/espresso/Polychrony
18. Benveniste A, Le Guernic P, Sorel Y, Sorine M. A denotational theory of synchronous reactive systems. Information and Computation, 1992, 99(2): 192–230
19. Lee E A, Sangiovanni-Vincentelli A. A framework for comparing models of computation. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, 1998, 17(12): 1217–1229

20. Cormen T, Leiserson C, Rivest R, Stein C. Introduction to Algorithms. MIT Press, 2009
21. Houssais B. The synchronous programming language signal-a tutorial. 2004
22. Benveniste A, Caillaud B, Carloni L P, Caspi P, Sangiovanni-Vincentelli A L. Composing heterogeneous reactive systems. ACM Transactions on Embedded Computing Systems, 2008, 7(4): 1–36
23. Benveniste A, Caillaud B, Le Guernic P. Compositionality in dataflow synchronous languages: specification distributed code generation. Information and Computation, 2000, 125–171
24. Boulmé S, Hamon G. Certifying synchrony for free. In: Proceedings of the Artificial Intelligent on Logic for Progamming (LPAR). 2001, 495–506
25. Schneider K. Proving the equivalence of microstep and macrostep semantics. LNCS2410, 2002, 314–331
26. Kerboeuf M, Nowak D, Talpin J P. Formal proof of a polychronous protocol for loosely time-triggerd architectures. In: Proceedings of the 5th International Conference on Formal Engineering Methods, ICFEM 03. 2003, 359–374
27. Potop-Butucaru D, Caillaud B, Benveniste A. Concurrency in synchronous systems. Formal Methods in System Design, 2006, 111–130
28. B.A. J. Formal model driven software synthesis for embedded systems. PhD thesis, Virginia Polytechnic Institute and State Univeristy, 2011
29. Papailiopoulou V, Potop-Butucaru D, Sorel Y, Simone d R, Besnard L, Talpin J P. From design-time concurrency to effective implementation parallelism: the multi-clock reactive case. In: Proceedings of Electronic System Level Synthesis Conference, 2011, 1–6

Dr. Zhibin Yang received his PhD in Computer Science from Beihang University, China in February 2012 . Since April 2012, he has been a Postdoc in IRIT of University of Toulouse, France. His research interests include safety-critical real-time system, formal verification, AADL, synchronous languages.

Dr. Jean-Paul Bodeveix received his PhD of Computer Science from the University of Paris-Sud 11 in 1989. He has been an assistant professor at University of Toulouse III since 1989 and is now a professor of computer science since 2003. His main research interests concern formal specifications, automated and assisted verification of protocols as well as of proof environments. He has participated in European and national projects related to these domains. His current activities are linked to real time modeling and verification either via model checking techniques or at the semantics level.

Dr. Mamoun Filali is a full time researcher at CNRS (Centre National de la Recherche Scientifique). His main research interests concern the certified development of embedded systems. He is concerned by formal methods, model checking, and theorem proving. During the last years, he has been mainly involved in the French Nationwide TOPCASED Project where he was concerned by the verification topic. He has also participated to the proposal of the AADL behavioral annex which has been been adopted as part of the AADL SAE standard.