

# On the Meaning of Transition System Specifications

Rob van Glabbeek

Data61, CSIRO, Sydney, Australia

School of Computer Science and Engineering, University of New South Wales, Sydney, Australia

rvg@cs.stanford.edu

Transition System Specifications provide programming and specification languages with a semantics. They provide the meaning of a closed term as a *process graph*: a state in a labelled transition system. At the same time they provide the meaning of an  $n$ -ary operator, or more generally an open term with  $n$  free variables, as an  $n$ -ary operation on process graphs. The classical way of doing this, the *closed-term semantics*, reduces the meaning of an open term to the meaning of its closed instantiations. It makes the meaning of an operator dependent on the context in which it is employed. Here I propose an alternative *process graph semantics* of TSSs that does not suffer from this drawback.

Semantic equivalences on process graphs can be lifted to open terms conform either the closed-term or the process graph semantics. For pure TSSs the latter is more discriminating.

I consider five sanity requirements on the semantics of programming and specification languages equipped with a recursion construct: *compositionality*, applied to  $n$ -ary operators, recursion and variables, *invariance under  $\alpha$ -conversion*, and the *recursive definition principle*, saying that the meaning of a recursive call should be a solution of the corresponding recursion equations. I establish that the satisfaction of four of these requirements under the closed-term semantics of a TSS implies their satisfaction under the process graph semantics.

## 1 Introduction

Transition System Specifications (TSSs) [16] are a formalisation of *Structural Operational Semantics* [22] providing programming and specification languages with an interpretation. They provide the meaning of a closed term as a *process graph*: a state in a labelled transition system. At the same time they provide the meaning of an  $n$ -ary operator of the language, or more generally an open term with  $n$  free variables, as an  $n$ -ary operation on process graphs. The classical way of doing this proceeds by reducing the meaning of an open term to the meaning of its closed instantiations. I call this the *closed-term semantics* of TSSs. A serious shortcoming of this approach is that it makes the meaning of an operator dependent on the context in which it is employed.

**Example 1** Consider a TSS featuring unary operators  $f$ ,  $id$  and  $a.$  for each action  $a$  drawn from an alphabet  $A$ , and a constant  $0$ . The set of admitted transition labels is  $Act := A \uplus \{\tau\}$ . The transition rules are

$$a.x \xrightarrow{a} x \text{ (for all } a \in A) \quad \frac{x \xrightarrow{a} x'}{f(x) \xrightarrow{a} f(x')} \text{ (for all } a \in A) \quad \frac{x \xrightarrow{\alpha} x'}{id(x) \xrightarrow{\alpha} id(x')} \text{ (for all } \alpha \in Act)$$

When considered in their own right, the operators  $f$  and  $id$  are rather different: the latter can mimic  $\tau$ -transitions of its argument, and the former can not. Yet, in the context of the given TSS, one has  $f(p) \xleftrightarrow{\tau} id(p)$ , no matter which term  $p$  is substituted for the argument of these operators. Here  $\xleftrightarrow{\tau}$  denotes strong bisimulation equivalence, as defined in [21, 12]. This is because no process in the given TSS ever generates a transition with the label  $\tau$ . The identification of  $f$  and  $id$  up to  $\xleftrightarrow{\tau}$  can be considered

unfortunate, one reason being that it ceases to hold as soon as the language is enriched with a fresh operator  $\tau.$  with the transition rule  $\tau.x \xrightarrow{\tau} x$ . As I will show later, this invalidates an intuitively plausible theorem on the relative expressiveness of specification languages.

Here I propose an alternative *process graph semantics* of TSSs that does not suffer from this drawback.

In [10] I proposed five requirements on the semantics of programming and specification languages equipped with a recursion construct: *compositionality*, applied to  $n$ -ary operators, recursion and variables, *invariance under  $\alpha$ -conversion*, and the *recursive definition principle*, saying that the meaning of a recursive call should be a solution of the corresponding recursion equations.

In many prior works on structural operational semantics, (some of) these requirements have been shown to hold for various TSSs when employing the closed-term semantics. It would be time consuming to redo all that work for the process graph semantics proposed here. To prevent this I show that the satisfaction of four of these requirements under the closed-term semantics of a TSS implies their satisfaction under the process graph semantics. The remaining requirement holds almost always.

**Overview of the paper** Section 2 presents the syntax of the programming and specification languages I consider here. For simplicity I restrict myself to languages with single-sorted signature, optionally featuring a recursion construct. This is a rich enough setting to include process algebras like CCS [21], CSP [5], ACP [3], MEIJE [1, 24] and SCCS [20].

The traditional “*closed-term*” interpretation of the process calculi CCS, MEIJE and SCCS effectively collapses syntax and semantics by interpreting the entire language as one big labelled transition system (LTS) in which the closed terms of the language constitute the set of states. This LTS is generated by a TSS, as formally defined in Section 5. Semantic equivalences on LTSs thereby directly relate closed terms. Two open terms are judged equivalent iff each of their closed substitutions are.

In Section 3 I present an interpretation of programming and specification languages that is more common in universal algebra and mathematical logic [19], and is also used in the traditional semantics of ACP and CSP. It separates syntax and semantics through a semantic mapping that associates with each closed term a value, and with each open term an operation on values. This matches with what often is called *denotational semantics*, except that I do not require the meaning of recursion constructs to be provided by means of fixed point techniques. In Section 7 I specialise this general approach to an operational one by taking the values to be *process graphs*: states in labelled transition systems. Likewise, Section 6 casts the closed-term interpretation as a special case of the approach from Section 3, by taking the values to be the closed terms.

Section 3 also formulates the five sanity requirements mentioned above, most in a couple of equivalent forms. Section 6 shows how these requirements simplify to better recognisable forms under the closed-term interpretation of programming and specification languages. These requirements are parametrised by the choice of a semantic equivalence  $\sim$  on values, relating values that one does not need to distinguish. The traditional treatments of universal algebra and mathematical logic, and the process algebra CSP, do not involve such a semantic equivalence; this corresponds to letting  $\sim$  be the identity relation. In Section 4 I observe that any choice of  $\sim$  can be reduced to the identity relation, namely by taking as values  $\sim$ -equivalence classes of values. This reduction preserves the five sanity requirements.

After these preparations, Section 8 defines the promised process graph interpretation of TSSs. Some TSSs do not have a process graph interpretation, but I show that the large class of *pure* TSSs do.

Semantic equivalences on process graphs can be lifted to open terms conform either the closed-term or the process graph interpretation. Section 9 shows, under some mild conditions, that for pure TSSs the latter is more discriminating. Section 10 illustrates on a practical process algebra that whether a semantic equivalence is a congruence may depend on which of the two interpretations is chosen.

Section 11 proves the promised result that when four of the five sanity requirements have been established for the closed-term interpretation of a TSS, they also hold for its process graph interpretation. It also shows that the remaining requirement almost always holds

Section 12 argues that something is gained by moving from the closed-term interpretation of TSSs to the process-graph interpretation. Based on Example 1 it formulates an intuitively plausible theorem relating relative expressiveness of specification languages and conservative extensions, and shows how this theorem fails under the closed-term interpretation, but holds under the process graph interpretation.

Section 13 addresses related work, and Section 14 evaluates the five sanity requirements for languages specified by TSSs of a specific form.

## 2 Syntax

In this paper  $Var$  is an infinite set of *variables*, ranged over by  $X, Y, x, y, x_i$  etc.

**Definition 1** (*Terms*). A *function declaration* is a pair  $(f, n)$  of a *function symbol*  $f \notin Var$  and an *arity*  $n \in \mathbb{N}$ .<sup>1</sup> A function declaration  $(c, 0)$  is also called a *constant declaration*. A *signature* is a set of function declarations. The set  $\mathbb{T}^r(\Sigma)$  of *terms with recursion* over a signature  $\Sigma$  is defined inductively by:

- $Var \subseteq \mathbb{T}^r(\Sigma)$ ,
- if  $(f, n) \in \Sigma$  and  $t_1, \dots, t_n \in \mathbb{T}^r(\Sigma)$  then  $f(t_1, \dots, t_n) \in \mathbb{T}^r(\Sigma)$ ,
- If  $V_S \subseteq Var$ ,  $S : V_S \rightarrow \mathbb{T}^r(\Sigma)$  and  $X \in V_S$ , then  $\langle X|S \rangle \in \mathbb{T}^r(\Sigma)$ .

A term  $c()$  is abbreviated as  $c$ . A function  $S$  as appears in the last clause is called a *recursive specification*. It is often displayed as  $\{X = S_X \mid X \in V_S\}$ . Each term  $S_Y$  for  $Y \in V_S$  counts as a subterm of  $\langle X|S \rangle$ . An occurrence of a variable  $y$  in a term  $t$  is *free* if it does not occur in a subterm of the form  $\langle X|S \rangle$  with  $y \in V_S$ . For  $t \in \mathbb{T}^r(\Sigma)$  a term,  $var(t)$  denotes the set of variables occurring free in  $t$ . A term is *closed* if it contains no free occurrences of variables. For  $W \subseteq Var$ , let  $\mathbb{T}^r(\Sigma, W)$  denote the set of terms  $t$  with  $var(t) \subseteq W$ , and let  $\mathbb{T}^r(\Sigma) = \mathbb{T}^r(\Sigma, \emptyset)$  be the set of closed terms over  $\Sigma$ . The sets  $\mathbb{T}(\Sigma)$ ,  $\mathbb{T}(\Sigma, W)$  and  $\mathbb{T}(\Sigma)$  of open and closed terms over  $\Sigma$  without recursion are defined likewise, but without the last clause.

**Definition 2** (*Substitution*). A  $\Sigma$ -*substitution*  $\sigma$  is a partial function from  $Var$  to  $\mathbb{T}^r(\Sigma)$ . It is *closed* if it is a total function from  $Var$  to  $\mathbb{T}^r(\Sigma)$ . If  $\sigma$  is a substitution and  $t$  a term, then  $t[\sigma]$  denotes the term obtained from  $t$  by replacing, for  $x$  in the domain of  $\sigma$ , every free occurrence of  $x$  in  $t$  by  $\sigma(x)$ , while renaming bound variables if necessary to prevent name-clashes. In that case  $t[\sigma]$  is called a *substitution instance* of  $t$ . A substitution instance  $t[\sigma]$  where  $\sigma$  is given by  $\sigma(x_i) = u_i$  for  $i \in I$  is denoted as  $t[u_i/x_i]_{i \in I}$ , and for  $S$  a recursive specification  $\langle t|S \rangle$  abbreviates  $t[\langle Y|S \rangle/Y]_{Y \in V_S}$ .

Sometimes the syntax of a language is given as a signature together with an annotation that places some restrictions on the use of recursion [10]. This annotation may for instance require the sets  $V_S$  to be finite, the functions  $S$  computable, or the sets of equations  $S$  to be *guarded*: a syntactic criterion that ensures that they have unique solutions under a given interpretation. It may also rule out recursion altogether.

## 3 Semantics

A *language* can be given by an annotated signature, specifying its syntax, and an *interpretation*, assigning to every term  $t$  its meaning  $\llbracket t \rrbracket$ . The meaning of a closed term is a *value* chosen from a class of values  $\mathbb{D}$ ,

<sup>1</sup>This work generalises seamlessly to operators with infinitely many arguments. Such operators occur, for instance, in [4, Appendix A.2]. Hence one may take  $n$  to be any ordinal. It also generalises to operators, like the *summation* or *choice* of CCS [21], that take any *set* of arguments.

which is called a *domain*. The meaning of an open term is a *Var-ary operation* on  $\mathbb{D}$ : a function of type  $\mathbb{D}^{Var} \rightarrow \mathbb{D}$ , where  $\mathbb{D}^{Var}$  is the class of functions from  $Var$  to  $\mathbb{D}$ . It associates a value  $\llbracket t \rrbracket(\rho) \in \mathbb{D}$  to  $t$  that depends on the choice of a *valuation*  $\rho : Var \rightarrow \mathbb{D}$ . The valuation assigns a value from  $\mathbb{D}$  to each variable.

A *partial valuation* is a function  $\xi : W \rightarrow \mathbb{D}$  for  $W \subseteq Var$  that assigns a value only to certain variables. A *W-ary operation*, for  $W \subseteq Var$ , is a function  $F : \mathbb{D}^W \rightarrow \mathbb{D}$ . It associates a value to every  $W$ -tuple of values, i.e. to every partial valuation of the variables with domain  $W$ . If  $F : \mathbb{D}^{Var} \rightarrow \mathbb{D}$  and  $\zeta \in \mathbb{D}^{Var \setminus W}$  then  $F(\zeta) : \mathbb{D}^W \rightarrow \mathbb{D}$  is given by  $F(\zeta)(\xi) = F(\xi \cup \zeta)$  for any  $\xi \in \mathbb{D}^W$ . For  $\rho$  a valuation and  $W$  a set of variables,  $\rho \setminus W$  is the partial valuation with domain  $Var \setminus W$  such that  $(\rho \setminus W)(x) = \rho(x)$  for  $x \in Var \setminus W$ .

### 3.1 Sanity requirements on interpretations

Usually interpretations are required to satisfy some sanity requirements. The work [10] proposed five such requirements: *compositionality*, applied to variables,  $n$ -ary operators and recursion, *invariance under  $\alpha$ -conversion*, and the *recursive definition principle* (RDP).

In this paper I work with domains of interpretation  $\mathbb{D}$  that are equipped with a semantic equivalence relation  $\sim \subseteq \mathbb{D} \times \mathbb{D}$ . It indicates that values  $v, w \in \mathbb{D}$  with  $v \sim w$  need not be distinguished on our chosen level of abstraction. The equivalence  $\sim$  extends to functions  $F, G : \mathbb{D}^W \rightarrow \mathbb{D}$  by  $F \sim G$  iff  $F(\xi) \sim G(\xi)$  for all  $\xi \in \mathbb{D}^W$ . It extends to partial valuations  $\rho, \nu : W \rightarrow \mathbb{D}$  or functions  $\rho, \nu$  of type  $(\mathbb{D}^W \rightarrow \mathbb{D})^W$  by  $\rho \sim \nu$  iff  $\rho(X) \sim \nu(X)$  for all  $X \in W$ . Such an equivalence relation relaxes the requirements invariance under  $\alpha$ -conversion and RDP, and modifies compositionality; I speak of *compositionality up to  $\sim$* . The default case in which no semantic equivalence is in force corresponds to taking  $\sim$  to be the identity relation.

*Compositionality up to  $\sim$*  demands that the meaning of a variable is given by the chosen valuation, i.e.,

$$\llbracket x \rrbracket(\rho) \sim \rho(x) \quad (1)$$

for each  $x \in Var$  and valuation  $\rho : Var \rightarrow \mathbb{D}$ , and that the meaning of a term is completely determined by the meaning of its direct subterms. This means that for operators  $(f, n) \in \Sigma$  and valuations  $\rho, \nu : Var \rightarrow \mathbb{D}$

$$\llbracket t_i \rrbracket(\rho) \sim \llbracket u_i \rrbracket(\nu) \text{ (for all } i = 1, \dots, n) \Rightarrow \llbracket f(t_1, \dots, t_n) \rrbracket(\rho) \sim \llbracket f(u_1, \dots, u_n) \rrbracket(\nu) \quad (2)$$

and for recursive specifications  $S$  and  $S'$  with  $X \in V_S = V_{S'}$  and valuations  $\rho, \nu : Var \rightarrow \mathbb{D}$

$$\llbracket S_Y \rrbracket(\rho \setminus V_S) \sim \llbracket S'_Y \rrbracket(\nu \setminus V_S) \text{ (for all } Y \in V_S) \Rightarrow \llbracket \langle X | S \rangle \rrbracket(\rho) \sim \llbracket \langle X | S' \rangle \rrbracket(\nu). \quad (3)$$

Note that the precondition of (3) evaluates the variables in  $S_Y$  and  $S'_Y$  that are free in  $\langle X | S \rangle$  and  $\langle X | S' \rangle$  according to  $\rho$  and  $\nu$ , respectively, and the variables from  $V_S = V_{S'}$  under any common valuation  $\zeta$ .

*Invariance under  $\alpha$ -conversion* demands that the meaning of a term is independent of the names of its bound variables, i.e. for any injective substitution  $\gamma : V_S \rightarrow Var$  such that the range of  $\gamma$  contains no variables occurring free in  $\langle S_Y | S \rangle$  for some  $Y \in V_S$

$$\llbracket \langle \gamma(X) | S[\gamma] \rangle \rrbracket \sim \llbracket \langle X | S \rangle \rrbracket. \quad (4)$$

Finally, the meaning of a term  $\langle X | S \rangle$  should be the  $X$ -component of a solution of  $S$ . To be precise,

$$\llbracket \langle X | S \rangle \rrbracket \sim \llbracket \langle S_X | S \rangle \rrbracket. \quad (5)$$

This property is called the *recursive definition principle* [6].

A straightforward structural induction on  $t$  using (1), (2) and (3) shows that  $\llbracket t \rrbracket(\rho)$  depends only on the restriction of  $\rho$  to those variables that occur free in  $t$ . If there are no such variables,  $\llbracket t \rrbracket(\rho)$  does not depend on  $\rho$  at all, and consequently can be abbreviated to  $\llbracket t \rrbracket$ .

### 3.2 Alternative forms of the sanity requirements

Note that (2) holds iff for every  $(f, n) \in \Sigma$  there is a function  $f^{\mathbb{D}} : \mathbb{D}^n \rightarrow \mathbb{D}$  such that

$$\forall \rho \in \mathbb{D}^{Var} : \llbracket f(t_1, \dots, t_n) \rrbracket(\rho) \sim f^{\mathbb{D}}(\llbracket t_1 \rrbracket(\rho), \dots, \llbracket t_n \rrbracket(\rho)).$$

Requirement (3) can be characterised in the same vein:

**Proposition 1** Property (3) holds iff for every set  $W \subseteq Var$  there is a function  $\mu_W^{\mathbb{D}} : (\mathbb{D}^W \rightarrow \mathbb{D})^W \rightarrow \mathbb{D}^W$  such that for every recursive specification  $S : W \rightarrow \mathbb{T}^r(\Sigma)$  with  $X \in W$ , and every  $\rho : Var \rightarrow \mathbb{D}$ ,

$$\llbracket \langle X | S \rangle \rrbracket(\rho) \sim \mu_W^{\mathbb{D}}(\llbracket S \rrbracket(\rho \setminus W))(X).$$

**Proof:** Since the meaning of term  $S_Y \in \mathbb{T}^r(\Sigma)$  is of type  $\mathbb{D}^{Var} \rightarrow \mathbb{D}$ , the meaning of a recursive specification  $S : W \rightarrow \mathbb{T}^r(\Sigma)$  is of type  $\llbracket S \rrbracket : W \rightarrow (\mathbb{D}^{Var} \rightarrow \mathbb{D})$ . Applying to that a partial valuation  $\rho \setminus W : Var \setminus W \rightarrow \mathbb{D}$  yields a function  $\llbracket S \rrbracket(\rho \setminus W)$  of type  $(\mathbb{D}^W \rightarrow \mathbb{D})^W$ . Now for each  $\chi \in (\mathbb{D}^W \rightarrow \mathbb{D})^W$  choose, if possible, a pair  $S^\chi : W \rightarrow \mathbb{T}^r(\Sigma)$  and  $\rho^\chi : Var \rightarrow \mathbb{D}$  such that  $\llbracket S^\chi \rrbracket(\rho^\chi \setminus W) \sim \chi$ , and define  $\mu_W^{\mathbb{D}}(\chi)(X)$ , for  $X \in W$ , to be  $\llbracket \langle X | S^\chi \rangle \rrbracket(\rho^\chi)$ . For a  $\chi$  for which no such pair can be found the definition of  $\mu_W^{\mathbb{D}}(\chi)(X)$  is arbitrary. Now pick  $S : W \rightarrow \mathbb{T}^r(\Sigma)$  and  $\rho : Var \rightarrow \mathbb{D}$ . Let  $\chi := \llbracket S \rrbracket(\rho \setminus W)$ . Then  $\llbracket S \rrbracket(\rho \setminus W) = \chi \sim \llbracket S^\chi \rrbracket(\rho^\chi \setminus W)$ , so by (3) one has

$$\llbracket \langle X | S \rangle \rrbracket(\rho) \sim \llbracket \langle X | S^\chi \rangle \rrbracket(\rho^\chi) = \mu_W^{\mathbb{D}}(\chi)(X) = \mu_W^{\mathbb{D}}(\llbracket S \rrbracket(\rho \setminus W))(X).$$

The other direction, that the existence of such a  $\mu_W^{\mathbb{D}}$  implies (3), is trivial.  $\square$

Write  $t \stackrel{\alpha}{=} u$  if the terms  $t, u \in \mathbb{T}^r(\Sigma)$  differ only in the names of their bound variables. Then (4) can be rewritten as

$$t \stackrel{\alpha}{=} u \Rightarrow \llbracket t \rrbracket \sim \llbracket u \rrbracket. \quad (6)$$

**Proposition 2** In the presence of (2) and (3), (4) is equivalent to (6).

**Proof:** Clearly, (4) is a special case of (6). The other direction proceeds by structural induction on  $t$ .

In case  $t = X \in Var$  then  $u = X$  and thus  $\llbracket t \rrbracket \sim \llbracket u \rrbracket$ .

Let  $t = f(t_1, \dots, t_n)$ . Then  $u = f(u_1, \dots, u_n)$  and  $t_i \stackrel{\alpha}{=} u_i$  for each  $i = 1, \dots, n$ . By induction  $\llbracket t_i \rrbracket \sim \llbracket u_i \rrbracket$  for each  $i$ . This means that  $\llbracket t_i \rrbracket(\rho) \sim \llbracket u_i \rrbracket(\rho)$  for each  $\rho : Var \rightarrow \mathbb{D}$ . Hence, by (2),  $\llbracket t \rrbracket \sim \llbracket u \rrbracket$ .

Let  $t = \langle X | S \rangle$ . Then  $u = \langle \gamma(X) | S'[\gamma] \rangle$  for a recursive specification  $S' : V_S \rightarrow \mathbb{T}^r(\Sigma)$  with  $S_Y \stackrel{\alpha}{=} S'_Y$  for all  $Y \in V_S$ , and an injective substitution  $\gamma : V_S \rightarrow Var$  such that the range of  $\gamma$  contains no variables occurring free in  $\langle S'_Y | S' \rangle$  for some  $Y \in V_S$ . By induction  $\llbracket S_Y \rrbracket \sim \llbracket S'_Y \rrbracket$  for each  $Y$ . This means that  $\llbracket S_Y \rrbracket(\rho) \sim \llbracket S'_Y \rrbracket(\rho)$  for each  $\rho : Var \rightarrow \mathbb{D}$ , so in particular  $\llbracket S_Y \rrbracket(\rho \setminus V_S) \sim \llbracket S'_Y \rrbracket(\rho \setminus V_S)$  for each such  $\rho$ . Hence, by (3),  $\llbracket \langle X | S \rangle \rrbracket \sim \llbracket \langle \gamma(X) | S'[\gamma] \rangle \rrbracket$ . Thus, by (4),  $\llbracket t \rrbracket \sim \llbracket \langle X | S' \rangle \rrbracket \sim \llbracket \langle \gamma(X) | S'[\gamma] \rangle \rrbracket = \llbracket u \rrbracket$ .  $\square$

### 3.3 Applying semantic interpretations to substitutions

The semantic mapping  $\llbracket \cdot \rrbracket : \mathbb{T}^r(\Sigma) \rightarrow ((Var \rightarrow \mathbb{D}) \rightarrow \mathbb{D})$  extends to substitutions  $\sigma : Var \rightarrow \mathbb{T}^r(\Sigma)$  by  $\llbracket \sigma \rrbracket(\rho)(X) := \llbracket \sigma(X) \rrbracket(\rho)$  for all  $X \in Var$  and  $\rho : Var \rightarrow \mathbb{D}$ —here  $\sigma$  is extended to a total function by  $\sigma(Y) := Y$  for all  $Y \notin dom(\sigma)$ . Thus  $\llbracket \sigma \rrbracket$  is of type  $(Var \rightarrow \mathbb{D}) \rightarrow (Var \rightarrow \mathbb{D})$ , i.e. a map from valuations to valuations. The following result applies to languages satisfying sanity requirements (1)–(4).

**Proposition 3** Let  $t \in \mathbb{T}^r(\Sigma)$  be a term,  $\sigma : Var \rightarrow \mathbb{T}^r(\Sigma)$  a substitution, and  $\rho : Var \rightarrow \mathbb{D}$  a valuation. Then

$$\llbracket t[\sigma] \rrbracket(\rho) \sim \llbracket t \rrbracket(\llbracket \sigma \rrbracket(\rho)). \quad (7)$$

**Proof:** By the definition of substitution, there is an  $u \in \mathbb{T}^r(\Sigma)$  with  $t \stackrel{\alpha}{=} u$ , such that  $t[\sigma] = u[\sigma]$ , and when performing the substitution  $\sigma$  on  $u$  there is no need to rename any bound variables occurring in  $u$ . It now suffices to obtain  $\llbracket u[\sigma] \rrbracket(\rho) \sim \llbracket u \rrbracket(\llbracket \sigma \rrbracket(\rho))$ , because then  $\llbracket t[\sigma] \rrbracket(\rho) = \llbracket u[\sigma] \rrbracket(\rho) \sim \llbracket u \rrbracket(\llbracket \sigma \rrbracket(\rho)) \stackrel{(6)}{\sim} \llbracket t \rrbracket(\llbracket \sigma \rrbracket(\rho))$ . For this reason it suffices to establish (7) for terms  $t$  and substitutions  $\sigma$  with the property



(\*) that whenever a variable  $Z$  occurs free within a subterm  $\langle X|S \rangle$  of  $t$  with  $Y \in V_S$  then  $Y$  does not occur free in  $\sigma(Z)$ . I proceed with structural induction on  $t$ , while quantifying over all  $\rho$ .

Let  $t = X \in \text{Var}$ . Then  $\llbracket X \rrbracket(\llbracket \sigma \rrbracket(\rho)) \stackrel{(1)}{\sim} (\llbracket \sigma \rrbracket(\rho))(X) \stackrel{\text{def}}{=} \llbracket \sigma(X) \rrbracket(\rho) = \llbracket X[\sigma] \rrbracket(\rho)$ .

Let  $t = f(t_1, \dots, t_n)$ . By induction I may assume that  $\llbracket t_i[\sigma] \rrbracket(\rho) \sim \llbracket t_i \rrbracket(\llbracket \sigma \rrbracket(\rho))$  for  $i = 1, \dots, n$ . Hence

$$\llbracket f(t_1, \dots, t_n)[\sigma] \rrbracket(\rho) = \llbracket f(t_1[\sigma], \dots, t_n[\sigma]) \rrbracket(\rho) \stackrel{(2)}{\sim} \llbracket f(t_1, \dots, t_n) \rrbracket(\llbracket \sigma \rrbracket(\rho)).$$

Let  $t = \langle X|S \rangle$ . Then  $t[\sigma] = \langle X|S[\sigma \setminus V_S] \rangle$ . Given that the pair  $t, \sigma$  satisfies property (\*), so do the pairs  $S_Y, \sigma \setminus V_S$  for all  $Y \in V_S$ . Moreover, no  $Y \in V_S$  occurs free in  $\sigma(Z)$  for  $Z \notin V_S$  occurring free in  $S$ . (#)

By induction I assume  $\llbracket S_Y[\sigma \setminus V_S] \rrbracket(\rho) \sim \llbracket S_Y \rrbracket(\llbracket \sigma \setminus V_S \rrbracket(\rho))$  for all  $Y \in V_S$  and all  $\rho : \text{Var} \rightarrow \mathbb{D}$ . Any such  $\rho$  can be written as  $(\rho \setminus V_S) \cup \xi$  for some  $\xi : V_S \rightarrow \mathbb{D}$ . Now (#) yields that for all  $Z \in \text{Var}$  occurring free in  $S$

$$(\llbracket \sigma \setminus V_S \rrbracket((\rho \setminus V_S) \cup \xi))(Z) = (((\llbracket \sigma \rrbracket(\rho)) \setminus V_S) \cup \xi)(Z).$$

Hence  $\llbracket S_Y[\sigma \setminus V_S] \rrbracket(\rho \setminus V_S)(\xi) \sim \llbracket S_Y \rrbracket(\llbracket \sigma \setminus V_S \rrbracket((\rho \setminus V_S) \cup \xi)) = \llbracket S_Y \rrbracket((\llbracket \sigma \rrbracket(\rho)) \setminus V_S)(\xi)$  for all  $Y \in V_S$ ,  $\rho : \text{Var} \rightarrow \mathbb{D}$  and  $\xi : V_S \rightarrow \mathbb{D}$ . So  $\llbracket S_Y[\sigma \setminus V_S] \rrbracket(\rho \setminus V_S) \sim \llbracket S_Y \rrbracket((\llbracket \sigma \rrbracket(\rho)) \setminus V_S)$  for all  $Y \in V_S$  and  $\rho : \text{Var} \rightarrow \mathbb{D}$ .

One obtains  $\llbracket \langle X|S \rangle[\sigma] \rrbracket(\rho) = \llbracket \langle X|S[\sigma \setminus V_S] \rangle \rrbracket(\rho) \stackrel{(3)}{\sim} \llbracket \langle X|S \rangle \rrbracket(\llbracket \sigma \rrbracket(\rho))$ .  $\square$

## 4 Quotient Domains

An equivalence relation  $\sim$  on  $\mathbb{D}$  is a *congruence*<sup>2</sup> for  $\mathcal{L}$  if

$$\rho \sim \nu \Rightarrow \llbracket t \rrbracket(\rho) \sim \llbracket t \rrbracket(\nu) \tag{8}$$

for any term  $t$  and any valuations  $\rho, \nu : \text{Var} \rightarrow \mathbb{D}$ .

**Proposition 4** If a language  $\mathcal{L}$  is compositional up to an equivalence  $\sim$  then  $\sim$  is a congruence for  $\mathcal{L}$ .

**Proof:** A straightforward structural induction on  $t$ .  $\square$

Given a domain  $\mathbb{D}$  for interpreting languages and an equivalence relation  $\sim$ , the *quotient domain*  $\mathbb{D}/\sim$  consists of the  $\sim$ -equivalence classes of elements of  $\mathbb{D}$ . For  $v \in \mathbb{D}$  let  $[v]_\sim \in \mathbb{D}/\sim$  denote the equivalence class containing  $v \in \mathbb{D}$ . Likewise, for a valuation  $\rho : \text{Var} \rightarrow \mathbb{D}$  in  $\mathbb{D}$ , the valuation  $[\rho]_\sim : \text{Var} \rightarrow \mathbb{D}/\sim$  in  $\mathbb{D}/\sim$  is given by  $[\rho]_\sim(x) := [\rho(x)]_\sim$ ; it also represents the  $\sim$ -equivalence class of valuations in  $\mathbb{D}$  of which  $\rho$  is a member. Each valuation in  $\mathbb{D}/\sim$  is of the form  $[\rho]_\sim$ .

An interpretation  $\llbracket \_ \rrbracket : \mathbb{T}^r(\Sigma) \rightarrow (\mathbb{D}^{\text{Var}} \rightarrow \mathbb{D})$  that satisfies (8) is turned into the *quotient interpretation*  $\llbracket \_ \rrbracket_\sim : \mathbb{T}^r(\Sigma) \rightarrow ((\mathbb{D}/\sim)^{\text{Var}} \rightarrow \mathbb{D}/\sim)$  by defining  $\llbracket t \rrbracket_\sim([\rho]_\sim) := \llbracket t \rrbracket(\rho)$ . By (8), this is independent of the choice of a representative valuation  $\rho$  within the equivalence class  $[\rho]_\sim$ .

Let  $\llbracket \_ \rrbracket$  be an interpretation and  $\sim$  an equivalence such that (8) holds. Then  $\llbracket \_ \rrbracket$  satisfies the sanity requirements (1)–(5) of Section 3 up to  $\sim$  iff  $\llbracket \_ \rrbracket_\sim$  satisfies these requirements up to  $=$ .

<sup>2</sup>This property is called *lean congruence* in [13]. There  $\sim$  is called a *full congruence* for  $\mathcal{L}$  iff  $\mathcal{L}$  is compositional up to  $\sim$ . In the absence of recursion this is equivalent to (8), but in general it is a stronger requirement—i.e., the reverse of Proposition 4 does not hold [13].

## 5 Transition System Specifications

**Definition 3** (*Transition system specification*; GROOTE & VAANDRAGER [16]). Let  $\Sigma$  be an annotated signature and  $A$  a set (of actions). A (positive)  $(\Sigma, A)$ -literal is an expression  $t \xrightarrow{a} t'$  with  $t, t' \in \mathbb{T}^r(\Sigma)$  and  $a \in A$ . A transition rule over  $(\Sigma, A)$  is an expression of the form  $\frac{H}{\lambda}$  with  $H$  a set of  $(\Sigma, A)$ -literals (the premises of the rule) and  $\lambda$  a  $(\Sigma, A)$ -literal (the conclusion). A rule  $\frac{H}{\lambda}$  with  $H = \emptyset$  is also written  $\lambda$ . A transition system specification (TSS) is a triple  $(\Sigma, A, R)$  with  $R$  a set of transition rules over  $(\Sigma, A)$ .

The following definition (from [9]) tells when a literal is provable from a TSS. It generalises the standard definition (see e.g. [16]) by (also) allowing the derivation of transition rules. The derivation of a literal  $t \xrightarrow{a} t'$  corresponds to the derivation of the rule  $\frac{H}{t \xrightarrow{a} t'}$  with  $H = \emptyset$ . The case  $H \neq \emptyset$  corresponds to the derivation of  $t \xrightarrow{a} t'$  under the assumptions  $H$ .

**Definition 4** (*Proof*). Let  $P = (\Sigma, A, R)$  be a TSS. A proof of a transition rule  $\frac{H}{\lambda}$  from  $P$  is a well-founded, upwardly branching tree of which the nodes are labelled by  $(\Sigma, A)$ -literals, such that:

- the root is labelled by  $\lambda$ , and
- if  $\kappa$  is the label of a node  $q$  and  $K$  is the set of labels of the nodes directly above  $q$ , then
  - either  $K = \emptyset$  and  $\kappa \in H$ ,
  - or  $\frac{K}{\kappa}$  is a substitution instance of a rule from  $R$ .

If a proof of  $\frac{H}{\lambda}$  from  $P$  exists, then  $\frac{H}{\lambda}$  is provable from  $P$ , denoted  $P \vdash \frac{H}{\lambda}$ .

A labelled transition system (LTS) is a triple  $(S, A, \rightarrow)$  with  $S$  a set of states or processes,  $A$  a set of actions, and  $\rightarrow \subseteq S \times A \times S$  the transition relation, or set of transitions. A TSS  $P = (\Sigma, A, R)$  specifies the LTS  $(\mathbb{T}^r(\Sigma), A, \rightarrow)$  whose states are the closed terms over  $\Sigma$  and whose transitions are the closed literals provable from  $P$ .

For the sake of simplicity, the above treatment of TSSs deals with positive premises only. However, all results of this paper apply equally well, and with unaltered proofs, to TSS with negative premises  $t \xrightarrow{a} \neg$ , following the treatment below. The rest of the section may be skipped in first reading.

### 5.1 TSSs with negative premises

A negative  $(\Sigma, A)$ -literal is an expression  $t \xrightarrow{a} \neg$ . A transition rule may have positive and negative literals as premises, but must have a positive conclusion. Literals  $t \xrightarrow{a} u$  and  $t \xrightarrow{a} \neg$  are said to deny each other.

**Definition 5** [11] Let  $P = (\Sigma, A, R)$  be a TSS. A well-supported proof from  $P$  of a closed literal  $\lambda$  is a well-founded tree with the nodes labelled by closed literals, such that the root is labelled by  $\lambda$ , and if  $\kappa$  is the label of a node and  $K$  is the set of labels of the children of this node, then:

1. either  $\kappa$  is positive and  $\frac{K}{\kappa}$  is a closed substitution instance of a rule in  $R$ ;
2. or  $\kappa$  is negative and for each set  $N$  of closed negative literals with  $\frac{N}{\nu}$  provable from  $P$  and  $\nu$  a closed positive literal denying  $\kappa$ , a literal in  $K$  denies one in  $N$ .

$P \vdash_{ws} \lambda$  denotes that a well-supported proof from  $P$  of  $\lambda$  exists. A standard TSS  $P$  is complete if for each  $p$  and  $a$ , either  $P \vdash_{ws} p \xrightarrow{a} \neg$  or there exists a closed term  $q$  such that  $P \vdash_{ws} p \xrightarrow{a} q$ .

In [11] it is shown that no TSS admit well-supported proofs of literals that deny each other. Only a complete TSSs specifies an LTS; its transitions are the closed positive literals with a well-supported proof.

## 6 The Closed-term Semantics of Transition System Specifications

The default semantics of a language given as a TSS  $(\Sigma, A, R)$  is to take the domain  $\mathbb{D}$  in which the expressions are interpreted to be  $\mathbb{T}^r(\Sigma)$ , the set of closed terms over  $\Sigma$ . The meaning of a closed expression  $p \in \mathbb{T}^r(\Sigma)$  is simply itself:  $\llbracket p \rrbracket := p$ . The meaning  $\llbracket t \rrbracket \in \mathbb{D}^{Var} \rightarrow \mathbb{D}$  of an open expression  $t \in \mathbb{T}^r(\Sigma)$  is given by  $\llbracket t \rrbracket(\rho) := t[\rho]$ . Here one uses the fact that a valuation  $\rho : Var \rightarrow \mathbb{D}$  is also a closed substitution  $\rho : Var \rightarrow \mathbb{T}^r(\Sigma)$ . Given a semantic equivalence relation  $\sim \subseteq \mathbb{T}^r(\Sigma) \times \mathbb{T}^r(\Sigma)$ , the closed-term semantics of a TSS always satisfies Requirement (1), whereas (2)–(5) simplify to

$$p_i \sim q_i \text{ (for all } i = 1, \dots, n) \Rightarrow f(p_1, \dots, p_n) \sim f(q_1, \dots, q_n) \quad \text{and} \quad (2')$$

$$S_Y[\sigma] \sim S'_Y[\sigma] \text{ (for all } Y \in W \text{ and } \sigma : W \rightarrow \mathbb{T}^r(\Sigma)) \Rightarrow \langle X|S \rangle \sim \langle X|S' \rangle \quad (3')$$

$$\langle \gamma(X)|S[\gamma] \rangle \sim \langle X|S \rangle \quad (4')$$

$$\langle X|S \rangle \sim \langle S_X|S \rangle \quad (5')$$

for all functions  $(f, n) \in \Sigma$ , closed terms  $p_i, q_i \in \mathbb{T}^r(\Sigma)$ , recursive specifications  $S, S' : W \rightarrow \mathbb{T}^r(\Sigma, W)$  with  $X \in W \subseteq Var$ , and  $\gamma : W \rightarrow Var$  injective.

## 7 Process Graphs

When the expressions in a language are meant to represent processes, they are called *process expressions*, and the language a *process description language*. Suitable domains for interpreting process description languages are the class of *process graphs* [3] and its quotients. In such *graph domains* a process is represented by either a process graph, or an equivalence class of process graphs. Process graphs are also known as *state-transition diagrams* or *automata*. They are LTSs equipped with an initial state. A process graph can also be seen as a state in an LTS.

**Definition 6** A *process graph*, labelled over a set  $A$  of actions, is a triple  $G = (S, A, \rightarrow, I)$  with

- $S$  a set of *nodes* or *states*,
- $\rightarrow \subseteq S \times A \times S$  a set of *edges* or *transitions*,
- and  $I \in S$  the *root* or *initial* state.

Let  $\mathbb{G}(A)$  be the domain of process graphs labelled over  $A$ .

One writes  $r \xrightarrow{a} s$  for  $(r, a, s) \in \rightarrow$ . Virtually all so-called *interleaving models* for the representation of processes are isomorphic to graph models. For instance, the *failure sets* that represent expressions in the process description language CSP [5] can easily be encoded as equivalence classes of graphs, under a suitable equivalence. In [3] the language ACP is equipped with a process graph semantics, and the semantics of CCS, SCCS and MEIJE given in [21, 20, 1, 24] are operational ones, which, as I will show below, induce process graph semantics. In the languages  $\mathcal{L}$  studied in this paper, the domain  $\mathbb{D}$  in which  $\mathcal{L}$ -expressions are interpreted will be  $\mathbb{G}(A)$  for some set of actions  $A$ , or a subclass of  $\mathbb{G}(A)$ .

Usually the parts of a graph that cannot be reached from the initial state by following a finite path of transitions are considered meaningless for the description of processes. This means that one is only interested in process graphs as a model of system behaviour up to some equivalence, and this equivalence identifies at least graphs with the same reachable parts.

**Definition 7** The *reachable part* of a process graph  $(S, A, \rightarrow, I)$  is the process graph  $(S', A, \rightarrow', I)$  where

- $S' \subseteq S$  is the smallest set such that (1)  $I \in S'$  and (2) if  $r \in S'$  and  $r \xrightarrow{a} s$  then  $s \in S'$ ,
- and  $\rightarrow'$  is the restriction of  $\rightarrow$  to  $S' \times A \times S'$ .



## 8 A Process Graph Semantics of Transition System Specifications

This section proposes a process graph semantics of TSSs. For each TSS  $P = (\Sigma, A, R)$  it defines an interpretation  $\llbracket - \rrbracket_P : \mathbb{T}^r(\Sigma) \rightarrow (\mathbb{G}(A)^{\text{Var}} \rightarrow \mathbb{G}(A))$ , thus taking  $\mathbb{D}$  to be  $\mathbb{G}(A)$ .

**Definition 8** (*Interpreting the closed expressions in a TSS as process graphs*). Let  $P = (\Sigma, A, R)$  be a TSS and  $p \in \mathbb{T}^r(\Sigma)$ . Then  $\llbracket p \rrbracket_P^0 \in \mathbb{G}(A)$  is the reachable part of the process graph  $(\mathbb{T}^r(\Sigma), A, \rightarrow, p)$  with  $\rightarrow$  the set of transitions provable from  $P$ .

To define an interpretation  $\llbracket - \rrbracket_P : \mathbb{T}^r(\Sigma) \rightarrow (\mathbb{G}(A)^{\text{Var}} \rightarrow \mathbb{G}(A))$  of the open  $\Sigma$ -terms in  $\mathbb{G}(A)$  I would like to simply add to the signature  $\Sigma$  a constant  $G$  for each process graph  $G \in \mathbb{G}(A)$ . However,  $\mathbb{G}(A)$  is a proper class, whereas a signature needs to be a set. For this reason I work with appropriate subsets  $\mathbb{G}^*$  of  $\mathbb{G}(A)$  instead of with  $\mathbb{G}(A)$  itself. I will discuss the selection of  $\mathbb{G}^*$  later, but one requirement will be

$$\text{if } (S, A, \rightarrow, r) \in \mathbb{G}^* \text{ and } r \xrightarrow{a} s \text{ then also } (S, A, \rightarrow, s) \in \mathbb{G}^* \quad (\text{transition closure}).$$

Define the transition relation  $\rightarrow_{\mathbb{G}^*} \subseteq \mathbb{G}^* \times A \times \mathbb{G}^*$  by  $G \xrightarrow{a}_{\mathbb{G}^*} G'$  iff (i)  $G = (S, A, \rightarrow, r)$ , (ii) there is a transition  $(r, a, s) \in \rightarrow$ , and (iii)  $G' = (S, A, \rightarrow, s)$  is the same graph but with  $s$  as initial state.

Now consider a term  $t \in \mathbb{T}^r(\Sigma)$  and a valuation  $\rho : \text{Var} \rightarrow \mathbb{G}(A)$ . In order to define  $\llbracket t \rrbracket_P(\rho)$ , make sure that  $\mathbb{G}^*$  *supports*  $(t, \rho)$ , meaning that it contains  $\rho(x)$  for any variable  $x \in \text{Var}$  occurring free in  $t$ . Let  $P + \mathbb{G}^*$  be the TSS  $P$  to which all graphs  $G \in \mathbb{G}^*$  have been added as constants, and all transitions in  $\rightarrow_{\mathbb{G}^*}$  as transition rules without premises. As the valuation  $\rho$  now also is a substitution,  $t[\rho]$  is a closed term in the TSS  $P + \mathbb{G}^*$ . Define  $\llbracket t \rrbracket_P^{\mathbb{G}^*}(\rho)$  to be  $\llbracket t[\rho] \rrbracket_{P + \mathbb{G}^*}^0 \in \mathbb{G}(A)$ : the interpretation according to Definition 8 of the closed term  $t[\rho]$  from the TSS  $P + \mathbb{G}^*$ .

**Definition 9** (*The simple process graph semantics of TSSs*). A TSS  $P$  *manifestly induces a process graph semantics* iff, for any term  $t$  and valuation  $\rho$ , the interpretation  $\llbracket t \rrbracket_P^{\mathbb{G}^*}(\rho)$  is independent of the choice of  $\mathbb{G}^*$ , as long as  $\mathbb{G}^*$  is transition-closed and supports  $(t, \rho)$ . In that case  $\llbracket t \rrbracket_P(\rho)$  is defined to be  $\llbracket t \rrbracket_P^{\mathbb{G}^*}(\rho)$ .

It is possible to enlarge the class of TSSs that induce a process graph semantics a little bit:

**Definition 10** (*The process graph semantics of TSSs*). Call a choice of  $\mathbb{G}^*$  *adequate* for (the interpretation of)  $\llbracket t \rrbracket_P(\rho)$  if  $\mathbb{G}^*$  is transition-closed and supports  $(t, \rho)$ , and  $\llbracket t \rrbracket_P^{\mathbb{G}'}(\rho) = \llbracket t \rrbracket_P^{\mathbb{G}^*}(\rho)$  for any transition-closed superset  $\mathbb{G}'$  of  $\mathbb{G}^*$ . Now  $P$  *induces a process graph semantics* iff, for any term  $t$  and valuation  $\rho$ , an adequate choice  $\mathbb{G}^*$  for  $\llbracket t \rrbracket_P(\rho)$  exists. In that case  $\llbracket t \rrbracket_P(\rho)$  is defined to be  $\llbracket t \rrbracket_P^{\mathbb{G}^*}(\rho)$  for any adequate choice of  $\mathbb{G}^*$ .

**Example 1 (continued)** Reconsider the operators  $f$  and  $id$  from Example 1. To judge whether they are essentially different one compares the open terms  $f(x)$  and  $id(x)$ . Their meanings are values that depend on the choice of a valuation  $\rho$ , mapping variables to values. In fact they depend on the value  $\rho(x)$  only.

Under the closed term interpretation of the TSS  $P$  of Example 1,  $\rho(x)$  is a closed term in the language; it cannot have an outgoing  $\tau$ -transition. Thus  $\llbracket f(x) \rrbracket_P(\rho) \not\leftrightarrow \llbracket x \rrbracket_P(\rho) \not\leftrightarrow \llbracket id(x) \rrbracket_P(\rho)$  for any such  $\rho$ , so  $\llbracket f(x) \rrbracket_P \not\leftrightarrow \llbracket x \rrbracket_P \not\leftrightarrow \llbracket id(x) \rrbracket_P$ , i.e.,  $f$  and  $id$  are strongly bisimilar.

However, under the process graph interpretation,  $\rho(x)$  is a process graph, and one may take  $\rho(x)$  to be  $\rightarrow \xrightarrow{\tau} \circ \xrightarrow{c} \circ \rightarrow$ , where the short arrow indicates the initial state. With this valuation, the process graph  $\llbracket id(x) \rrbracket_P(\rho)$  is isomorphic to  $\rho(x)$ , whereas  $\llbracket f(x) \rrbracket_P$  has no outgoing transitions. So  $\llbracket f(x) \rrbracket_P(\rho) \not\leftrightarrow \llbracket x \rrbracket_P(\rho) \not\leftrightarrow \llbracket id(x) \rrbracket_P(\rho)$  and consequently  $\llbracket f(x) \rrbracket_P \not\leftrightarrow \llbracket x \rrbracket_P \not\leftrightarrow \llbracket id(x) \rrbracket_P$ , i.e.,  $f$  and  $id$  are not bisimilar.

The smallest set of process graphs  $\mathbb{G}^*$  that is adequate for the interpretation of  $\llbracket f(x) \rrbracket_P(\rho)$  and  $\llbracket id(x) \rrbracket_P(\rho)$  is  $\left\{ \begin{array}{c} \downarrow \\ \circ \end{array} \xrightarrow{\tau} \circ \xrightarrow{c} \circ \rightarrow, \circ \xrightarrow{\tau} \begin{array}{c} \downarrow \\ \circ \end{array} \xrightarrow{c} \circ \rightarrow, \circ \xrightarrow{\tau} \circ \xrightarrow{c} \begin{array}{c} \downarrow \\ \circ \end{array} \right\}$ .

**Example 2** A TSS with a rule  $c \xrightarrow{a} x$  (without premises) does not induce a process graph semantics. Namely, no matter which set  $\mathbb{G}^*$  one takes, there is always a larger set  $\mathbb{G}'$  and a process graph  $G \in \mathbb{G}' \setminus \mathbb{G}^*$ . Thus, the process graph  $\llbracket c \rrbracket_P^{\mathbb{G}'}$  has the transition  $c \xrightarrow{a} G$  but  $\llbracket c \rrbracket_P^{\mathbb{G}^*}$  does not. Hence  $\mathbb{G}^*$  is not adequate. Consequently, the TSS does not induce a process graph semantics.

**Example 3** Let  $P$  be the TSS with constants  $c$  and  $0$  and as only rule

$$\frac{x \xrightarrow{a} y \quad z \xrightarrow{b} y}{c \xrightarrow{a} 0}.$$

Then  $\llbracket c \rrbracket_P^{\mathbb{G}^*}(\rho)$  is independent of  $\rho$ , since  $c$  is a closed term. In any adequate choice of  $\mathbb{G}^*$  there is a graph in which an  $a$ -transition and a  $b$ -transition end in a common state, and using such a choice one finds that  $\llbracket c \rrbracket_P^{\mathbb{G}^*}$  has the transition  $c \xrightarrow{a} 0$ . However, when taking  $\mathbb{G}^*$  to be a set of trees,  $\llbracket c \rrbracket_P^{\mathbb{G}^*}$  has no transitions.

$P$  induces a process graph semantics according to Definition 10, but not according to Definition 9.

If we stick with Definition 9,  $\llbracket t \rrbracket_P = \llbracket t \rrbracket_P^{\emptyset}$  for closed terms  $t$ .

**Definition 11** The *rule-bound variables* of a transition rule  $\frac{H}{t \xrightarrow{a} t'}$  form the smallest set  $B$  such that

- $\text{var}(t) \subseteq B$ , and
- if  $u \xrightarrow{b} u'$  is a premise in  $H$  and  $\text{var}(u) \subseteq B$  then  $\text{var}(u') \subseteq B$ .

A TSS is called *pure* if all variables occurring free in one of its rules are rule-bound in that rule.

This concept of a pure TSS generalises the one from [16], and coincides with it for TSSs in the tyft/tyxt format studied in [16]. The TSSs of all common process algebras are pure. So is the TSS of Example 1, but the ones of Examples 2 and 3 are not.

**Proposition 5** Any pure TSS manifestly induces a process graph semantics.

**Proof:** For a given term  $t \in \mathbb{T}^r(\Sigma)$  and valuation  $\rho : \text{Var} \rightarrow \mathbb{G}(A)$ , let  $\mathbb{G}_0^*$  be the smallest set of process graphs that is transition-closed and supports  $(t, \rho)$ . Then for any  $\mathbb{G}^* \supseteq \mathbb{G}_0^*$  and any transition  $t[\rho] \xrightarrow{a} u$  provable from  $P + \mathbb{G}^*$ , any term  $v$  occurring in a proof of this transition, including the target  $u$ , has the form  $t'[\rho]$  with  $t' \in \mathbb{T}^r(\Sigma)$ , such that  $\rho(x) \in \mathbb{G}_0^*$  for any  $x \in \text{var}(t)$ . This follows by a fairly straightforward induction on the size of proofs, with a nested induction on the derivation of rule-boundedness. As a consequence, the process graph  $\llbracket t[\rho] \rrbracket_{P+\mathbb{G}^*}^{\emptyset}$  does not depend in any way on  $\mathbb{G}^* \setminus \mathbb{G}_0^*$ .  $\square$

**Remark** One may wonder whether the above treatment can be simplified by skipping, in Definition 8, “the reachable part of”. The answer is negative, for in that case  $\llbracket t \rrbracket_P^{\mathbb{G}^*}(\rho)$  would never be independent of the choice of  $\mathbb{G}^*$ , because all  $G \in \mathbb{G}^*$  would occur as (unreachable) states in the process graph  $\llbracket t \rrbracket_P^{\mathbb{G}^*}(\rho)$ .

**Summary** In this section, terms in a TSS are interpreted in the domain of process graphs as follows. Let  $P = (\Sigma, A, \mathcal{R})$  be a TSS and  $t \in \mathbb{T}^r(\Sigma)$  a term. The meaning  $\llbracket t \rrbracket_P : \mathbb{G}(A)^{\text{Var}} \rightarrow \mathbb{G}(A)$  of  $t$  is given by  $\llbracket t \rrbracket_P(\rho) := \llbracket t[\rho] \rrbracket_{P+\mathbb{G}^*}^{\emptyset}$ , with  $\mathbb{G}^*$  adequate for  $\llbracket t \rrbracket_P(\rho)$ . Here  $\mathbb{G}^*$  is transition-closed and supports  $(t, \rho)$ ; it is *adequate* if further increasing this set does not alter the definition of  $\llbracket t \rrbracket_P(\rho)$ . If no adequate  $\mathbb{G}^*$  exists,  $\llbracket t \rrbracket_P(\rho)$  remains undefined, and  $P$  does not induce a process graph semantics. If  $P$  is pure, any transition-closed set  $\mathbb{G}^* \subseteq \mathbb{G}(A)$  supporting  $(t, \rho)$  is adequate.

## 9 Lifting Semantic Equivalences to Open Terms

The following definition shows how any equivalence relation  $\sim$  defined on a domain  $\mathbb{D}$  in which a language  $\mathcal{L}$  is interpreted, lifts to the open terms of  $\mathcal{L}$ .

**Definition 12** (*Lifting equivalences to open terms*). For a language  $\mathcal{L}$ , given as an annotated signature  $\Sigma$  and an interpretation  $\llbracket \_ \rrbracket : \mathbb{T}^r(\Sigma) \rightarrow (\mathbb{D}^{\text{Var}} \rightarrow \mathbb{D})$ , and an equivalence relation  $\sim$  on  $\mathbb{D}$ , write  $t \sim_{\mathcal{L}} u$  for  $t, u \in \mathbb{T}^r(\Sigma)$  iff  $\llbracket t \rrbracket(\rho) \sim \llbracket u \rrbracket(\rho)$  for all valuations  $\rho : \text{Var} \rightarrow \mathbb{D}$ .

This definition can be applied to any language  $\mathcal{L}$  given by a TSS  $P = (\Sigma, A, R)$ . In this case  $\sim$  must be defined on  $\mathbb{G}(A)$ . Write  $\sim_P^{\text{pg}}$  for  $\sim_{\mathcal{L}}$  as defined above when taking as interpretation the process graph semantics  $\llbracket \_ \rrbracket : \mathbb{T}^r(\Sigma) \rightarrow (\mathbb{G}(A)^{\text{Var}} \rightarrow \mathbb{G}(A))$  of  $\mathcal{L}$ .

An equivalence  $\sim$  on  $\mathbb{G}(A)$  also lifts to the closed terms  $\mathbb{T}^r(\Sigma)$  of  $\mathcal{L}$ . Namely, let  $(\mathbb{T}^r(\Sigma), A, \rightarrow)$  be the LTS specified by  $P$  as defined in Section 5. Then  $(\mathbb{T}^r(\Sigma), A, \rightarrow, p) \in \mathbb{G}(A)$  is a process graph for any  $p \in \mathbb{T}^r(\Sigma)$ . Now write  $p \sim q$ , for  $p, q \in \mathbb{T}^r(\Sigma)$ , whenever  $(\mathbb{T}^r(\Sigma), A, \rightarrow, p) \sim (\mathbb{T}^r(\Sigma), A, \rightarrow, q)$ .

Using this, Definition 12 can also be instantiated by taking as interpretation the closed-term semantics  $\llbracket \_ \rrbracket : \mathbb{T}^r(\Sigma) \rightarrow (\mathbb{T}^r(\Sigma)^{\text{Var}} \rightarrow \mathbb{T}^r(\Sigma))$  of  $\mathcal{L}$ , as defined in Section 6. Write  $\sim_P^{\text{ci}}$  for  $\sim_{\mathcal{L}}$  defined thusly. So

$$t \sim_P^{\text{ci}} u \text{ iff } t[\sigma] \sim u[\sigma] \text{ for any closed substitution } \sigma,$$

i.e., two open terms are related by  $\sim_P^{\text{ci}}$  if all of their *closed instantiations* are related by  $\sim$ .

Having lifted semantic equivalences  $\sim$  from process graphs to open terms in two ways, one wonders how the resulting equivalences compare. Instantiating  $\sim$  with strong bisimilarity,  $\leftrightarrow$ , Example 1 shows that  $f(x) \leftrightarrow_P^{\text{ci}} id(x)$  yet  $f(x) \not\leftrightarrow_P^{\text{pg}} id(x)$ . For the other direction consider the TSS with constants  $0, c$  and  $d$ , alphabet  $\{a, b\}$ , and the rules

$$d \xrightarrow{a} 0 \quad \frac{x \xrightarrow{b} y}{c \xrightarrow{a} 0}.$$

Under the closed-term interpretation, no  $b$ -transition from any term can be derived, so  $0 \leftrightarrow_P^{\text{ci}} c \not\leftrightarrow_P^{\text{ci}} d$ . Yet under the process graph interpretation, since there exists some graph that can do a  $b$ -transition, one has  $c \xrightarrow{a} 0$ , and obtains  $0 \not\leftrightarrow_P^{\text{pg}} c \leftrightarrow_P^{\text{pg}} d$ . So in general  $\leftrightarrow_P^{\text{ci}}$  and  $\leftrightarrow_P^{\text{pg}}$  are incomparable.

The above TSS is not pure; the variable  $x$  is not rule-bound. For pure TSSs no such example exists.

**Theorem 1** Let  $P = (\Sigma, A, R)$  be a pure TSS and  $\approx$  an equivalence on  $\mathbb{G}(A)$  that relates each process graph with its reachable part. Moreover, let  $\sim \subseteq \approx$  be a possibly finer, or more discriminating, equivalence that satisfies requirements (1)–(4) of Section 3.1. Then  $t \approx_P^{\text{pg}} u$  implies  $t \approx_P^{\text{ci}} u$ .

**Proof:** Suppose  $t \approx_P^{\text{pg}} u$ , and let  $\sigma : \text{Var} \rightarrow \mathbb{T}^r(\Sigma)$  be a closed substitution. It suffices to establish that  $t[\sigma] \approx u[\sigma]$ . Let  $\llbracket \sigma \rrbracket_P : \text{Var} \rightarrow \mathbb{G}(A)$  be the valuation defined by  $\llbracket \sigma \rrbracket_P(X) := \llbracket \sigma(X) \rrbracket_P$ . This is the definition of  $\llbracket \sigma \rrbracket_P$  from Section 3.3, specialised to closed substitutions. Here  $\llbracket q \rrbracket_P$ , for  $q \in \mathbb{T}^r(\Sigma)$ , is the process graph semantics of  $q$  as defined in Section 8, so  $\llbracket q \rrbracket_P = \llbracket q \rrbracket_{P+\mathbb{G}^*}^{\emptyset}$  for an adequate choice of  $\mathbb{G}^*$ . Since  $P$  is pure and  $q$  closed, the empty set of process graphs is adequate by Proposition 5. By Definition 8,  $\llbracket q \rrbracket_P^{\emptyset}$  is the reachable part of the process graph  $(\mathbb{T}^r(\Sigma), A, \rightarrow, p)$ , so  $\llbracket q \rrbracket_P^{\emptyset} \approx (\mathbb{T}^r(\Sigma), A, \rightarrow, p)$ . Since  $t \approx_P^{\text{pg}} u$ , one has  $\llbracket t \rrbracket_P(\llbracket \sigma \rrbracket_P) \approx \llbracket u \rrbracket_P(\llbracket \sigma \rrbracket_P)$  by Definition 12. Moreover,  $\llbracket t \rrbracket_P(\llbracket \sigma \rrbracket_P) \sim \llbracket t[\sigma] \rrbracket_P$  by Proposition 3. Hence

$$\llbracket t[\sigma] \rrbracket_P^{\emptyset} = \llbracket t[\sigma] \rrbracket_P \approx \llbracket t \rrbracket_P(\llbracket \sigma \rrbracket_P) \approx \llbracket u \rrbracket_P(\llbracket \sigma \rrbracket_P) \approx \llbracket u[\sigma] \rrbracket_P = \llbracket u[\sigma] \rrbracket_P^{\emptyset}$$

and thus  $(\mathbb{T}^r(\Sigma), A, \rightarrow, t[\sigma]) \approx \llbracket t[\sigma] \rrbracket_{P+\mathbb{G}^*}^{\emptyset} \approx \llbracket u[\sigma] \rrbracket_P^{\emptyset} \approx (\mathbb{T}^r(\Sigma), A, \rightarrow, u[\sigma])$ , i.e.,  $t[\sigma] \approx u[\sigma]$ .  $\square$

So, under the conditions of Theorem 1,  $\approx_P^{\text{pg}}$  is a finer, or more discriminating, equivalence than  $\approx_P^{\text{ci}}$ .

## 10 Congruence Properties

Whether a semantic equivalence  $\sim$  is a congruence (cf. (8) in Section 4) may depend on whether the closed-term or the process graph semantics is chosen. The following example illustrates this for a practical process algebra.

**Example 4** Consider the TSS with constants 1 and  $\alpha$  for  $\alpha \in Act = A \uplus \{\tau\}$  and binary operators  $+$  and  $;$ , denoting choice and sequencing in a process algebra, with the following transition rules:

$$\alpha \xrightarrow{\alpha} 1 \quad \frac{x \xrightarrow{\alpha} x'}{x+y \xrightarrow{\alpha} x'} \quad \frac{y \xrightarrow{\alpha} y'}{x+y \xrightarrow{\alpha} y'} \quad \frac{x \xrightarrow{\alpha} x'}{x;y \xrightarrow{\alpha} x';y} \quad \frac{x \xrightarrow{\alpha} x' \quad y \xrightarrow{\beta} y'}{x;y \xrightarrow{\beta} y'} \text{ for all } \alpha \in Act$$

The process 1, like 0 in CCS, has no outgoing transitions, meaning that it performs no actions. The sequencing operator performs all actions its first argument can do, until its first argument can perform no further actions; then it continues with its second argument. I employ no recursion here.

As equivalence relation  $\sim$  I take weak bisimulation equivalence,  $\xrightarrow{w}$ , as defined in [21, 12]. For the term  $t$  from (8) take  $x;b$ . Let  $\rho$  and  $\nu$  be valuations with

$$\rho(x) = \begin{array}{c} \longrightarrow \circ \xrightarrow{a} \circ \end{array} \quad \text{and} \quad \nu(x) = \begin{array}{c} \longrightarrow \circ \xrightarrow{a} \circ \begin{array}{l} \curvearrowright \tau \\ \curvearrowleft \end{array} \end{array} .$$

Then  $\rho(x) \xrightarrow{w} \nu(x)$ , so that I may assume  $\rho \xrightarrow{w} \nu$ . Now the term  $t$  performs the sequential composition of the process filled in for  $x$  with the process doing a single  $b$  action. One has  $\llbracket x;b \rrbracket(\rho) \not\xrightarrow{w} \llbracket x;b \rrbracket(\nu)$  because only the first of these processes can ever perform the  $b$ . Thus, when using the process graph semantics of this TSS,  $\xrightarrow{w}$  fails to be a congruence for the language specified.

However, when taking the closed-term semantics, all processes that may be filled in for  $x$  are terms in the given language and thus must terminate after performing finitely many transitions. In this setting  $\xrightarrow{w}$  is actually a congruence. However, it stops being a congruence when recursion is added to the language.

## 11 Relating Sanity Requirements for the Two Semantics of TSSs

Given an equivalence relation  $\sim$  on  $\mathbb{G}(A)$ , let  $\sim_P^{\text{cl}}$  be the equivalence relation on the set  $\mathbb{T}^r(\Sigma)$  of closed terms of a TSS  $P = (\Sigma, A, R)$  defined by  $p \sim_P^{\text{cl}} q$  iff  $\llbracket p \rrbracket_P^0 \sim \llbracket q \rrbracket_P^0$  (cf. Definition 8). In case  $\sim$  relates each process graph with its reachable part, the equivalence  $\sim_P^{\text{cl}}$  coincides with  $\sim_P^{\text{ci}}$ , as defined in Section 9.

**Observation 1** If  $P$  is pure and  $p, q \in \mathbb{T}^r(\sigma)$ , then  $p \sim_P^{\text{cl}} q$  iff  $p \sim_P^{\text{pg}} q$ .

**Theorem 2** Let  $P$  be a TSS that induces a process graph semantics  $\llbracket - \rrbracket_P$  and let  $\sim$  be an equivalence relation on  $\mathbb{G}(A)$ . Then  $\llbracket - \rrbracket_P$  satisfies the sanity requirements (2)–(5) of Section 3 up to  $\sim$  if the closed-term semantics of  $P + \mathbb{G}^*$  satisfies these requirements up to  $\sim_{P+\mathbb{G}^*}^{\text{cl}}$  for any choice of  $\mathbb{G}^*$ .

**Proof:** Let  $\rho, \nu : Var \rightarrow \mathbb{G}(A)$ ,  $(f, n) \in \Sigma$  and  $t_i, u_i \in \mathbb{T}^r(\Sigma)$ , such that  $\llbracket t_i \rrbracket_P(\rho) \sim \llbracket u_i \rrbracket_P(\nu)$  for  $i = 1, \dots, n$ . Let the set  $\mathbb{G}^* \subseteq \mathbb{G}(A)$  be adequate for the definition of  $\llbracket t_i \rrbracket_P(\rho)$  and  $\llbracket u_i \rrbracket_P(\nu)$  for  $i = 1, \dots, n$  as well as for  $\llbracket f(t_1, \dots, t_n) \rrbracket_P(\rho)$  and  $\llbracket f(u_1, \dots, u_n) \rrbracket_P(\nu)$ . Then  $\llbracket t_i \rrbracket_P(\rho) \sim_{P+\mathbb{G}^*}^{\text{cl}} \llbracket u_i \rrbracket_P(\nu)$ , i.e.,  $t_i \sim_{P+\mathbb{G}^*}^{\text{cl}} u_i$ , for  $i = 1, \dots, n$ . So  $f(t_1[\rho], \dots, t_n[\rho]) \sim_{P+\mathbb{G}^*}^{\text{cl}} f(u_1[\nu], \dots, u_n[\nu])$  by Requirement (2') for the closed-term semantics of  $P + \mathbb{G}^*$ ; that is,  $\llbracket f(t_1[\rho], \dots, t_n[\rho]) \rrbracket_{P+\mathbb{G}^*}^0 \sim \llbracket f(u_1[\nu], \dots, u_n[\nu]) \rrbracket_{P+\mathbb{G}^*}^0$ , or  $\llbracket f(t_1, \dots, t_n) \rrbracket_P(\rho) \sim \llbracket f(u_1, \dots, u_n) \rrbracket_P(\nu)$ .

Let  $\rho, \nu : Var \rightarrow \mathbb{G}(A)$  and  $S, S' : W \rightarrow \mathbb{T}^r(\Sigma)$  with  $X \in W \subseteq Var$ , such that  $\llbracket S_Y \rrbracket_P(\rho \setminus W) \sim \llbracket S'_Y \rrbracket_P(\nu \setminus W)$  for  $Y \in W$ . The latter means that  $\llbracket S_Y \rrbracket_P(\rho \setminus W)(\xi) \sim \llbracket S'_Y \rrbracket_P(\nu \setminus W)(\xi)$  for all  $Y \in W$  and  $\xi : W \rightarrow \mathbb{G}(A)$ . Let  $\mathbb{G}^* \subseteq \mathbb{G}(A)$  be adequate for  $\llbracket S_Y \rrbracket_P(\xi \cup \rho \setminus W)$  and  $\llbracket S'_Y \rrbracket_P(\xi \cup \nu \setminus W)$  for all  $Y \in W$  and  $\xi : W \rightarrow \mathbb{G}(A)$ ,

as well as for  $\llbracket \langle X|S \rangle \rrbracket_P(\rho)$  and  $\llbracket \langle X|S' \rangle \rrbracket_P(\nu)$ . Then  $S_Y[\rho \setminus W][\xi] \sim_{P+\mathbb{G}^*}^{\text{cl}} S'_Y[\nu \setminus W][\xi]$  for all  $Y \in W$  and  $\xi : W \rightarrow \mathbb{G}(A)$ . So  $\langle X|S[\rho \setminus W] \rangle \sim_{P+\mathbb{G}^*}^{\text{cl}} \langle X|S'[\nu \setminus W] \rangle$  by Requirement (3') for the closed-term semantics of  $P+\mathbb{G}^*$ , and  $\llbracket \langle X|S \rangle \rrbracket_P(\rho) = \llbracket \langle X|S[\rho] \rangle \rrbracket_{P+\mathbb{G}^*}^0 = \llbracket \langle X|S[\rho \setminus W] \rangle \rrbracket_{P+\mathbb{G}^*}^0 \sim \llbracket \langle X|S'[\nu \setminus W] \rangle \rrbracket_{P+\mathbb{G}^*}^0 = \llbracket \langle X|S' \rangle \rrbracket_P(\nu)$ .

Let  $S : V_S \rightarrow \mathbb{T}^r(\Sigma)$  with  $X \in V_S \subseteq \text{Var}$ , and let  $\gamma : V_S \rightarrow \text{Var}$  be an injective substitution such that the range of  $\gamma$  contains no variables occurring free in  $\langle S_Y|S \rangle$  for some  $Y \in V_S$ . Take  $\rho : \text{Var} \rightarrow \mathbb{G}(A)$ . Let  $\mathbb{G}^*$  be adequate for  $\llbracket \langle X|S \rangle \rrbracket_P(\rho)$  and  $\llbracket \langle \gamma(X)|S[\gamma] \rangle \rrbracket_P(\rho)$ . By Requirement (4') for the closed-term semantics of  $P+\mathbb{G}^*$  one has  $\langle \gamma(X)|S[\gamma][\rho] \rangle \sim_{P+\mathbb{G}^*}^{\text{cl}} \langle X|S[\rho] \rangle$ , using that  $\langle X|S[\rho] \rangle = \langle X|S[\rho \setminus V_S] \rangle$  and  $\langle \gamma(X)|S[\gamma][\rho] \rangle = \langle \gamma(X)|S[\gamma][\rho \setminus \gamma(V_S)] \rangle = \langle \gamma(X)|S[\rho \setminus V_S][\gamma] \rangle$ . So  $\llbracket \langle \gamma(X)|S[\gamma] \rangle \rrbracket_P(\rho) \sim \llbracket \langle X|S \rangle \rrbracket_P(\rho)$ .

Let  $S : V_S \rightarrow \mathbb{T}^r(\Sigma)$  with  $X \in V_S \subseteq \text{Var}$ . Take  $\rho : \text{Var} \rightarrow \mathbb{G}(A)$ . Let  $\mathbb{G}^*$  be adequate for  $\llbracket \langle X|S \rangle \rrbracket_P(\rho)$  and  $\llbracket S_X|S \rrbracket_P(\rho)$ . Then  $\langle X|S[\rho \setminus V_S] \rangle \sim_{P+\mathbb{G}^*}^{\text{cl}} \langle S_X[\rho \setminus V_S]|S[\rho \setminus V_S] \rangle$  by Requirement (5') for the closed-term semantics of  $P+\mathbb{G}^*$ . Hence  $\llbracket \langle X|S \rangle \rrbracket_P(\rho) = \llbracket \langle X|S[\rho] \rangle \rrbracket_{P+\mathbb{G}^*}^0 = \llbracket \langle X|S[\rho \setminus V_S] \rangle \rrbracket_{P+\mathbb{G}^*}^0 \sim \llbracket \langle S_X[\rho \setminus V_S]|S[\rho \setminus V_S] \rangle \rrbracket_{P+\mathbb{G}^*}^0 = \llbracket \langle S_X|S \rangle \rrbracket_P(\rho)$ .  $\square$

Theorem 2 does not extend to sanity requirement (1). In fact, this requirement always holds for the closed-term interpretation of a TSS, yet it does not always hold for the process graph interpretation:

**Example 5** Let  $P$  be a TSS with the single rule  $\frac{x \xrightarrow{\tau} y, y \xrightarrow{\tau} z}{x \xrightarrow{\tau} z}$ . Then the process graph semantics of  $P$  fails to satisfy sanity requirement (1) up to  $\xrightarrow{\tau}$ . Namely, if  $\rho(x)$  is a graph  $\xrightarrow{\tau} \circ \xrightarrow{\tau} \circ \xrightarrow{\tau} \circ$ , then  $\llbracket x \rrbracket_P(\rho)$  is the graph  $\xrightarrow{\tau} \circ \xrightarrow{\tau} \circ \xrightarrow{\tau} \circ$ , and the two graphs are not (strongly) bisimulation equivalent.

Nevertheless, requirement (1) holds for almost all process algebras found in the literature:

**Proposition 6** Let  $P$  be a TSS that has no rule with a variable as the left-hand side of the conclusion. The process-graph interpretation of  $P$  always satisfies requirement (1) of Section 3 up to  $\xrightarrow{\tau}$ .

**Proof:** For  $x \in \text{Var}$  and  $\rho : \text{Var} \rightarrow \mathbb{G}(A)$  let  $\mathbb{G}^* \subseteq \mathbb{G}(A)$  be adequate for  $\llbracket x \rrbracket_P(\rho)$  and let  $\rho(x) \in \mathbb{G}^*$  be the graph  $g = (S, \rightarrow, I)$ . For each state  $s \in S$  there is a graph  $g_s := (S, \rightarrow, s)$  in  $\mathbb{G}^*$ . Now  $\llbracket x \rrbracket_P(\rho)$  is the reachable part of the graph  $(G, \rightarrow_G, g)$ , where  $G = \{g_s \mid s \in S\}$  and  $\rightarrow_G = \{(g_s, a, g_t) \mid (s, a, t) \in \rightarrow\}$ . The relation  $\mathcal{R}$  given by  $g_s \mathcal{R} s$  for all states  $s \in S$  reachable from  $I$  clearly is a bisimulation. Therefore  $\llbracket x \rrbracket_P(\rho) \xrightarrow{\tau} \rho(x)$ .  $\square$

## 12 Preservation of Relative Expressiveness under Conservative Extensions

**Definition 13** If  $P = (\Sigma_P, A, R_P)$  and  $Q = (\Sigma_Q, A, R_Q)$  are TSSs with  $\Sigma_P$  and  $\Sigma_Q$  disjoint then  $P+Q$  denotes the TSS  $(\Sigma_P \cup \Sigma_Q, A, R_P \cup R_Q)$ .

Let  $P_0$  be the TSS of Example 1, but without the operator  $f$ . Let  $P_f$  be the part of the TSS of Example 1 that only contains the operator  $f$ , so that the entire TSS of Example 1 is  $P_0 + P_f$ . This TSS does not feature recursion.

A *translation* between two languages with signatures  $\Sigma$  and  $\Sigma'$  is a mapping  $\mathcal{T} : \mathbb{T}^r(\Sigma) \rightarrow \mathbb{T}^r(\Sigma')$ . Consider the translation  $\mathcal{T}_{id}$  from the language specified by  $P_0$  to the language specified by  $P_0 + P_f$ , given by  $\mathcal{T}_{id}(t) = t$  for all  $t \in \Sigma_{P_0}$ . Also consider the translation  $\mathcal{T}_{op}$  in the opposite direction, given by  $\mathcal{T}_{op}(a.t) := a.\mathcal{T}_{op}(t)$ ,  $\mathcal{T}_{op}(id(t)) := id(\mathcal{T}_{op}(t))$  and  $\mathcal{T}_{op}(f(t)) := id(\mathcal{T}_{op}(t))$  for all  $t \in \mathbb{T}(\Sigma_{P_0+P_f})$ .

In e.g. [14] a concept of expressiveness of specification languages is studied such that language  $\mathcal{L}'$  is at least as expressive as language  $\mathcal{L}$  up to a semantic equivalence relation  $\sim$  iff there exist a translation from  $\mathcal{L}$  into  $\mathcal{L}'$  that is valid up to  $\sim$ . It is not important here to state the precise definition of validity; it suffices to point out that  $\mathcal{T}_{op}$  is valid up to  $\xrightarrow{\tau}$  iff one has  $f(x) \xrightarrow{\tau} id(x)$ . Thus, applying Definition 12, it is valid when employing the closed-term semantics, but not when employing the process graph semantics.



The transition  $\mathcal{T}_{id}$  on the other hand is valid up to  $\Leftrightarrow$  regardless which of the two interpretations one picks. So under the closed-term interpretation the two languages are equally expressive, whereas under the process graph semantics the language given by  $P_0 + P_f$  is more expressive than the one given by  $P_0$ .

For TSSs  $P$  and  $Q$ , write  $P \preceq Q$  if the language specified by  $Q$  is at least as expressive as the one specified by  $P$ . An intuitively plausible theorem is that

$$P_1 \preceq P_2 \quad \text{implies} \quad P_1 + Q \preceq P_2 + Q, \quad (9)$$

at least under some mild conditions on the TSSs  $P_1$ ,  $P_2$  and  $Q$ , for instance that they are pure and fit the *tyft* format defined in [16].<sup>3</sup> This theorem fails when employing the closed-term semantics of TSSs: take  $P_1$  to be  $P_0 + P_f$ ,  $P_2$  to be  $P_0$ , with  $\mathcal{T}_{op}$  being the witness for  $P_1 \preceq P_2$ , and  $Q$  to be the TSS with as single operator  $\tau_{\cdot}$  and as only transition rule  $\tau.x \xrightarrow{\tau} x$ . For the operator  $f$  in the TSS  $P_0 + P_f + Q$  drops  $\tau$ -transitions, and has no counterpart in the TSS  $P_0 + Q$ .

This problem is fixed when employing the process graph semantics. Once the omitted definition of validity is supplied [14], the proof of (9) is entirely straightforward.

### 13 Related Work

Dissatisfaction with the traditional closed-term interpretation of TSSs occurred earlier in [17, 18, 8, 23] and [2]. However, rather than adapting the interpretation of TSSs, as in the present paper, these papers abandon the notion of a TSS in favour of different frameworks of system specification that are arguably more suitable for giving meaning to open terms. Larsen and Liu [17] use *context systems*. The CCS transition rule

$$\frac{x \xrightarrow{a} x' \quad y \xrightarrow{\bar{a}} y'}{x|y \xrightarrow{\tau} x'|y'}$$

for instance takes in a context system the shape  $x|y \xrightarrow{\tau}_{a,\bar{a}} x'|y'$ , or rather, suppressing the redundant variable names,  $|\frac{\tau}{a,\bar{a}}|$ . It says that the operator  $|$  can perform a  $\tau$ -transition, provided its first argument does an  $a$ -transition, and its second argument an  $\bar{a}$ . The context systems of [17] form the counterpart of TSSs in the De Simone format [24]. The model is generalised by Lynch & Vaandrager [18] to *action transducers*, by Gadducci & U. Montanari [8] to the *tile model*, and by Rensink [23] to *conditional transition systems*. The latter two proposals are further generalised to *symbolic transition systems* by Baldan, Bracciali & Bruni [2].

One method to relate these models with TSSs under the closed-term and process graph interpretations is through notions of strong bisimilarity on open terms. This is a central theme in [23]. The most natural notion of bisimulation on the above models is *bisimulation under formal hypothesis*,  $\Leftrightarrow^{\text{fh}}$ . That name stems from De Simone [24], who defined the same concept in terms of TSSs. On the context systems sketched above it requires the usual transfer property for bisimulations for doubly labelled transitions such as  $\frac{\tau}{a,\bar{a}}$ . On TSSs, a bisimulation under formal hypothesis essentially is a symmetric relation  $\mathcal{R}$  on open terms such that

$$\text{if } t \mathcal{R} u \text{ and } P \vdash \frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{t \xrightarrow{a} t'} \text{ then } P \vdash \frac{\{x_i \xrightarrow{a_i} y_i \mid i \in I\}}{u \xrightarrow{a} u'} \text{ for an } u' \in \mathbb{T}^f(\Sigma_Q) \text{ with } t' \mathcal{R} u'.$$

Rensink [23] shows that  $\Leftrightarrow^{\text{fh}}$  is strictly finer than  $\Leftrightarrow^{\text{ci}}$ .

**Example 6** Let  $P$  be the TSS with inaction 0, action prefix, choice and intersection, specified by the following rules:

$$a.x \xrightarrow{a} x \quad \frac{x \xrightarrow{a} x'}{x+y \xrightarrow{a} x'} \quad \frac{y \xrightarrow{a} y'}{x+y \xrightarrow{a} y'} \quad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \cap y \xrightarrow{a} x' \cap y'}$$

<sup>3</sup>These mild conditions should ensure that  $P_i + Q$  is a *conservative extension* of  $P_i$ , for  $i = 1, 2$ , as defined in [16].

where  $a$  ranges over a set of actions  $A$ . Then  $b.0 + b.a.0 + b.(x \cap a.0) \xleftrightarrow{\text{ci}} b.0 + b.a.0$ , for no matter what one fills in for  $x$ , the process  $x \cap a.0$  either cannot perform any transitions, or it can only do a single  $a$ . So the term  $b.(x \cap a.0)$  behaves either like  $b.0$  or like  $b.a.0$ . On the other hand,  $b.0 + b.a.0 + b.(x \cap a.0) \not\xleftrightarrow{\text{fh}} b.0 + b.a.0$ . Namely any bisimulation under formal hypothesis  $\mathcal{R}$  relating  $b.0 + b.a.0 + b.(x \cap a.0)$  with  $b.0 + b.a.0$  would also have to relate  $x \cap a.0$  with either  $0$  or  $a.0$ . However, once this choice is made, substituting the wrong value for  $x$  shows that  $\mathcal{R}$  relates two terms that are not equivalent.

Rensink also defines a *hypotheses-preserving bisimulation equivalence*  $\xleftrightarrow{\text{hp}}$  on open terms, which is situated strictly between  $\xleftrightarrow{\text{fh}}$  and  $\xleftrightarrow{\text{ci}}$ . One has  $b.0 + b.a.0 + b.(x \cap a.0) \not\xleftrightarrow{\text{hp}} b.0 + b.a.0$ . His analysis can be reused to show that  $\xleftrightarrow{\text{hp}}$  is finer than  $\xleftrightarrow{\text{pg}}$ . Note that  $b.0 + b.a.0 + b.(x \cap a.0) \xleftrightarrow{\text{pg}} b.0 + b.a.0$ , for the same reasons as in Example 6. Thus, under the conditions of Theorem 1, we arrive at a hierarchy

$$\xleftrightarrow{\text{fh}} \subseteq \xleftrightarrow{\text{hp}} \subseteq \xleftrightarrow{\text{pg}} \subseteq \xleftrightarrow{\text{ci}} .$$

## 14 Concluding Remarks

This paper proposed a process graph semantics of TSSs as an alternative to the traditional closed-term semantics. It interprets an operator from the language as an operation on process graphs. Unlike the closed-term semantics, this interpretation is independent of the selection of processes that are expressible in the TSS as a whole. The intuitively plausible statement that an expressiveness inclusion between languages is preserved under a conservative extension of source and target language alike, fails for the closed-term semantics but holds for the proposed process graph semantics.

I reviewed five sanity requirements on languages equipped with a semantic equivalence relation  $\sim$ , and showed that four of them hold under the process semantics of a language if they hold under the closed-term semantics. Here I end with a few observations on when these requirements hold at all.

In [13], the *ntyft/ntyxt format with recursion* is introduced. It defines a wide class of TSSs, containing many known process algebras, including CCS, CSP, ACP, MEIJE and SCCS. It generalises the ntyft/ntyxt format of [15] by the addition of recursion as a separate language construct. The *tyft/tyxt format with recursion* is the same, but not allowing negative premises. [13] shows that all languages specified by a TSS in the ntyft/ntyxt format with recursion satisfy property (8) up to  $\xleftrightarrow{\sim}$ , saying that strong bisimilarity is a congruence. This is a stronger property than (2) up to  $\xleftrightarrow{\sim}$ , which thus also holds for the ntyft/ntyxt format. This was shown for the closed-term interpretation of TSSs. By Theorem 2 we now also have (2) up to  $\xleftrightarrow{\sim}$  for the process graph semantics of pure TSSs in the ntyft/ntyxt format with recursion.

The same paper establishes that (3) holds up to  $\xleftrightarrow{\sim}$  for the closed-term semantics of all TSSs in the tyft/tyxt format with recursion, thereby generalising a result from [23]. It thus also holds for the process graph semantics of all pure TSSs in the tyft/tyxt format with recursion.

It is not hard to show that also requirements (4) and (5) hold up to  $\xleftrightarrow{\sim}$  for the closed-term interpretation of TSSs in the ntyft/ntyxt format with recursion, and thus for the process graph semantics of pure TSSs in the ntyft/ntyxt format with recursion.

Thanks to the equational nature of requirements (1), (4) and (5), once they hold up to  $\xleftrightarrow{\sim}$ , they surely hold up to any coarser equivalence. This covers most semantic equivalences found in the literature. The same cannot be said for requirements (2) and (3). These need to be reestablished for each semantic equivalence. There is a lot of work on congruence formats, ensuring (2) for a variety of semantic equivalence. See for instance [7], and references therein. Yet, besides [23] and [13] I know of no congruence formats targeting requirement (3).

**Acknowledgement** My thanks to the referees for their thorough proofreading and helpful suggestions.

## References

- [1] D. Austry & G. Boudol (1984): *Algèbre de processus et synchronisations*. *Theoretical Computer Science* 30(1), pp. 91–131, doi:10.1016/0304-3975(84)90067-7.
- [2] P. Baldan, A. Bracciali & R. Bruni (2007): *A semantic framework for open processes*. *Theoretical Computer Science* 389(3), pp. 446–483, doi:10.1016/j.tcs.2007.09.004.
- [3] J.A. Bergstra & J.W. Klop (1984): *The algebra of recursively defined processes and the algebra of regular processes*. In J. Paredaens, editor: *Proceedings 11<sup>th</sup> ICALP, Antwerpen, LNCS 172*, Springer, pp. 82–94, doi:10.1007/3-540-13345-3\_7.
- [4] E. Bres, R.J. van Glabbeek & P. Höfner (2016): *A Timed Process Algebra for Wireless Networks with an Application in Routing*. Technical Report 9145, NICTA. Available at <http://arxiv.org/abs/1606.03663>. Extended abstract in P. Thiemann, editor: *Programming Languages and Systems: Proceedings 25th European Symposium on Programming, ESOP'16; held as part of the European Joint Conferences on Theory and Practice of Software, ETAPS'16, LNCS 9632*, Springer, 2016, pp. 95–122.
- [5] S.D. Brookes, C.A.R. Hoare & A.W. Roscoe (1984): *A theory of communicating sequential processes*. *Journal of the ACM* 31(3), pp. 560–599, doi:10.1145/828.833.
- [6] W.J. Fokkink (2000): *Introduction to Process Algebra*. Texts in Theoretical Computer Science, An EATCS Series, Springer, doi:10.1007/978-3-662-04293-9.
- [7] W.J. Fokkink, R.J. van Glabbeek & B. Luttik (2017): *Divide and Congruence III: Stability & Divergence*. In R. Meyer & U. Nestmann, editors: *Proceedings 28th International Conference on Concurrency Theory, CONCUR'17, Leibniz International Proceedings in Informatics (LIPIcs) 85*, Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 15:1–15:16, doi:10.4230/LIPIcs.CONCUR.2017.15. Available at <http://theory.stanford.edu/~rvg/abstracts.html#124>.
- [8] F. Gadducci & U. Montanari (2000): *The tile model*. In G.D. Plotkin, C. Stirling & M. Tofte, editors: *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, The MIT Press, pp. 133–166.
- [9] R.J. van Glabbeek (1993): *Full abstraction in structural operational semantics (extended abstract)*. In M. Nivat, C. Rattray, T. Rus & G. Scollo, editors: *Proceedings 3<sup>rd</sup> International Conference on Algebraic Methodology and Software Technology, AMAST'93*, Twente, The Netherlands, June 1993, Workshops in Computing, Springer, pp. 77–84. Available at <http://theory.stanford.edu/~rvg/abstracts.html#28>.
- [10] R.J. van Glabbeek (1994): *On the expressiveness of ACP (extended abstract)*. In A. Ponse, C. Verhoef & S.F.M. van Vlijmen, editors: *Proceedings First Workshop on the Algebra of Communicating Processes, ACP94*, Workshops in Computing, Springer, pp. 188–217, doi:10.1007/978-1-4471-2120-6\_8. Available at <http://theory.stanford.edu/~rvg/abstracts.html#31>.
- [11] R.J. van Glabbeek (2004): *The Meaning of Negative Premises in Transition System Specifications II*. *Journal of Logic and Algebraic Programming* 60–61, pp. 229–258, doi:10.1016/j.jlap.2004.03.007. Available at <http://theory.stanford.edu/~rvg/abstracts.html#53>.
- [12] R.J. van Glabbeek (2011): *Bisimulation*. In D. Padua, editor: *Encyclopedia of Parallel Computing*, Springer, pp. 136–139, doi:10.1007/978-0-387-09766-4\_149. Available at <http://theory.stanford.edu/~rvg/abstracts.html#45>.
- [13] R.J. van Glabbeek (2017): *Lean and Full Congruence Formats for Recursion*. In: *Proceedings 32<sup>nd</sup> Annual ACM/IEEE Symposium on Logic in Computer Science, LICS'17*, Reykjavik, Iceland, 2017, IEEE Computer Society Press, doi:10.1109/LICS.2017.8005142. Available at <https://arxiv.org/abs/1704.03160>.
- [14] R.J. van Glabbeek (2018): *A Theory of Encodings and Expressiveness*. In C. Baier & U. Dal Lago, editors: *Proceeding 21st International Conference on Foundations of Software Science and Computational Structures, FoSSaCS'18; held as part of the European Joint Conferences on Theory and Practice of Software, ETAPS'18, LNCS 10803*, Springer, pp. 183–202, doi:10.1007/978-3-319-89366-2\_10.
- [15] J.F. Groote (1993): *Transition System Specifications with Negative Premises*. *Theoretical Computer Science* 118, pp. 263–299, doi:10.1016/0304-3975(93)90111-6.

- [16] J.F. Groote & F.W. Vaandrager (1992): *Structured Operational Semantics and Bisimulation as a Congruence*. *Information and Computation* 100(2), pp. 202–260, doi:10.1016/0890-5401(92)90013-6.
- [17] K.G. Larsen & X. Liu (1991): *Compositionality through an Operational Semantics of Contexts*. *Journal of Logic and Computation* 1(6), pp. 761–795, doi:10.1093/logcom/1.6.761.
- [18] N.A. Lynch & F.W. Vaandrager (1996): *Action Transducers and Timed Automata*. *Formal Aspects of Computing* 8(5), pp. 499–538, doi:10.1007/BF01211907.
- [19] Y.I. Manin (1977): *A Course in Mathematical Logic*. *Graduate Texts in Mathematics* 53, Springer, doi:10.1007/978-1-4757-4385-2.
- [20] R. Milner (1983): *Calculi for synchrony and asynchrony*. *Theoretical Computer Science* 25(3), pp. 267–310, doi:10.1016/0304-3975(83)90114-7.
- [21] R. Milner (1990): *Operational and algebraic semantics of concurrent processes*. In J. van Leeuwen, editor: *Handbook of Theoretical Computer Science*, chapter 19, Elsevier Science Publishers B.V. (North-Holland), pp. 1201–1242. Alternatively see *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, 1989, of which an earlier version appeared as *A Calculus of Communicating Systems*, LNCS 92, Springer, 1980.
- [22] G.D. Plotkin (2004): *A Structural Approach to Operational Semantics*. *Journal of Logic and Algebraic Programming* 60–61, pp. 17–139, doi:10.1016/j.jlap.2004.05.001. Originally appeared in 1981.
- [23] A. Rensink (2000): *Bisimilarity of Open Terms*. *Information and Computation* 156(1-2), pp. 345–385, doi:10.1006/inco.1999.2818.
- [24] R. de Simone (1985): *Higher-level synchronising devices in MEIJE-SCCS*. *Theoretical Computer Science* 37, pp. 245–267, doi:10.1016/0304-3975(85)90093-3.