Verifikation nebenläufiger Programme
Wintersemester 2004/05

Ulrich Hannemann Jan Bredereke

13 Synchronous Transition Diagrams

13.1 Syntax and Semantics of Synchronous Transition Diagrams

We consider synchronous transition diagrams $P_1 \| \ldots \| P_n$ in which the components $P_1, \ldots, P_n$, called processes, do not share variables. These diagrams are inspired by Tony Hoare's language proposal Communicating Sequential Processes [Hoa78]. The processes which constitute those diagrams communicate by means of synchronous message passing along unidirectional channels which connect *at most* two different processes. These components are called sequential synchronous (transition) diagrams and are defined below. The term synchronous diagram is reserved for their parallel composition $P_1 \| \ldots \| P_n$, which includes sequential synchronous diagrams for $n = 1$ as a special case. Unless stated otherwise, primitive boolean and state functions are considered to be total. Let $CHAN$ be a set of channel names, with typical elements $C, D \ldots$. For $C \in CHAN$, e a semantic expression, i.e., $e : \Sigma \to VAL$, where VAL denotes the given underlying domain of values, and x a variable, execution of *output* statement $C!e$ has to wait for execution of a corresponding *input* statement $C?x$, and, similarly, execution of an input statement has to wait for execution of a corresponding output statement. If there exists a computation of $P_1 \| \ldots \| P_n$ in which both an input statement $C?x$ and an output statement $C!e$ are simultaneously executed, this implies that communication can take place and the value of e is assigned to x. Then one speaks of a *semantically-matching communication pair.* When formulating our proof method, we also need the concept of a *syntactically*-matching communication pair. This is a pair consisting of occurrences of an output statement $C!e$ and an input statement $C?x$ which refer to the same communication channel (in this case C), *irrespective of whether these communicate or not.* We often refer to an input or output statement as an *io-statement* or *communication* statement. Labels on edges in P_i can have the following form:

1. A boolean condition b followed by a state transformation f:

   $$\begin{tikzcd} \boxed{l} \arrow[rr, "b \to f"] & & \boxed{l'} \end{tikzcd}$$

   Transitions of this form are called *internal* transitions.

2. A guarded io-statement followed by a state transformation. There are two possibilities:

a)
$$\begin{tikzcd} l \arrow[r, "b; C!e \to f"] & l' \end{tikzcd}$$

b)
$$\begin{tikzcd} l \arrow[r, "b; C?x \to f"] & l' \end{tikzcd}$$

We call these transitions *communication* or *input-output* transitions. We extend the definitions of semantically- and syntactically-matching communication pairs to pairs of such transitions of which the communication statements satisfy the conditions listed above.

We assume that each sequential synchronous diagram P is associated with a set of program variables such that every condition, state transformation, expression, and input statement occurring in P involves only those program variables. We call sequential synchronous diagrams $P_1, \ldots, P_n$ *disjoint* if their associated sets of program variables are mutually disjoint, and every channel occurring in $P_1, \ldots, P_n$ is unidirectional and connects two different processes. Below we define the *closed* product $P_1 \parallel \ldots \parallel P_n$ of such diagrams, in which only the communication capabilities of those io-statements are resolved, concerning channels connecting two processes which both occur amongst $P_1, \ldots, P_n$.

Definition 13.1 (Closed product of synchronous sequential transition diagrams) Given disjoint sequential synchronous transition diagrams $P_1, \ldots, P_n$, with $P_i \equiv (L_i, T_i, s_i, t_i)$, we define the closed product P of $P_1 \parallel \ldots \parallel P_n$ as the following transition diagram (L, T, s, t):

- $L \equiv L_1 \times \ldots \times L_n$ is the set of locations of P,
- $l \xrightarrow{a} l'$ is a transition in T iff
  1. $l = \langle l_1, \ldots, l_i, \ldots, l_n \rangle$ and $l' = \langle l_1, \ldots, l'_i, \ldots, l_n \rangle$, with $l_i \xrightarrow{a} l'_i$ an internal transition of T_i, or
  2. $l = \langle l_1, \ldots, l_i, \ldots, l_j, \ldots, l_n \rangle$ and $l' = \langle l_1, \ldots, l'_i, \ldots, l'_j, \ldots, l_n \rangle$, $i \neq j$, with $(l_i \xrightarrow{a_i} l'_i) \in T_i$, $(l_j \xrightarrow{a_j} l'_j) \in T_j$, $a_i \equiv b; C!e \to f$, $a_j \equiv b'; C?x \to g$, and $a \equiv b \wedge b' \to f \circ g \circ (x := e)$; this is called a communication step. The assignment $x := e$, where $e : \Sigma \mapsto VAL$, for some domain of values VAL, denotes the following state-transformation $(x := e)(\sigma) \overset{\text{def}}{=} (\sigma : x \mapsto e(\sigma))$,
- $s \equiv \langle s_1, \ldots, s_n \rangle$,
- $t \equiv \langle t_1, \ldots, t_n \rangle$.

■

Communication between $a_i \equiv b; C!e \rightarrow f$ and $a_j \equiv b'; C?x \rightarrow g$ is modelled in three stages. First the boolean function $b \wedge b'$ is evaluated; when the result is tt, $x := e$ is executed, and finally f and g are applied in any order, because they operate on disjoint state spaces. Note that the closed parallel composition of sequential synchronous diagrams therefore gives rise to a transition diagram as defined in Session 1, and so the definitions from Session 1 apply for defining the semantics $\mathcal{M}$ of the parallel composition of synchronous transition diagrams, their specifications, partial correctness, φ-convergence, φ-success and total correctness.

We emphasise here that the closed product is a simple transition diagram in which no io-statements occur. As such communications with the outside world are no longer offered. For example, the closed product of $C!0 \parallel D?x$ does not contain any transitions, and models that this system when it is executed on its own will deadlock. In a compositional semantics, however, we will need an *open* interpretation of synchronous transition diagrams. For example, we want to be able to compose $C!0 \parallel D?x$ in parallel with a diagram which offers corresponding inputs and outputs. Therefore an open interpretation of the communications in $C!0 \parallel D?x$ is required in which the communication capabilities of $C!0$ and $D?x$ are postponed, i.e., are not yet resolved. This open interpretation of networks requires the channels to be uni-directional and also one-to-one.

13.2 Proof Methods for Partial Correctness

Partial correctness of a synchronous diagram $P \equiv P_1 \parallel \ldots \parallel P_n$ w.r.t. a specification $< \varphi, \psi >$ can be proved by constructing an inductive assertion network $\{\mathcal{Q}_l \,|\, l \in L\}$ for P (here L denotes the set of locations of P) such that $\models \varphi \rightarrow \mathcal{Q}_s$ and $\models \mathcal{Q}_t \rightarrow \psi$ hold. Analogous to concurrent programs with shared variables this leads to a number of verification conditions which are exponential in the number of parallel processes. Similarly, we try to improve upon this by attaching predicates to local locations. We associate a predicate $\mathcal{Q}_{l_i}$ with every local location l_i in P_i such that $\mathcal{Q}_{l_i}$ does not involve any of the variables of the other components P_j, $j \neq i$. Then with every global location $l \equiv \langle l_1, \ldots, l_n \rangle$ in P the predicate $\mathcal{Q}_l \overset{\text{def}}{=} \mathcal{Q}_{l_1} \wedge \ldots \wedge \mathcal{Q}_{l_n}$ is associated.

This assertion network is shown to be inductive by proving the verification conditions along each verification path. That is, for every transition $l \xrightarrow{a} l'$ of P, with $a \equiv b \rightarrow f$, we should prove

$$\models \mathcal{Q}_l \wedge b \rightarrow \mathcal{Q}_{l'} \circ f.$$

Thus, with $l = \langle l_1, \ldots, l_n \rangle$ and $l' = \langle l'_1, \ldots, l'_n \rangle$, we have to prove

$$\models (\mathcal{Q}_{l_1} \wedge \ldots \wedge \mathcal{Q}_{l_n} \wedge b) \rightarrow (\mathcal{Q}_{l'_1} \wedge \ldots \wedge \mathcal{Q}_{l'_n}) \circ f.$$

Observe that this verification condition can be of two kinds:

1. It stems from an internal transition of P_i. Then $\mathcal{Q}_{l_j} = \mathcal{Q}_{l'_j}$, for $j \neq i$, and thus we have to prove:

   - $\models (\mathcal{Q}_{l_i} \wedge b) \rightarrow \mathcal{Q}_{l'_i} \circ f$, i.e., local correctness.
   - $\models (\mathcal{Q}_{l_j} \wedge \mathcal{Q}_{l_i} \wedge b) \rightarrow \mathcal{Q}_{l_j} \circ f$, for $j \neq i$, i.e., interference freedom. However, since $\mathcal{Q}_{l_j}$ does not involve the variables of P_i, this implication is trivially satisfied.

2. It stems from a communication step between P_i and P_j; so there exist transitions $l_i \xrightarrow{a_i} l'_i$ and $l_j \xrightarrow{a_j} l'_j$ of P_i and P_j, with $a_i \equiv b; C!e \to f$ and $a_j \equiv b'; C?x \to g$ such that $a \equiv b \wedge b' \to f \circ g \circ (x := e)$. Let h denote the transformation $f \circ g \circ (x := e)$. Then, for $k \neq i$, $k \neq j$, we have $\mathcal{Q}_{l_k} = \mathcal{Q}_{l'_k}$ and a does not change the variables of P_k. Thus $\models \mathcal{Q}_{l_k} \to \mathcal{Q}_{l'_k} \circ h$. It remains to prove

$$\models \mathcal{Q}_{l_i} \wedge \mathcal{Q}_{l_j} \wedge b \wedge b' \to (\mathcal{Q}_{l'_i} \wedge \mathcal{Q}_{l'_j}) \circ h.$$

Definition 13.2 (First try to formulate an inductive assertion method for synchronous communication) In order to prove partial correctness of synchronous diagram $P_1 \parallel \ldots \parallel P_n$ w.r.t. $< \varphi, \psi >$ find local predicates $\mathcal{Q}_l$ (associated with the local locations l of P_i and which do not involve any of the variables of the other components) and prove:

1. the local verification conditions of P_i w.r.t. $\{\mathcal{Q}_l \mid l \textit{ is a location of } P_i\}$,
2. for every pair of syntactically-matching input-output transitions $l_i \xrightarrow{a_i} l'_i$ of P_i and $l_j \xrightarrow{a_j} l'_j$ of P_j, with $a_i \equiv b; C!e \to f$ and $a_j \equiv b'; C?x \to g$:

   $$\models \mathcal{Q}_{l_i} \wedge \mathcal{Q}_{l_j} \wedge b \wedge b' \to (\mathcal{Q}_{l'_i} \wedge \mathcal{Q}_{l'_j}) \circ h,$$

   where $h \overset{\text{def}}{\equiv} f \circ g \circ (x := e)$,
3. $\models \varphi \to \mathcal{Q}_{s_1} \wedge \ldots \wedge \mathcal{Q}_{s_n}$, with s_i the initial location of P_i, and $\models \mathcal{Q}_{t_1} \wedge \ldots \wedge \mathcal{Q}_{t_n} \to \psi$, with t_i the final location of P_i.

■

Observe that in the above we proved inductiveness of the global assertion network

$$\{\mathcal{Q}_l | l \in L_1 \times \ldots \times L_n\},$$

and, hence, by soundness of the inductive assertion method (Theorem 4.1), also soundness of the method of Definition 13.2.

The second requirement in Definition 13.2 corresponds to the *cooperation test* from the proof system of Apt, Francez & de Roever [AFdR80] for CSP. Their methodology consists essentially of two phases. In the first phase, covered by the first requirement, the local steps of the processes are proved correct and nothing is verified for the communication actions. This is called proving *local correctness*. In the second phase the communication actions are verified. This raises a problem, since we have no syntactic means at our disposal to determine which syntactically-matching pairs actually do communicate (recall that a *syntactically-matching pair of communication transitions* is a pair of occurrences of transitions with labels $b_1; C!e \to f_1$ and $b_2; C?x \to f_2$, irrespective of whether these transitions communicate or not). For we only need to prove something about actually occurring communication steps. Hence we shall resort to assertional means to express which pairs are *semantically matching*, i.e., do actually communicate, and which pairs do not. The difference between syntactically- and semantically-matching pairs is illustrated in the following example.

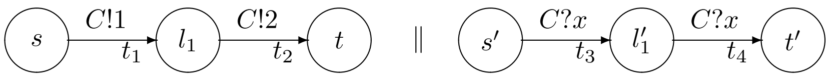


Figure 1: Illustrating the difference between syntactically- and semantically-matching pairs.

Example 13.3 Consider Figure 1 where t_1, t_2, t_3 and t_4 are names of the transitions. There are four syntactically-matching pairs: (t_1, t_3), (t_1, t_4), (t_2, t_3) and (t_2, t_4), whereas only two of them match semantically, namely, (t_1, t_3) and (t_2, t_4).

■

Observe that the cooperation test gives a verification condition for every syntactically-matching pair of communication transitions, although some of them may not match semantically.

Next we illustrate a case where logical variables are needed to express local assertions.

Example 13.4 (The need for logical variables) Consider $R \equiv R_1 \parallel R_2$ as in Figure 2.

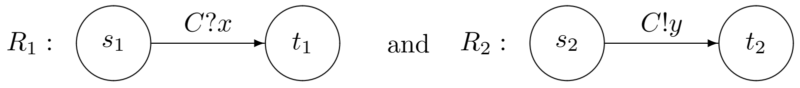
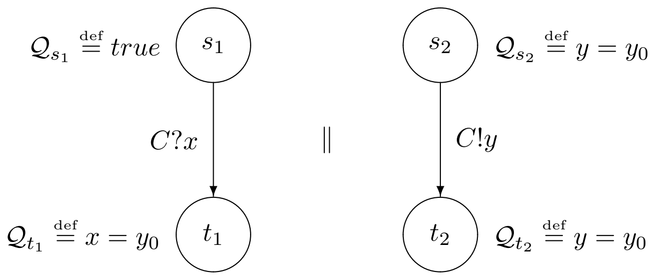


Figure 2: A simple synchronous diagram.

We would like to prove $\models \{true\}\ R_1 \| R_2\ \{x = y\}$. Due to the restriction that the assertion network $\{\mathcal{Q}_{s_1}, \mathcal{Q}_{t_1}\}$ should not involve the variables of R_2, i.e., y, one cannot choose $\mathcal{Q}_{t_1}$ as $x = y$. However, this choice is not forced upon us, according to Definition 13.2. All one needs is to find predicates $\mathcal{Q}_{t_1}$ and $\mathcal{Q}_{t_2}$ such that $\models \mathcal{Q}_{t_1} \wedge \mathcal{Q}_{t_2} \rightarrow x = y$. These can be found using logical variables, i.e., variables not occurring in any program text and therefore not involved in any process; they are introduced in Session 2. Let y_0 be a logical variable, then our local assertions can be chosen as in Figure 3 below.



$\mathcal{Q}_{s_1} \overset{\text{def}}{=} true$
s_1
$C?x$
t_1
$\mathcal{Q}_{t_1} \overset{\text{def}}{=} x = y_0$
$\|$
s_2
$\mathcal{Q}_{s_2} \overset{\text{def}}{=} y = y_0$
$C!y$
t_2
$\mathcal{Q}_{t_2} \overset{\text{def}}{=} y = y_0$

Figure 3: Program $R_1 \| R_2$ plus associated assertion network.

Note that $\{\mathcal{Q}_{s_1}, \mathcal{Q}_{t_1}\}$ does not involve any program variable of R_2, and neither does $\{\mathcal{Q}_{s_2}, \mathcal{Q}_{t_2}\}$ involve any program variable of R_1. Next, we prove that $\models \{true\}\ R_1\|R_2\ \{x = y\}$ holds:

- There are no local verification conditions to be checked.
- There is only one cooperation test, namely

$$\models\ (true \wedge y = y_0)\,(x, y) \to (x = y_0 \wedge y = y_0)(y, y),$$

  which holds, because $\models y = y_0 \to y = y_0$.
- Point (iii) of Definition 13.2 raises a problem since $\not\models true \to true \wedge y = y_0$. Note that we do have that $\models x = y_0 \wedge y = y_0 \to x = y$ holds. However, y_0 being a logical variable allows application of the initialisation Rule 9.1 of Session 9, since any set of logical variables satisfies Definition 9.1 (because it is a set of auxiliary variables for any program P). Consequently, by the initialisation rule $\{y = y_0\}\ R_1\|R_2\ \{x = y\}$ implies

$$\{y = y_0 \circ f\}\ R_1\|R_2\ \{x = y\}$$

  for $f(\sigma) \overset{\text{def}}{=} (\sigma : y_0 \mapsto \sigma(y))$, which implies $\{true\}\ R_1\|R_2\ \{x = y\}$.

■

References

[AFdR80] K.R. Apt, N. Francez, and W. P. de Roever. A proof system for communicating sequential processes. *ACM Transactions on Programming Languages and Systems*, 2:359–385, 1980.

[Hoa78] C.A.R. Hoare. Communicating sequential processes. *CACM*, 21(8):666–677, 1978.