Notes on specifying systems in EST

Robert Meolic, Tatjana Kapus
Faculty of Electrical Engineering and Computer Science
University of Maribor
Smetanova ulica 17, SI-2000 Maribor, Slovenia
{meolic,kapus}@uni-mb.si

Abstract

EST is a tool for formal verification of systems. A system to be verified should be specified in a CCS-like syntax. CCS has been chosen because it is widely used in theoretical approaches due to a relatively small set of operators and nice laws valid for them. In this paper we make an overview of operators supported by EST. They can be classified into two groups: standard CCS operators and additional operators which we introduced to shorten specifications and to facilitate traslations from other formalisms.

1 Introduction

EST (Efficient Symbolic Tools [4]) is a collection of verification tools. Among others it includes an equivalence checker and an ACTLW model checker. It is oriented toward action-based formalisms, and thus each system is internally represented as a labelled transition system (LTS). EST also includes a parser for CCS-like specifications. The aim of this parser is to read the specification and create an adequate LTS. In general, a given CCS-like specification has more than one adequate LTS, and from practical reasons it is best to create the smallest one. However, this paper does not address this problem.

In Section 2 we describe standard CCS operators supported by EST and present a simple example of system specification. In Section 3 we define additional operators and present another example of specification in EST demonstrating their use. In Conclusion we give some remarks.

2 Standard CCS operators

An LTS $\mathcal{M}$ is a quadruple $(\mathcal{S}, \mathcal{A}_\tau, \delta, s_0)$ where:

- $\mathcal{S}$ is a non-empty set of states;
- $\mathcal{A}_\tau$ is a non-empty set of actions containing observable actions and an unobservable action τ;
- $\delta \subseteq \mathcal{S} \times \mathcal{A}_\tau \times \mathcal{S}$ is the transition relation;
- s_0 is the initial state.

Set $\mathcal{A}_\tau$ is called an alphabet of LTS $\mathcal{M}$. EST requires that there exists a mapping between actions in $\mathcal{A}_\tau$ such that each observable action $a \in \mathcal{A}_\tau$ has a coaction $\bar{a} \in \mathcal{A}_\tau$ and $\bar{\bar{a}} = a$. Moreover, EST requires that the labels for action and its coaction differ only in the first character which is called a prefix and should be either "?" or "!". If the first character in the label of an observable action a is deleted we get an action name, denoted by $\hat{a}$. For example, actions `!stop` and `?stop` have both action name `stop`. A triple $(s, a, s') \in \delta$ is called an *a-transition* or shortly a *transition* from state s to state s'. If there exists $(s, a, s') \in \delta$, then we may also say that the LTS can *execute* an action a in state s.

CCS (Calculus of Communicating Systems, [5]) is a process calculus. Due to the nice laws valid for its operators it is also classiffied as process algebra. Each CCS expression defines a process (also called an agent). CCS operators supported by EST are:

- Prefix (.),
- Summation (+),
- Composition (|),
- Restriction (\), and
- Relabelling ([ ]).

To define these operators, we recall rules from [5]. We suppose that $\mathcal{M}$, $\mathcal{M}_1$, and $\mathcal{M}_2$ are LTSs while a and b are actions.

Operator Prefix is defined with the following rule:

$$\textbf{Prefix}\ \frac{}{a.\mathcal{M} \xrightarrow{a} \mathcal{M}}$$

This rule expresses that process $a.\mathcal{M}$ can execute action a and afterwards behave as process $\mathcal{M}$. This means that the initial state of the LTS representing process $a.\mathcal{M}$ has an a-transition to the initial state of the LTS representing process $\mathcal{M}$.

Operator Summation is defined with the following two rules:

$$\textbf{Sum1}\ \frac{\mathcal{M}_1 \xrightarrow{a} \mathcal{M}_1'}{(\mathcal{M}_1 + \mathcal{M}_2) \xrightarrow{a} \mathcal{M}_1'}$$

$$\textbf{Sum2}\ \frac{\mathcal{M}_2 \xrightarrow{a} \mathcal{M}_2'}{(\mathcal{M}_1 + \mathcal{M}_2) \xrightarrow{a} \mathcal{M}_2'}$$

PREPRINT for ERK'2006, September 25-27, 2006, Portorož, Slovenia

These rules state that if any process participating in the summation can execute action a, then the sum can also execute action a. This means that the initial state of the LTS representing process $\mathcal{M}_1 + \mathcal{M}_2$ has exactly those transitions which are present in the initial states of the LTSs representing processes $\mathcal{M}_1$ and $\mathcal{M}_2$.

Operator Composition is defined as follows:

$$\textbf{Com1}\ \frac{\mathcal{M}_1 \xrightarrow{a} \mathcal{M}_1'}{(\mathcal{M}_1|\mathcal{M}_2) \xrightarrow{a} (\mathcal{M}_1'|\mathcal{M}_2)}$$

$$\textbf{Com2}\ \frac{\mathcal{M}_2 \xrightarrow{a} \mathcal{M}_2'}{(\mathcal{M}_1|\mathcal{M}_2) \xrightarrow{a} (\mathcal{M}_1|\mathcal{M}_2')}$$

$$\textbf{Com3}\ \frac{\mathcal{M}_1 \xrightarrow{a} \mathcal{M}_1' \quad \mathcal{M}_2 \xrightarrow{\bar{a}} \mathcal{M}_2'}{(\mathcal{M}_1|\mathcal{M}_2) \xrightarrow{\tau} (\mathcal{M}_1'|\mathcal{M}_2')}\ (a \neq \tau)$$

The initial state of the LTS representing process $\mathcal{M}_1|\mathcal{M}_2$ has all those transitions which are present in the initial states of the LTSs representing processes $\mathcal{M}_1$ and $\mathcal{M}_2$ and one τ-transition for each a-transition starting in the initial state of the LTS representing process $\mathcal{M}_1$ for which an $\bar{a}$-transition exists in the initial state of the LTS representing process $\mathcal{M}_2$.

In EST, operator Restriction is defined as follows:

$$\textbf{Res}\ \frac{\mathcal{M} \xrightarrow{a} \mathcal{M}'}{(\mathcal{M}\backslash\hat{b}) \xrightarrow{a} (\mathcal{M}'\backslash\hat{b})}\ (b \neq \tau, a = \tau \vee \hat{a} \neq \hat{b})$$

States in the LTS representing process $\mathcal{M}\backslash\hat{b}$ have exactly those transitions which are present in the LTS reperesenting process $\mathcal{M}$ and are labelled with an action whose action name is not equal to $\hat{b}$.

Finally, we define operator Relabelling. Let $\mathcal{A}_\tau$ be the alphabet of process $\mathcal{M}$ and let $f : \mathcal{A}_\tau \longrightarrow \mathcal{A}_\tau$ be a relabelling function such that $f(\bar{a}) = \overline{f(a)}$ and $f(\tau) = \tau$. Then:

$$\textbf{Rel}\ \frac{\mathcal{M} \xrightarrow{a} \mathcal{M}'}{\mathcal{M}[f] \xrightarrow{f(a)} \mathcal{M}'[f]}$$

As stated by the rule, an LTS representing process $\mathcal{M}[f]$ can be obtained from LTS representing process $\mathcal{M}$ by changing all transition labels according to the relabelling function f. In EST, a relabelling function is given as a pair of action names and thus it can rename only one action. If more than one action is to be relabelled, a chain of relabelling operations is used.

The equivalence of specifications is defined by equivalence of the processes they define. We take that two specifications are equivalent only if the processes they define are strongly equivalent [5]. The following basic laws for defined operators are consistent with this presumption:

- $\mathcal{M}_1 + \mathcal{M}_2 = \mathcal{M}_2 + \mathcal{M}_1$
- $(\mathcal{M}_1 + \mathcal{M}_2) + \mathcal{M}_3 = \mathcal{M}_1 + (\mathcal{M}_2 + \mathcal{M}_3)$
- $\mathcal{M}_1|\mathcal{M}_2 = \mathcal{M}_2|\mathcal{M}_1$
- $(\mathcal{M}_1|\mathcal{M}_2)|\mathcal{M}_3 = \mathcal{M}_1|(\mathcal{M}_2|\mathcal{M}_3)$
- $\mathcal{M}\backslash\hat{a}\backslash\hat{b} = \mathcal{M}\backslash\hat{b}\backslash\hat{a}$

Moreover, operator Composition is distributive over operator Summation:

$$\mathcal{M}_1 \mid (\mathcal{M}_2 + \mathcal{M}_3) = (\mathcal{M}_1|\mathcal{M}_2) + (\mathcal{M}_1|\mathcal{M}_3)$$

The opposite is not true. Also, the operator Prefix is distributive neither over operator Summation nor over operator Composition:

- $\mathcal{M}_1 + (\mathcal{M}_2|\mathcal{M}_3) \neq (\mathcal{M}_1 + \mathcal{M}_2) \mid (\mathcal{M}_1 + \mathcal{M}_3)$
- $a.(\mathcal{M}_1 + \mathcal{M}_2) \neq a.\mathcal{M}_1 + a.\mathcal{M}_2$
- $a.(\mathcal{M}_1|\mathcal{M}_2) \neq a.\mathcal{M}_1 \mid a.\mathcal{M}_2$

The commutativity and associativity of the operators enable us to omit parentheses when the same operator is used sucessively two or more times. As an example, let us consider the classic Dining philosophers problem. Action names in this system are `think`, `eat`, `takeleft`, `takeright`, `dropleft`, and `dropright`. Regarding the number of philosophers the specification also includes action names `think1`, `think2`, ...,`eat1`, `eat2`, ..., `take1`, `take2`, ..., and `drop1`, `drop2`, ... Here is a complete EST specification of the system with two philosophers:

```
PHILO = !think.
        !takeleft.!takeright.
        !eat.
        !dropleft.!dropright.PHILO

FORK = ?take.?drop.FORK

DINNER = (
    PHILO [think1/think]
          [take2/takeleft]
          [take1/takeright]
          [eat1/eat]
          [drop2/dropleft]
          [drop1/dropright]
  | FORK  [take1/take]
          [drop1/drop]
  | PHILO [think2/think]
          [take1/takeleft]
          [take2/takeright]
          [eat2/eat]
          [drop1/dropleft]
          [drop2/dropright]
  | FORK  [take2/take]
          [drop2/drop]
)\take1\drop1\take2\drop2
```

The name of the process representing the system is DINNER. It is composed of two instances of process PHILO and two instances of process FORK. To get a

proper composition of processes, operators Relabelling and Restriction have to be used. Figures 1 and 2 show the LTSs generated by EST for the given specification.

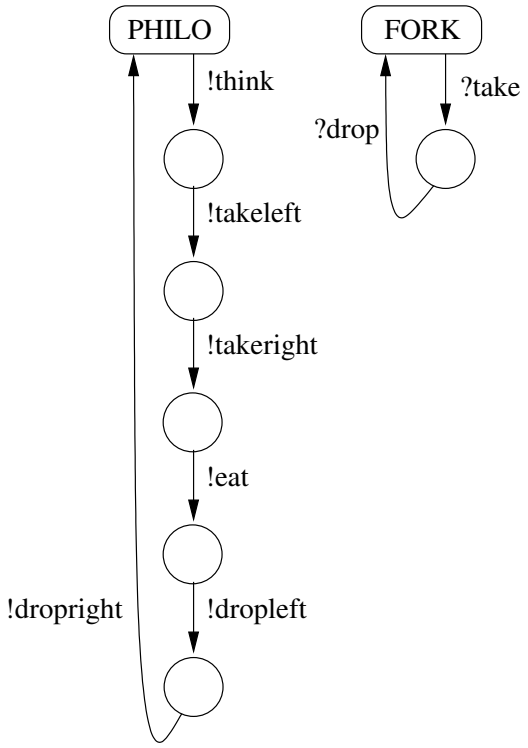

Figure 1: LTSs representing a philosopher and a fork

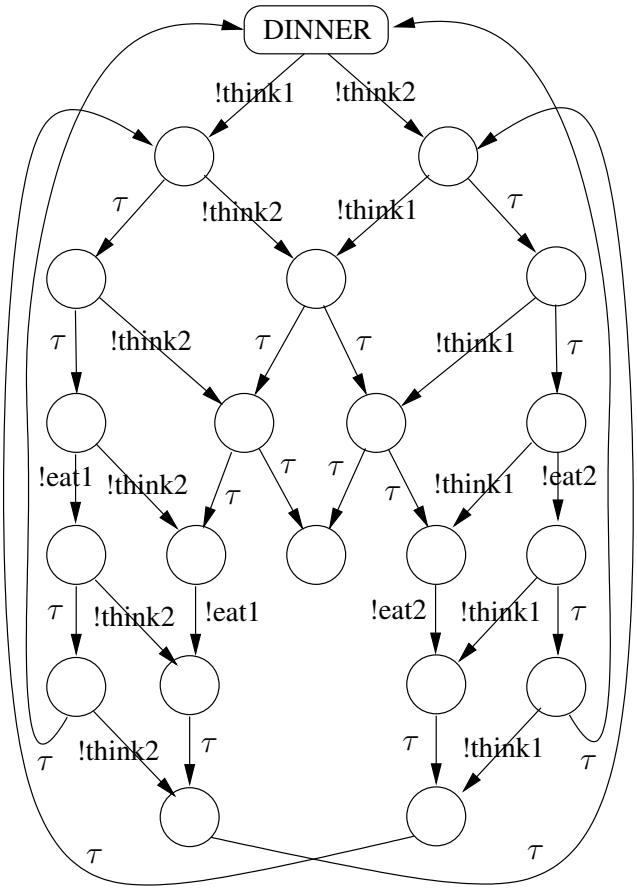

Figure 2: An LTS of the system with two philosophers

3 Additional operators

CCS is quite a simple formalism and it is not very practical for specification of real systems. To enhance it a little, we extended the parser in EST with some operators from other formalisms.

Let $\mathcal{M}_1$, and $\mathcal{M}_2$ be LTSs with possibly different alphabets $\mathcal{A}_\tau$ and $\mathcal{A}'_\tau$, respectively. Let a be an action and $A \subseteq \mathcal{A}_\tau \cup \mathcal{A}'_\tau - \{\tau\}$. First, we introduce operators Synchronisation ($||$) and Interleaving ($|||$):

$$\textbf{Syn1}\ \frac{\mathcal{M}_1 \xrightarrow{a} \mathcal{M}'_1}{(\mathcal{M}_1||\mathcal{M}_2) \xrightarrow{a} (\mathcal{M}'_1||\mathcal{M}_2)}\ (a{=}\tau \vee \bar{a} \notin \mathcal{A}'_\tau)$$

$$\textbf{Syn2}\ \frac{\mathcal{M}_2 \xrightarrow{a} \mathcal{M}'_2}{(\mathcal{M}_1||\mathcal{M}_2) \xrightarrow{a} (\mathcal{M}_1||\mathcal{M}'_2)}\ (a{=}\tau \vee \bar{a} \notin \mathcal{A}_\tau)$$

$$\textbf{Syn3}\ \frac{\mathcal{M}_1 \xrightarrow{a} \mathcal{M}'_1 \quad \mathcal{M}_2 \xrightarrow{\bar{a}} \mathcal{M}'_2}{(\mathcal{M}_1||\mathcal{M}_2) \xrightarrow{\tau} (\mathcal{M}'_1||\mathcal{M}'_2)}\ (a \neq \tau)$$

$$\textbf{Int1}\ \frac{\mathcal{M}_1 \xrightarrow{a} \mathcal{M}'_1}{(\mathcal{M}_1|||\mathcal{M}_2) \xrightarrow{a} (\mathcal{M}'_1|||\mathcal{M}_2)}$$

$$\textbf{Int2}\ \frac{\mathcal{M}_2 \xrightarrow{a} \mathcal{M}'_2}{(\mathcal{M}_1|||\mathcal{M}_2) \xrightarrow{a} (\mathcal{M}_1|||\mathcal{M}'_2)}$$

Next, operator Partial synchronisation ($|[\ ]|$) is defined with the following rules:

$$\textbf{Part1}\ \frac{\mathcal{M}_1 \xrightarrow{a} \mathcal{M}'_1}{(\mathcal{M}_1|[A]|\mathcal{M}_2) \xrightarrow{a} (\mathcal{M}'_1|[A]|\mathcal{M}_2)}\ (a \notin A)$$

$$\textbf{Part2}\ \frac{\mathcal{M}_2 \xrightarrow{a} \mathcal{M}'_2}{(\mathcal{M}_1|[A]|\mathcal{M}_2) \xrightarrow{a} (\mathcal{M}_1|[A]|\mathcal{M}'_2)}\ (a \notin A)$$

$$\textbf{Part3}\ \frac{\mathcal{M}_1 \xrightarrow{a} \mathcal{M}'_1 \quad \mathcal{M}_2 \xrightarrow{\bar{a}} \mathcal{M}'_2}{(\mathcal{M}_1|[A]|\mathcal{M}_2) \xrightarrow{\tau} (\mathcal{M}'_1|[A]|\mathcal{M}'_2)}\ (a \in A)$$

Operator Partial synchronisation is a generalisation of operators Synchronisation and Interleaving. They can be both derived from it:

$$\mathcal{M}_1||\mathcal{M}_2 = \mathcal{M}_1|[\mathcal{A}_\tau \cap \mathcal{A}'_\tau - \{\tau\}]|\mathcal{M}_2$$

$$\mathcal{M}_1|||\mathcal{M}_2 = \mathcal{M}_1|[\emptyset]|\mathcal{M}_2$$

Operator Synchronisation can also be derived from operators Composition and Restriction: If $\mathcal{A}_\tau \cap \mathcal{A}'_\tau - \{\tau\} = \{a_1, a_2, \ldots, a_n\}$, then:

$$\mathcal{M}_1||\mathcal{M}_2 = (\mathcal{M}_1|\mathcal{M}_2)\backslash\hat{a}_1\backslash\hat{a}_2 \ldots \backslash\hat{a}_n$$

Operators Synchronisation, Interleaving, and Partial synchronisation are all commutative. Operator Interleaving is also associative. Operators Synchronisation and

Partial synchronisation are not associative and are in EST resolved from left to right. Similar operators are present in CSP [3] and LOTOS [1]. A good comparison of CCS and CSP operators is in [6].

Furthermore, we define a useful abbreviation supported by the EST parser. Let $\mathcal{M}$ be an LTS and $a_{i,1}, a_{i,2}, \ldots, a_{i,n_i} \in \mathcal{A}_\tau$ for all $i \in [1, N]$. Moreoever, let $\underline{A_i}$ stand for $a_{i,1}.a_{i,2}. \ \ldots \ .a_{i,n_i}$ and let $\underline{A'_i}$ stand for $a_{i,2}. \ \ldots \ .a_{i,n_i}$. Then, the following abbreviation is allowed in the specification:

$$(\underline{A_1}+\underline{A_2}+\ldots+\underline{A_N}).\mathcal{M} \triangleq \underline{A_1}.\mathcal{M}+\underline{A_2}.\mathcal{M}+\ldots+\underline{A_N}.\mathcal{M}$$

The following rule can be stated for it:

$$\textbf{PrefixN} \ \frac{}{(\underline{A_1}+\ldots+\underline{A_i}+\ldots+\underline{A_N}).\mathcal{M} \overset{a_{i,1}}{\rightarrow} \underline{A'_i}.\mathcal{M}}$$

As an example let us consider the Gas Station problem [2] given in EST with the following specification:

```
OPERATOR =
  (?prepay1+?prepay2).OP_PREPAID +
  (?charge1+?charge2).OP_CHARGED
OP_PREPAID =
  (?avlbl.!act+!occupied).OPERATOR
OP_CHARGED =
  (!change1+!change2).
  (?wait.!act+!none).OPERATOR
QUEUE = !avlbl.QUEUE_ACTIVE
QUEUE_ACTIVE =
  ?none.QUEUE +
  ?occupied.!wait.QUEUE_ACTIVE
PUMP =
  ?act.
  (?start1+?start2).
  (?finish1+?finish2).
  (!charge1+!charge2).PUMP
CUST =
  !prepay.!start.!stop.?change.CUST
STATION =
  OPERATOR
  |[avlbl,occupied,none,wait]|
  QUEUE
  |[activate,charge1,charge2]|
  PUMP
CUSTOMERS =
  CUST [prepay1/prepay][start1/start]
       [stop1/stop][change1/change]
  |||
  CUST [prepay2/prepay][start2/start]
       [stop2/stop][change2/change]
SYSTEM =
  STATION
  |[start1,start2,finish1,finish2]|
  CUSTOMERS
```

The system consists of an operator, a pump, and customers. A queue is added to the system for holding customers requests. The operator initially accepts money prepaid by customers and then activates the pump if it is available. On receiving the charge information from the pump, the operator gives the change to the customer. An LTS representing the operator is given in Figure 3.

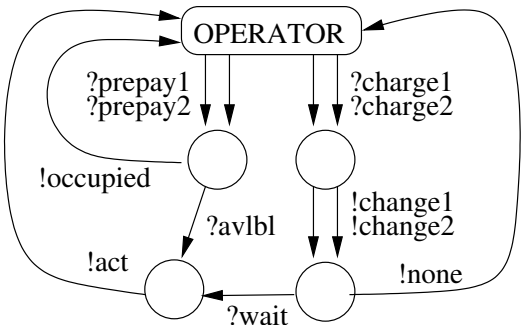

Figure 3: A detail from the Gas Station problem

Due to the lack of space we cannot discuss the Gas Station problem in detail. Let us only mention that the system obtained from the given specification is not correct because customers can receive the wrong change.

4 Conclusion

This paper presents the operators supported by parser in EST. Beside CCS-like operators, the parser includes additional operators similar to those from CSP and LOTOS. However, one should be careful about the meaning of these operators. Namely, in CSP and LOTOS two processes synchronise with each other by executing the same action. On the other hand, our definitions follow the CCS style and thus two processes synchronise with each other by simultaneous execution of an action and its coaction. The result of synchronisation is unobservable action τ.

References

[1] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, January 1987.

[2] S.C. Cheung and J. Kramer. Checking Subsystem Safety Properties in Compositional Reachability Analysis. In *The Proceedings of ICSE*, pages 144–154, 1996.

[3] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[4] Robert Meolic. EST Home Page. `http://lms.uni-mb.si/EST/`.

[5] R. Milner. *Communication and Concurrency*. Prentice-Hall International Series in Computer Science, 1989.

[6] R.J. van Glabbeek. Notes on the Methodology of CCS and CSP. *Theoretical Computer Science*, 177:329–349, 1997.