Transforming SOS specifications to linear processes

Citation for published version (APA):

Stappers, F. P. M., Reniers, M. A., & Weber, S. (2011). *Transforming SOS specifications to linear processes*. (Computer science reports; Vol. 1107). Technische Universiteit Eindhoven.

Document status and date:

Published: 01/01/2011

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Transforming SOS Specifications to Linear Processes

F.P.M. Stappers¹, M.A. Reniers², and S. Weber³

¹ Department of Computer Science, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands

² Department of Mechanical Engineering, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands

³ Department for Architecture and Platform, ASML,
P.O. Box 324, NL-5500 AH Veldhoven, The Netherlands

Abstract. This paper describes an approach to transform a Structural Operational Semantics given as a set of deduction rules to a Linear Process Specification. The transformation is provided for deduction rules in De Simone format, including predicates. The Linear Process Specifications are specified in the syntax of the mCRL2 language, that, with help of the underlying (higher-order) re-writer/tool-set, can be used for simulation, labeled transition system generation and verification of behavioral properties. We illustrate the technique by showing the effect of the transformation from the Structural Operational Semantics specification of a simple process algebra to a Linear Process Specification.

1 Introduction

The behavior of a *system* can be analyzed in various ways. It can be achieved by observing output from simulations, or by examining the behavioral descriptions (*e.g.*, code of a controller). To perform such an analysis, one always requires syntax (the way to denote behavior), semantics (the way in which grammatically correct behavior is executed) and a relationship between the two.

One way for describing the formal execution of a system, is to use *Structural Operational Semantics* (SOS) [34]. There, semantics is assigned to syntax, by means of deduction rules that describe the allowed set of actions of a piece of syntax. Unfortunately, there are hardly any suitable automated transformations from SOS specifications, along with a syntactical instance, to languages that can be subjected to formal analysis.

In this paper, we address this gap by formulating a systematic approach by which the deduction rules specified in SOS, along with the signature of the syntax, are transformed into a symbolic representation of a labeled transition system, called a Linear Process Specification (LPS) [6, 20]. The LPS can later be subjected to formal analysis (*e.g.*, simulation, explicit labeled transition system generation, and verification). We restrict the deduction rules to the De Simone-format [18].

We have chosen LPS as a target formalism, because it (i) has a mathematical representation that strongly relates to deduction rules in SOS and (ii) can be directly implemented in the mCRL2 language [22, 27]. In fact, LPS serves as a backbone for the representation and manipulation of behavioral models in the mCRL2 tool-set. Since this tool-set facilitates a higher-order term rewrite system, a transition generator and other transformation tools, we are able to exhaustively explore the state space and conduct profound analyses.

The framework aims at the transformation of formal behavioral specifications to specifications suitable for analysis, *e.g.*, simulation and model-checking. The technique can be used when prototyping Formal Domain Specific Languages, to investigate behavior dictated by the underlying operational semantics or to automate translations of formal languages towards the mCRL2 tool-set.

Outline Section 2 describes the preliminaries on SOS and LPS. Section 3 describes the transformation of the signature and SOS of a language to an LPS. Section 4 provides a small but nevertheless illustrative example. Section 5 discusses discrepancies between the presentation and implementation. In Section 6, we discuss extensions of the framework with amongst others predicates. In Section 7, we position this work. Section 8 concludes and elaborates on future work.

2 Preliminaries

2.1 Structural Operational Semantics

Structural Operational Semantics (SOS) defines the possible actions that a piece of syntax can perform. SOS is typically represented by a transition system specification (TSS) [8]. The syntax for which the semantics is defined, is represented by a signature. A signature fixes the composition operators and their corresponding arities, where a function with arity zero represents a constant. We assume sets of variables $\mathcal{V}$ and action labels $\mathcal{A}$.

The definitions of signature and transition system specification provided below are restricted to signatures with a single sort and transition system specifications with a single transition relation symbol.

A *signature* Σ is a collection of function symbols together with their arities. The arity of a function symbol $f \in \Sigma$ is denoted $ar(f)$. The collection of *terms* over signature Σ , denoted $\mathcal{T}(\Sigma)$, is the smallest set such that (i) a variable $x \in \mathcal{V}$ is a term, and (ii) if $t_1, \dots, t_n$ are terms and $f \in \Sigma$ is an n -ary function symbol, then $f(t_1, \dots, t_n)$ is a term. The set of *closed terms* over signature Σ , denoted $\mathcal{C}(\Sigma)$, is the set of all terms over Σ in which no variables occur. The *variables* that occur in a term p are denoted by $vars(p)$. A *transition formula* is of the form $p \xrightarrow{l} p'$ for $p, p' \in \mathcal{T}(\Sigma)$ and $l \in \mathcal{A}$.

A *transition system specification* (TSS) is a tuple $(\Sigma, \mathcal{D})$ where Σ is a signature and $\mathcal{D}$ is a set of deduction rules. A deduction rule is of the form $\frac{H}{C}$ where H is a set of transition formulas, called the set of *premises* and C is a transition formula, called the *conclusion*.

To illustrate our technique, we only consider TSSs that consist of deduction rules of a specific form; we restrict to TSSs in the De Simone-format [18]. A TSS $(\Sigma, \mathcal{D})$ is in De Simone format, if every deduction rule $d \in \mathcal{D}$ complies to the following form:

$$\frac{\{x_i \xrightarrow{l_i} y_i \mid i \in I\}}{f(x_1, \dots, x_{ar(f)}) \xrightarrow{l} t} [Cond_d]$$

where all of $x_1, \dots, x_{ar(f)}$ and y_i , for $i \in I$ are distinct variables, $f \in \Sigma$, $I \subseteq \{1, \dots, ar(f)\}$, and t is a process term that only contains variables from $\{x_j \mid j \notin I\} \cup \{y_i \mid i \in I\}$ and does not have repeated occurrences of variables, l_i 's and l are labels and $Cond_d$ is a condition on the labels of the premises and the label of the conclusion. A TSS defines a set of transitions, a so-called transition relation; see, *e.g.*, [2, 33] for formal definitions thereof.

2.2 (Simplified) Linear Process Specifications

In this paper we transform a TSS to an LPS. Informally, an LPS consists of a signature, variable declarations, a collection of data equations, action declarations, a linear process equation, and an initialization. An LPS can be viewed as a symbolic representation for (possible infinite) labeled transition systems. A formal definition of a Linear Process Specification and its components can be found in [22].

A *signature* is a triple $(\mathcal{S}, \mathcal{C}, \mathcal{M})$ where

1. $\mathcal{S}$ is a set of *sort names*, a non-empty (possible infinite) set of data elements.
2. $\mathcal{C}$ is a set of constructor function declarations of the form $f: S_1 \times \dots \times S_n \rightarrow S$ with $S_1, \dots, S_n, S \in \mathcal{S}$. Constructor functions are functions by which exactly all elements in the sort can be denoted.
3. $\mathcal{M}$ is a set of mapping declarations of the form $f: S_1 \times \dots \times S_n \rightarrow S$ with $S_1, \dots, S_n, S \in \mathcal{S}$. Mapping functions define auxiliary functions to rewrite terms of a sort.

The sets $\mathcal{C}$ and $\mathcal{M}$ are disjoint.

A variable declaration is of the form $x_1, \dots, x_n: S$ where the x_i are variable names and S is a sort name. From the signature and the variable declarations, terms (of a certain sort) can be constructed. A data equation is of the form $p = p'$ where p and p' are terms of the same sort.

A linear process equation (LPE) is an equation of the form:

$$X(d:D) = \sum_{i \in I} \sum_{e_i: E_i} c_i(d, e_i) \rightarrow a_i(d, e_i) \cdot X(g_i(d, e_i))$$

where I is a finite index set of summand variables, where for $i \in I$ holds:

- $c_i(d, e_i)$ is a term of sort $\mathbb{B}$ (denoting the set of Booleans) that serves as a Boolean guard to allow actions,
- $a_i(d, e_i) \in \mathcal{A}$,

- $g_i(d, e_i)$ is a term of sort D that denotes the next state,
- e_i and E_i denote a variable name and a sort expression, respectively.

The original definition of an LPE allows more features such as actions with data parameters, time annotations, termination, *etc.*, which are not needed in this paper and are therefore omitted. The initialization is a statement of the form $X(p)$, where p is an (open) term of sort D .

3 Method

We provide a template that transforms a TSS (in the De Simone format) to an LPS. This LPS is described in mCRL2 notation, which is a symbolic description of the transition relation (transition system) described by the TSS. In order to directly implement it as an mCRL2 specification, we sometimes slightly deviate from notations that are common in mathematics, (*e.g.*, when denoting a set comprehension). The framework that we present is restricted to the use of *mCRL2-restrictive* TSSs, as defined below. The method is illustrated by an example in Section 4.

Definition 1 (mCRL2-restrictive TSS). *A TSS is mCRL2-restrictive if*

1. *the signature Σ contains finitely many function symbols,*
2. *the set of labels $\mathcal{A}$ is finite,*
3. *the set of deduction rules $\mathcal{D}$ is finite, and*
4. *the conditions of the deduction rules can be represented in mCRL2.*

In Section 6, we discuss possibilities for relaxing some of these restrictions.

3.1 Signature transformation

For a signature Σ that consists of different function symbols $f_1, \dots, f_n$, we define a sort $\mathcal{T}$ together with some additional functions in the mCRL2 language by:

sort $\mathcal{T} =$ **struct** $f_1(\pi_1 : \mathcal{T}, \dots, \pi_{ar(f_1)} : \mathcal{T}) ? is_{f_1}$
 $\vdots$
 $| f_n(\pi_1 : \mathcal{T}, \dots, \pi_{ar(f_n)} : \mathcal{T}) ? is_{f_n};$

For terms of this sort, $f_1, \dots, f_n \in \mathcal{C}$ are the constructor functions. The projection functions $\pi_i \in \mathcal{M}$ are used to retrieve argument i of a function symbol. These functions are defined by the equations $\pi_i(f(x_1, \dots, x_{ar(f)})) = x_i$ in case $i \leq ar(f)$ and undefined otherwise. The recognizer functions $is_{f_i} \in \mathcal{M}$ facilitate the evaluation whether a term is of a particular form. The equations defining recognizer function is_{f_i} are $is_{f_i}(f_i(x_0, \dots, x_{ar(f_i)-1})) = true$ and $is_{f_i}(f_j(x_0, \dots, x_{ar(f_j)-1})) = false$ for $i \neq j$. For a detailed description about sorts in the mCRL2 language consider [22]. For sort $\mathcal{T}$ equality is denoted by $\approx$.

3.2 Transitions

The structured sort *Solution* is introduced to model pairs of a label and a term. We assume that the set of action labels, say $\{a_1, \dots, a_n\}$, is represented by a sort $\mathcal{A}$.

sort $\mathcal{A} = \mathbf{struct} \ a_1 \mid \dots \mid a_n;$
sort *Solution* = **struct** $sol(\pi_l: \mathcal{A}, \pi_t: \mathcal{T});$

The projection functions π_l and π_t are used to retrieve the transition label and process term from a solution, respectively.

We introduce a function R that satisfies the property, for all $s, s' \in \mathcal{C}(\Sigma)$ and labels $l \in \mathcal{A}$

$$sol(l, s') \in R(s) \quad \text{iff} \quad s \xrightarrow{l} s'$$

Since every transition is derivable due to a specific lastly applied deduction rule, this is accomplished by introducing a function $R_d: \mathcal{T} \rightarrow \text{Set}(\text{Solution})$ for each deduction rule d of the TSS. Then, for $\mathcal{D} = \{d_1, \dots, d_n\}$, the function $R: \mathcal{T} \rightarrow \text{Set}(\text{Solution})$ is defined by means of the single equation

var $p: \mathcal{T};$
eqn $R(p) = R_{d_1}(p) \cup \dots \cup R_{d_n}(p);$

Consider a deduction rule d of the form

$$\frac{\{x_i \xrightarrow{l_i} y_i \mid i \in I\}}{f(x_1, \dots, x_{ar(f)}) \xrightarrow{l} t} [\text{Cond}_d(l_{i_1}, \dots, l_{i_{|I|}}, l)]$$

in the De Simone format, where $I = \{i_1, \dots, i_{|I|}\}$. The equation that is introduced for R_d is given next, followed by an informal explanation of its structure and the used auxiliary functions. Finally, their formal definitions are provided.

eqn $R_d(p) = \{ \ s: \text{Solution} \mid$
 $\quad is_f(p)$
 $\quad \wedge \sigma^t(\pi_t(s))$
 $\quad \wedge \exists_{l_{i_1}, \dots, l_{i_{|I|}}} (\text{Cond}_d(l_{i_1}, \dots, l_{i_{|I|}}, \pi_l(s))$
 $\quad \quad \wedge \bigwedge_{i \in I} y_i \in \text{vars}(t) \Rightarrow sol(l_i, \mu_{y_i}^t(\pi_t(s))) \in R(\pi_{i_i}(p))$
 $\quad \quad \wedge \bigwedge_{i \in I} y_i \notin \text{vars}(t) \Rightarrow \exists_{z_i: \mathcal{T}} sol(l_i, z_i) \in R(\pi_{i_i}(p)))$
 $\quad \wedge \bigwedge_{j \notin I} x_j \in \text{vars}(t) \Rightarrow \mu_{x_j}^t(\pi_t(s)) \approx \pi_j(p)$
 $\quad \};$

The conjunct $is_f(p)$ states that the rule can only be applied to terms p that are headed by function symbol f . The conjunct $\sigma^t(\pi_t(s))$ states that the target term must have the same structure as the term t from the deduction rule. The third, fourth and fifth conjunct state that labels l_i and terms y_i need to be found such that the condition and premises of the deduction rule are satisfied. Here, the third conjunct states that we require a solution that fulfills the condition. Then fourth and fifth conjunct restrict the possible solutions to those that agree

with the substitution for the occurrences of x_i and y_i in t to obtain $\pi_t(s)$. The expression $\mu_x^t(p)$ denotes the term (from p) that is used to instantiate variable x in t . The last condition checks that the substitutions used for the source variables, occurring in the target, are those provided by p .

Check target structure The resulting target term must be an instance of the term t . We define a function $\sigma^t: \mathcal{T} \rightarrow \mathbb{B}$ that checks this. If t is of the form x for some variable x then we introduce the following equation:

var $p: \mathcal{T}$;
eqn $\sigma^x(p) = \text{true}$;

and for t of the form $f(t_1, \dots, t_{ar(f)})$, for some function symbol f and terms $t_1, \dots, t_{ar(f)}$, we introduce the equation

var $p: \mathcal{T}$;
eqn $\sigma^{f(t_1, \dots, t_{ar(f)})}(p) = is_f(p) \wedge \sigma^{t_1}(\pi_1(p)) \wedge \dots \wedge \sigma^{t_{ar(f)}}(\pi_{ar(f)}(p))$;

and auxiliary functions $\sigma^{t_i}: \mathcal{T} \rightarrow \mathbb{B}$ with their corresponding equations.

Capture conditions The user of this framework has to introduce functions $Cond_d$ that capture the meaning of the conditions in the deduction rules. This means that applicability is restricted to such conditions that can be captured as Boolean expressions in the mCRL2 syntax.

map $Cond_d: \mathcal{A} \times \dots \times \mathcal{A} \times \mathcal{A} \rightarrow \mathbb{B}$;

For practical cases, these functions are easily captured in the mCRL2 data language.

Extract instance of a variable To retrieve the term that is used to instantiate a variable x in the term t , we introduce a projection function $\mu_x^t: \mathcal{T} \rightarrow \mathcal{T}$.

In case t is of the form x we introduce the equation

eqn $\mu_x^x(p) = p$

In case t is of the form $f(t_1, \dots, t_{ar(f)})$ for some function symbol f and terms $t_1, \dots, t_{ar(f)}$, we introduce an equation

eqn $\mu_x^{f(t_1, \dots, t_{ar(f)})}(p) = \mu_x^{t_i}(\pi_i(p))$

for each term t_i in which x occurs. Additionally we add the auxiliary functions $\mu_x^{t_i}: \mathcal{T} \rightarrow \mathcal{T}$ and their corresponding equations. Note that we only use μ_x^t in those cases where $x \in vars(t)$. Hence it does not matter that the function μ_x^t is not defined for variables different from x that do not occur in t . Since we only consider t in which every variable occurs at most once, μ_x^t is well-defined.

3.3 Linear process transition generator

Basically, transitions are performed as long as the set of solutions belonging to term p is non-empty. So we declare process X , with the process parameter $p: \mathcal{T}$. For each iteration, we select a solution s such that $s \in R(p)$ holds. Then, for s we need to dispatch the transition (e.g., $\pi_l(s)$) and update term p to be $\pi_t(s)$. Putting, it all together provides us with

$$\mathbf{proc} \ X(p: \mathcal{T}) = \sum_{s: \text{Solution}} s \in R(p) \rightarrow \pi_l(s) \cdot X(\pi_t(s));$$

To obtain the behavior associated with a particular term p , we consider the process $X(p)$:

init $X(p)$;

The following theorem expresses the correspondence between the labeled transition systems associated with the closed process term and the mCRL2 process $X(p)$. A proof of this theorem can be found in [35].

Theorem 1 (Correspondence). *Let $(\Sigma, \mathcal{D})$ be an mCRL2-restrictive TSS in the De Simone format. Then for every $p \in \mathcal{C}(\Sigma)$, the labeled transition system associated with p and the labeled transition system associated with $X(p)$ are isomorphic.*

4 Application

To illustrate our approach we consider the process algebra MPT from [4] extended with an interleaving parallel composition operator. Assume a finite set of actions $\mathcal{A} = \{a_0, \dots, a_n\}$. The signature of this language consists of the nullary function symbol 0, the unary function symbols α_- (for $\alpha \in \mathcal{A}$, $-$ denoting the argument), and the binary function symbols $- + -$ and $- \parallel -$. In this section we will use infix notation for the binary function symbols.

When applying the signature transformation we get:

sort $\mathcal{T} = \mathbf{struct}$ $zero?is_{zero} \mid a_0(\pi_1: \mathcal{T})?is_{a_0} \mid \dots \mid a_n(\pi_1: \mathcal{T})?is_{a_n}$
 $\mid alt(\pi_1: \mathcal{T}, \pi_2: \mathcal{T})?is_{alt} \mid par(\pi_1: \mathcal{T}, \pi_2: \mathcal{T})?is_{par};$
sort $\mathcal{A} = \mathbf{struct}$ $a_1 \mid \dots \mid a_n;$
sort $Solution = \mathbf{struct}$ $sol(\pi_l: \mathcal{A}, \pi_t: \mathcal{T});$

where $zero$, a_i , alt and par represent 0, a_i , $+$, and $\parallel$ respectively. The deduction rules for this process algebra are:

$$\begin{array}{c} (a_0) \frac{}{a_0.x_1 \xrightarrow{a_0} x_1} \quad \dots \quad (a_n) \frac{}{a_n.x_1 \xrightarrow{a_n} x_1} \quad (\mathbf{a1}) \frac{x_1 \xrightarrow{l} y_1}{x_1 + x_2 \xrightarrow{l} y_1} \\ (\mathbf{a2}) \frac{x_2 \xrightarrow{l} y_2}{x_1 + x_2 \xrightarrow{l} y_2} \quad (\mathbf{p1}) \frac{x_1 \xrightarrow{l} y_1}{x_1 \parallel x_2 \xrightarrow{l} y_1 \parallel x_2} \quad (\mathbf{p2}) \frac{x_2 \xrightarrow{l} y_2}{x_1 \parallel x_2 \xrightarrow{l} x_1 \parallel y_2} \end{array}$$

As no conditions (other than *true*) appear in these deduction rules we do not consider them in the remainder of this section. To accommodate the (auxiliary) computation we introduce the following functions and variables:

map $R, R_{a_0}, \dots, R_{a_n}, R_{\mathbf{a1}}, R_{\mathbf{a2}}, R_{\mathbf{p1}}, R_{\mathbf{p2}}: \mathcal{T} \rightarrow \text{Set}(\text{Solution});$
 $\sigma^{x_1}, \sigma^{x_2}, \sigma^{y_1}, \sigma^{y_2}, \sigma^{y_1 \parallel x_2}, \sigma^{x_1 \parallel y_2}: \mathcal{T} \rightarrow \mathbb{B};$
 $\mu_{x_1}^{x_1}, \mu_{y_1}^{y_1}, \mu_{y_2}^{y_2}, \mu_{y_1}^{y_1 \parallel x_2}, \mu_{x_2}^{y_1 \parallel x_2}, \mu_{x_1}^{x_1 \parallel y_2}, \mu_{y_2}^{x_1 \parallel y_2}: \mathcal{T} \rightarrow \mathcal{T};$
var $v: \mathcal{T};$

The sort *Solution* refers to the declaration defined in Section 3.2. The overall relation function we define as:

eqn $R(v) = R_{a_0}(v) \cup \dots \cup R_{a_n}(v) \cup R_{\mathbf{a1}}(v) \cup R_{\mathbf{a2}}(v) \cup R_{\mathbf{p1}}(v) \cup R_{\mathbf{p2}}(v);$

Then the resulting equations for the action prefix terms are, for each $\alpha \in \mathcal{A}$

eqn $\sigma^{x_1}(v) = \text{true};$
 $\mu_{x_1}^{x_1}(v) = v;$
 $R_\alpha(v) = \{s: \text{Solution} \mid is_\alpha(v) \wedge \sigma^{x_1}(\pi_t(s)) \wedge \mu_{x_1}^{x_1}(\pi_t(s)) \approx \pi_1(v)\};$

The required equations for deduction rule **(a1)** are:

eqn $\sigma^{y_1}(v) = \text{true};$
 $\mu_{y_1}^{y_1}(v) = v;$
 $R_{\mathbf{a1}}(v) = \{s: \text{Solution} \mid is_{alt}(v) \wedge \sigma^{y_1}(\pi_t(s))$
 $\quad \wedge \exists l_1: \mathcal{A}(\text{sol}(l_1, \mu_{y_1}^{y_1}(\pi_t(s))) \in R(\pi_1(v)))\};$

For deduction rule **(p1)**, the following set of equations is constructed:

eqn $\sigma^{y_1 \parallel x_2}(v) = is_{par}(v) \wedge \sigma^{y_1}(\pi_1(v)) \wedge \sigma^{x_2}(\pi_2(v));$
 $\sigma^{y_1}(v) = \text{true};$
 $\sigma^{x_2}(v) = \text{true};$
 $\mu_{y_1}^{y_1 \parallel x_2}(v) = \mu_{y_1}^{y_1}(\pi_1(v));$
 $\mu_{y_1}^{y_1}(v) = v;$
 $\mu_{x_2}^{y_1 \parallel x_2}(v) = \mu_{x_2}^{x_2}(\pi_2(v));$
 $\mu_{x_2}^{x_2}(v) = v;$
 $R_{\mathbf{p1}}(v) = \{s: \text{Solution} \mid is_{par}(v) \wedge \sigma^{y_1 \parallel x_2}(\pi_t(s))$
 $\quad \wedge \exists l_1: \mathcal{A}(\text{sol}(l_1, \mu_{y_1}^{y_1 \parallel x_2}(\pi_t(s))) \in R(\pi_1(v))$
 $\quad \wedge \mu_{x_2}^{y_1 \parallel x_2}(\pi_t(s)) \approx \pi_2(v)\};$

The treatment of deduction rules **(a2)** and **(p2)** is analogous to the treatment of rules **(a1)** and **(p1)**.

To perform a meaningful analysis for the closed term p , we provide the following LPE, instantiated by p as:

proc $X(v: \mathcal{T}) = \sum_{s: \text{Solution}} s \in R(v) \rightarrow \pi_t(s) \cdot X(\pi_t(s));$
init $X(p);$

To illustrate that the method is effective, in Figure 1, we provide graphs generated by the mCRL2 tool-set (release-March 2011), that are obtained by applying the framework. In each case, the initial process parameter p from sort $\mathcal{T}$, which generates the labeled transition system, is provided in the caption below the the graphs. The tools that have been used to generate the pictures are subsequently `txt2lps` and `lps2lts`. The first tool reads a textual LPS and stores it into the binary LPS format. The second tool unfolds an LPS into a labeled transition system.

Fig. 1: Three different specifications, as generated by the mCRL2 tool-set

5 Implementation

In order to implement a specification, we require a finite number of deduction rules and a finite signature, such that we can generate a finite textual specification. Furthermore we need to apply two restrictions, in order to conduct an analysis. The first restriction applies to the use of actions. The second restriction applies to the use of quantifiers.

In the example we use elements of sort $\mathcal{A}$ (part of the data specification) as actions in mCRL2. Within mCRL2 language the direct use of data sorts as actions is prohibited. In fact, mCRL2 requires two separate declarations. To overcome this limitation, we declare a (dummy) action with a data parameter of sort $\mathcal{A}$ and use this data parameter to encode the SOS-action. So instead of $p \xrightarrow{a} p'$, we get $p \xrightarrow{\text{Trans}(a)} p'$, where *Trans* is the dummy action name carrying $\mathcal{A}$ as its parameter.

The second restriction applies to the use of quantifiers. The mCRL2 language allows the user to specify existential ($\exists$) quantifiers, but their evaluation within the tool-set is currently being developed. The existential quantification over the action labels can be dealt with by the tool-set since these concern a finite domain.

The existential quantifications over the z_i variables are not necessarily over a finite domain. The mCRL2 tool-set cannot compute these in all cases. However, the expressions $\exists_{z_i} \text{sol}(l_i, z_i) \in R(\pi_i(p))$ can be replaced by expressions $l_i \in$

$R^l(\pi_i(p))$, where the function R_l is like R but instead of returning a set of solutions, which consists of labels and terms, it returns only a set of labels. Let $R^l, (R_d^l)_{d \in \mathcal{D}}: \mathcal{T} \rightarrow \text{Set}(\mathcal{A})$ be the derived function along with its auxiliary functions. Then $R^l = \bigcup_{d \in \mathcal{D}} R_d^l$, where the auxiliary functions are defined as:

$$\text{eqn } R_d^l(p) = \{a: \mathcal{A} \mid \text{is}_f(p) \wedge \exists_{l_{i_1}, \dots, l_{i_{|I|}}} (\text{Cond}_d(l_{i_1}, \dots, l_{i_{|I|}}, a) \wedge \bigwedge_{i \in I} (l_i \in R^l(\pi_i(p)))) \};$$

6 Extension of the framework

Although we have not shown it in this paper, we believe that there are no reasons why the mCRL2 tool-set would not be able to deal with multi-sorted signatures. The main adaptation to the presented method is that for each sort in the signature a different mCRL2 sort needs to be defined.

Also extending the framework to deal with TSSs in which multiple transition relations occur is no problem. It requires defining a different function R for each transition relation in the TSS. Also different solution sorts must be provided based on the arities of the involved transition relations. mCRL2 requires that the sorts do not share function symbol names. The example in the end of this section illustrates the treatment of multiple transition relations.

In the previous section we have only dealt with sets of actions labels that are finite and hence easily captured by means of a structured sort in mCRL2. mCRL2 allows the use of much more involved sorts. As long as the label set can be captured as a sort in mCRL2 we can also deal with infinite label sets. A problem may arise in the impossibilities of the tool-set in dealing with the existential quantifications in that case.

In the remainder of this section we discuss how predicates can be dealt with. Predicates are a useful addition to TSSs [5]. Predicates are used to express behavioral properties, like termination and divergence. A deduction rule d in a TSS with transition relation symbol $\longrightarrow$ and predicate P is of the form:

$$\frac{\{x_i \xrightarrow{l_i} y_i \mid i \in I\} \cup \{Px_j \mid j \in J\}}{Pf(x_1, \dots, x_{ar(f)})} [\text{Cond}_d]$$

or

$$\frac{\{x_i \xrightarrow{l_i} y_i \mid i \in I\} \cup \{Px_j \mid j \in J\}}{f(x_1, \dots, x_{ar(f)}) \xrightarrow{l} t} [\text{Cond}_d]$$

where all of $x_1, \dots, x_{ar(f)}$ and y_i , for $i \in I$ are distinct variables, $f \in \Sigma$, $I, J \subseteq \{1, \dots, ar(f)\}$ and $I \cap J = \emptyset$, t is a process term that only contains variables from $\{x_k \mid k \notin I \cup J\} \cup \{y_i \mid i \in I\}$ and does not have repeated occurrences of variables, and l_i 's and l are labels and Cond_d is a condition on the labels of the premises and the conclusion (if any).

Predicates can be considered a special type of transition relation, with special transition labels. Therefore, we introduce a special transition relation symbol

$\xrightarrow{P}$ for each predicate P . Then the above deduction rules can be represented by:

$$\frac{\{x_i \xrightarrow{l_i} y_i \mid i \in I\} \cup \{x_j \xrightarrow{P} y_j \mid j \in J\}}{f(x_1, \dots, x_{ar(f)}) \xrightarrow{P} f(x_1, \dots, x_{ar(f)})} [Cond_d]$$

$$\frac{\{x_i \xrightarrow{l_i} y_i \mid i \in I\} \cup \{x_j \xrightarrow{P} y_j \mid j \in J\}}{f(x_1, \dots, x_{ar(f)}) \xrightarrow{l} t} [Cond_d]$$

Since we have assumed I and J to be disjoint these rules are in the De Simone format.

For transition relations we redefine function R to R_{Trans} , such that for all $s, s' \in \mathcal{C}(\Sigma)$ and labels $l \in \mathcal{A}_{Trans}$ holds:

$$sol(l, s') \in R_{Trans}(s) \quad \text{iff} \quad s \xrightarrow{l} s'$$

and for predicate relations we define the function R_{Pred} , such that for all $s \in \mathcal{C}(\Sigma)$ and labels $P \in \mathcal{A}_{Pred}$, holds:

$$sol(P, s) \in R_{Pred}(s) \quad \text{iff} \quad s \xrightarrow{P} s$$

Note that this predicates modeled in this way, appear as a self-loop transition in a labeled transition system. To emphasize the difference between action transitions and predicate transitions, we use action name *Trans* for action transitions and action name *Pred* for predicate transitions.

act $Trans: \mathcal{A}_{Trans};$
 $Pred: \mathcal{A}_{Pred};$

proc $X(p: T) = \sum_{s: \text{Solution}} s \in R_{Trans}(p) \rightarrow Trans(\pi_l(s)) \cdot X(\pi_t(s))$
 $+ \sum_{s: \text{Solution}} s \in R_{Pred}(p) \rightarrow Pred(\pi_l(s)) \cdot X(\pi_t(s));$

Predicate application In this example we extend the MPT with termination. By introducing termination, the signature as mentioned in Section 4, is extended with the function symbol 1. Within the MPT extension it is common to write $x \downarrow$ instead of $\downarrow x$. The deduction rules for this extension are:

$$\begin{array}{ccc} \text{(t1)} \frac{}{1 \downarrow} & \text{(t2)} \frac{x_1 \downarrow}{x_1 + x_2 \downarrow} & \text{(t3)} \frac{x_2 \downarrow}{x_1 + x_2 \downarrow} \end{array}$$

The deduction rules that we obtain by replacing the predicates by transition relations are the following:

$$\begin{array}{ccc} \text{(t1)} \frac{}{1 \xrightarrow{\downarrow} 1} & \text{(t2)} \frac{x_1 \xrightarrow{\downarrow} y_1}{x_1 + x_2 \xrightarrow{\downarrow} y_1 + x_2} & \text{(t3)} \frac{x_2 \xrightarrow{\downarrow} y_2}{x_1 + x_2 \xrightarrow{\downarrow} x_1 + y_2} \end{array}$$

Now, we first extend the signature by adding a nullary constructor function *one* representing the constant 1 and a recognizer function *is_{one}* as follows:

sort $\mathcal{T} = \mathbf{struct}$ $zero?is_{zero} \mid one?is_{one} \mid a_0(\pi_1: \mathcal{T})?is_{a_1} \mid \dots \mid a_n(\pi_1: \mathcal{T})?is_{a_n}$
 $\mid alt(\pi_1: \mathcal{T}, \pi_2: \mathcal{T})?is_{alt} \mid par(\pi_1: \mathcal{T}, \pi_2: \mathcal{T})?is_{par};$

Again, we assume a finite set of action labels $\mathcal{A}_{Trans} = \{a_0, \dots, a_n\}$ for the transition relation and a singleton set of action labels $\mathcal{A}_{Pred} = \{\downarrow\}$ for the predicate. Because we deal with a separate set of action labels for predicates as well as transitions, we need to substitute all occurrences of $\mathcal{A}$ by $\mathcal{A}_{Trans}$, for $R_{a_1}, \dots, R_{a_n}, R_{a1}, R_{a2}, R_{p1}, R_{p2}$ as they still reason about $\mathcal{A}$ as it were the only set of action labels available.

To compute the solution belonging to the termination predicate we introduce function the R_{Pred} , supported by three auxiliary functions R_{t1}, R_{t2}, R_{t3} . Valid solutions for predicates are computed by:

map $R_{Pred}, R_{t1}, R_{t2}, R_{t3}: \mathcal{T} \rightarrow Set(Solution)$
eqn $R_{Pred}(v) = R_{t1} \cup R_{t2} \cup R_{t3}: \mathcal{T} \rightarrow Set(Solution)$

where the auxiliary functions are defined as:

eqn $R_{t1}(v) = \{s: Solution \mid is_{one}(v) \wedge \pi_t(s) \approx v \wedge \pi_l(s) \approx \downarrow\};$
 $R_{t2}(v) = \{s: Solution \mid is_{alt}(v) \wedge \sigma^{y_1+x_2}(\pi_t(s))$
 $\quad \wedge sol(\downarrow, \mu_{y_1}^{y_1+x_2}(\pi_t(s))) \in R_{Pred}(\pi_1(v)) \wedge \mu_{x_2}^{y_1+x_2}(\pi_t(s)) \approx \pi_2(v)\};$
 $R_{t3}(v) = \{s: Solution \mid is_{alt}(v) \wedge \sigma^{x_1+y_2}(\pi_t(s))$
 $\quad \wedge sol(\downarrow, \mu_{y_2}^{x_1+y_2}(\pi_t(s))) \in R_{Pred}(\pi_2(v)) \wedge \mu_{x_1}^{x_1+y_2}(\pi_t(s)) \approx \pi_1(v)\};$

To illustrate the use of predicates within the framework, consider Figure 2 that shows a generated example with the mCRL2 tool-set. The initial specification p is shown in the caption. Here the process can either perform action a_0 and deadlock or perform action a_1 and terminate successfully. The tools used are identical to those used in our previous example.

Fig. 2: Example of a predicate, generated by the mCRL2 tool-set for $alt(a_0(zero), a_1(one))$

Remark 1. In Section 5 we discussed the use of a dummy action to resolve an implementation mismatch. Such dummy actions are now used to deviate between transition relations and predicates as well.

7 Related work

SOS meta-theory research is mainly aimed at proving useful properties about TSSs [2, 33] such as congruence results [23], deriving equational theories [1], conservative extensions [21], and soundness of axioms [3]. Research on how to implement them is underexposed. Most of the related work is performed with the Maude model checker [36]. Other authors have studied the link between the rewriting logic [26] and SOS both from a theoretical [10, 11, 28, 19, 33] as well as practical point of view [12, 13, 19, 32, 39, 40].

In [13], the outline of a translation from Modular SOS (MSOS) [30, 31] to the Maude rewriting logic is given and proven correct. The translation is straightforward and the technical twist is in the decomposition of labels, *e.g.*, to the structure of the labels in MSOS. A more elaborate explanation of this can be found in [11]. Within the work of [40], they try to capture the CCS semantics rewrites. While rewrites have no labels, labels are encoded as the result of a rewrite rule, *e.g.*, the CCS transition of $p \xrightarrow{a} q$ is written as $a.p \longrightarrow \{a\}p$. Though this is a correct transition, $(a.p) \parallel q \longrightarrow (\{a\}p) \parallel q$ is not, since the right-hand side term is not well formed. To overcome this problem, they introduce a dummy operator by which they extend the semantics in order to generate the transitive closure (p34-p38). Basically, rewrites can only be performed on the outermost function symbol and the result needs to be constructed as such. Since we use tuples to store a solution, rather than encoding it into a single term, we do not suffer from this drawback.

In the works of Mousavi and Reniers [32], Verdejo [38], and Verdejo and Marti-Oliet [39, 40], we see that the most noticeable difference is the formalism in which they express the TSS. In these works the authors stick to a representation for which hardly any tooling for formal analysis is available, or needs to be developed from scratch. This hinders a formal analysis. We have chosen a formalism, that is supported by a collection of tools that is specially aimed at performing formal analysis.

LETOS [25] is a tool environment that generates $\text{\LaTeX}$ documents and executable animations in Miranda [37]. This can be accomplished for a wide range of semantics, including some deterministic SOS forms. Since LETOS can only deal with deterministic semantics, it poses some problems when analysing the behavior of concurrent (non-deterministic) systems.

An approach for implementing SOS rules is presented in [15], which combines (unconditional) term-rewriting and λ -calculus for simulation. It demonstrates how SOS can be used in proof tools based on term rewriting. For that the Larch Prover [24] is used, and explained in [14]. Their method aims to demonstrate and prove the equivalence between different semantics definitions. We, however, aim at creating a bridge that closes the gap between a language for specification and a language for performing analysis. Furthermore, we include conditions and predicates, whereas they only allow predicates.

Process Algebra Compiler [16] is a tool that takes the signature and the SOS rules of a language and generates a LEX/YACC scanner/parser as well as verification libraries (Lisp and in Standard ML which are respectively compiled

with the kernels of the MAUTO tool [9] and the Concurrency Workbench [17]). In fact, PAC is a compiler that can be used as a front-end for verification tools. With the help of so-called back-end procedures, they generate the required routines for the different target systems, by relating concepts from the original language to those in the target formalism. How the relationship is defined between them, still needs to be addressed by the user. As our work describes such a relation, this method can be implemented into PAC.

8 Assessment and Future Work

In this paper we have demonstrated that a subclass of SOSs adhering to the De Simone rule format, can be transformed into a Linear Process Specification in the mCRL2 language. These can be subsequently accommodated with the mCRL2 tool-set. Although we have selected mCRL2 as our specification/implementation language, we do not foresee any difficulties when choosing another language as long as it has the same expressive power, *e.g.*, it facilitates a higher-order rewrite system to compute set comprehensions and a transition generator to (exhaustively) explore behavior.

The work presented here originates from work carried out as part of the KWR 09124 project LithoSysSL at ASML. The core activity within this project is to investigate how to formalize a Language-oriented, Domain Specific Modeling Environment and use it for specification, verification and validation purposes within the Lithography domain. During the project we have performed several successful analyses, based on the framework presented here. Ad-hoc concepts that have been incorporated into language are multiple signatures, complex predicates for transition synchronization and the enabling of transition via data. The framework has furthermore been used to study the composition of language elements, by which unintended behavior surfaced.

Extensions towards multi-sorted transition specifications with multiple transition relations are also planned to be added. For the future we are to accommodate extensions of the De Simone format rules and the scalability of the framework. We claim to be able to deal with negative premises of the form $x_i \xrightarrow{l}$, with copying of variables in the premises and in the target term of the conclusion. This means that we can deal with TSSs where the deduction rules have finitely many premises and are in the GSOS rule format [7]. Besides the extension to the GSOS format, we also have preliminary evidence that look-ahead in premises can be dealt with to some extent and that the use of state vectors in the TSS (as far as these are expressible in the mCRL2 data language) is not problematic either.

By incorporating these extensions, we are able to transform more languages towards the mCRL2 tool-set. The scalability of the framework concerns the size of the models (both terms of syntax as well as number of deduction rules), and performance by which it is possible to conduct analyses.

References

1. L. Aceto, B. Bloom, and F.W. Vaandrager. Turning SOS Rules into Equations. *Inf. Comput.*, 111(1):1–52, 1994.
2. L. Aceto, W. Fokkink, and C. Verhoef. Conservative Extension in Structural Operational Semantics. In *Current Trends in Theoretical Computer Science*, pages 504–524. 2001.
3. L. Aceto, A. Ingolfsdottir, M.R. Mousavi, and M.A. Reniers. Algebraic Properties for Free! *Bulletin of the EATCS*, 99:81–104, October 2009.
4. J.C.M. Baeten, T. Basten, and M.A. Reniers. *Process Algebra: Equational Theories of Communicating Processes (Cambridge Tracts in Theoretical Computer Science)*. Cambridge University Press, December 2009.
5. Jos C. M. Baeten and Chris Verhoef. A Congruence Theorem for Structured Operational Semantics with Predicates. In *CONCUR*, pages 477–492, 1993.
6. M. Bezem, R.N. Bol, and J.F. Groote. Formalizing Process Algebraic Verifications in the Calculus of Constructions. *Formal Asp. Comput.*, 9(1):1–48, 1997.
7. B. Bloom, S. Istrail, and A.R. Meyer. Bisimulation can’t be traced. *J. ACM*, 42(1):232–268, 1995.
8. R.N. Bol and J.F. Groote. The Meaning of Negative Premises in Transition System Specifications. *J. ACM*, 43(5):863–914, 1996.
9. G. Boudol, V. Roy, R. de Simone, and D. Vergamini. Process Calculi, from Theory to Practice: Verification Tools. In *Automatic Verification Methods for Finite State Systems*, pages 1–10, 1989.
10. C. Braga and J. Meseguer. Modular Rewriting Semantics in Practice. *ENTCS*, 117:393–416, 2005.
11. C. de O. Braga. *Rewriting Logic as a Semantic Framework for Modular Structural Operational Semantics*. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2001.
12. C. de O. Braga, E.H. Haeusler, J. Meseguer, and P.D. Mosses. Maude Action Tool: Using Reflection to Map Action Semantics to Rewriting Logic. In *AMAST*, pages 407–421, 2000.
13. C. de O. Braga, E.H. Haeusler, J. Meseguer, and P.D. Mosses. Mapping Modular SOS to Rewriting Logic. In *LOPSTR*, pages 262–277, 2002.
14. K-H. Buth. Using SOS Definitions in Term Rewriting Proofs. In *Larch, Workshops in Computing*, pages 36–54. Springer, 1992.
15. K-H. Buth. Simulation of SOS Definitions with Term Rewriting Systems. In *ESOP*, volume 788 of *LNCS*, pages 150–164. Springer, 1994.
16. R. Cleaveland, E. Madelaine, and S. Sims. A Front-End Generator for Verification Tools. In *TACAS*, volume 1019 of *LNCS*, pages 153–173. Springer, 1995.
17. R. Cleaveland and S. Sims. The NCSU Concurrency Workbench. In *CAV*, volume 1102 of *LNCS*, pages 394–397. Springer, 1996.
18. R. de Simone. Higher-Level Synchronising Devices in Meije-SCCS. *Theor. Comput. Sci.*, 37:245–267, 1985.
19. P. Degano, F. Gadducci, and C. Priami. A Causal Semantics for CCS via Rewriting Logic. *Theor. Comput. Sci.*, 275(1-2):259–282, 2002.
20. W. Fokkink. *Modelling Distributed Systems*. Springer Berlin Heidelberg, 2007.
21. W. Fokkink and C. Verhoef. A Conservative Look at Operational Semantics with Variable Binding. *Inf. Comput.*, 146(1):24–54, 1998.
22. J.F. Groote, A.J.H. Mathijssen, M.A. Reniers, Y.S. Usenko, and M.J. van Weerdenburg. The Formal Specification Language mCRL2. In *Methods for Modelling*

- Software Systems (MMOSS)*, number 06351 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.
23. J.F. Groote and F.W. Vaandrager. Structured Operational Semantics and Bisimulation as a Congruence. *Inf. Comput.*, 100(2):202–260, 1992.
 24. J.V. Guttag and J.J. Horning. *Larch: Languages and Tools for Formal Specification*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.
 25. P.H. Hartel. LETOS - a Lightweight Execution Tool for Operational Semantics. *Softw., Pract. Exper.*, 29(15):1379–1416, 1999.
 26. N. Martí-Oliet and J. Meseguer. Rewriting Logic as a Logical and Semantic Framework. *ENTCS*, 4, 1996.
 27. The mCRL2 toolset. Available from <http://www.mcrl2.org/>.
 28. J. Meseguer. Conditioned Rewriting Logic as a United Model of Concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.
 29. M. Monteban. Reduction Algorithms on Linear Process Equations. IS-TI 005, Vrije Universiteit Amsterdam, 2005.
 30. P.D. Mosses. Exploiting Labels in Structural Operational Semantics. In *SAC*, pages 1476–1481, 2004.
 31. P.D. Mosses. Modular Structural Operational Semantics. *J. Log. Algebr. Program.*, 60-61:195–228, 2004.
 32. M.R. Mousavi and M.A. Reniers. Prototyping SOS Meta-theory in Maude. *ENTCS*, 156(1):135–150, 2006.
 33. M.R. Mousavi, M.A. Reniers, and J.F. Groote. SOS Formats and Meta-theory: 20 Years After. *Theor. Comput. Sci.*, 373(3):238–272, 2007.
 34. G.D. Plotkin. A Structural Approach to Operational Semantics. *J. Log. Algebr. Program.*, 60-61:17–139, 2004.
 35. F.P.M. Stappers, M.A. Reniers, and S. Weber. Transforming SOS Specifications to Linear Processes. Computer Science Report No. 11-07, Eindhoven University of Technology, May 2011.
 36. The Maude system. Available from <http://maude.cs.uiuc.edu/>.
 37. D.A. Turner. Miranda: A Non-Strict Functional Language with Polymorphic Types. In *FPCA*, pages 1–16, 1985.
 38. A. Verdejo. Building Tools for LOTOS Symbolic Semantics in Maude. In *FORTE*, volume 2529 of *LNCS*, pages 292–307. Springer, 2002.
 39. A. Verdejo and N. Martí-Oliet. Implementing CCS in Maude 2. *ENTCS*, 71, 2002.
 40. A. Verdejo and N. Martí-Oliet. Executable Structural Operational Semantics in Maude. *J. Log. Algebr. Program.*, 67(1-2):226–293, 2006.

A Proofs

A.1 Labeled Transition System associated with an LPS

This subsection briefly describes the way in which a Labeled Transition System can be associated with an LPS [29].

Definition 2 (Labeled Transition System). *A labeled transition system (LTS) is a triple $(S, L, \rightarrow, s_0)$ where:*

- S is a set of states,
- L is a set of labels,
- $\rightarrow \subseteq S \times L \times S$ is a transition relation,
- $s_0 \in S$ is the initial state.

An LPS describes a transition relation by means of the following transition system specification. The signature of the transition system specification is left implicit. The only deduction rule is the following

$$\frac{\models c_i(d, e_i) = \text{true}}{X(d) \xrightarrow{a(d, e_i)} X(g_i(d, e_i))}$$

where $\models c_i(d, e_i)$ indicates that the Boolean expression $c_i(d, e_i)$ must be derivable equal to *true*.

The labeled transition system associated with closed term $X(p)$ with p a closed term of sort $\mathcal{T}$ is that part of the transition relation described by the LPS for X that is reachable from $X(p)$.

A.2 Labeled Transition System associated with a TSS

In [23] it is clearly defined how a transition relation is defined by means of a transition system specification. The labeled transition system associated with a closed term $p \in \mathcal{C}(\Sigma)$ is then obtained by considering that part of the transition relation described by the TSS that is reachable from p .

A.3 Lemmas

Lemma 1. *For all $x \in \mathcal{V}$, $t \in \mathcal{T}(\Sigma)$, and substitutions $\bar{\sigma}: \mathcal{V} \rightarrow \mathcal{T}(\Sigma)$*

$$x \in \text{vars}(t) \quad \Rightarrow \quad \mu_x^t(\bar{\sigma}(t)) = \bar{\sigma}(x)$$

Proof. By induction on the structure of term t .

- t is a variable. In case $t = x$ we have $\mu_x^t(\bar{\sigma}(t)) = \mu_x^x(\bar{\sigma}(t)) = \bar{\sigma}(x)$. The case where t is a variable different from x cannot occur as $x \notin \text{vars}(t)$.

- t is of the form $f(t_1, \dots, t_{ar(f)})$ for some $f \in \Sigma$ and $t_1, \dots, t_{ar(f)} \in \mathcal{T}(\Sigma)$. Since $x \in vars(t)$, we have $x \in vars(t_i)$ for some i such that $1 \leq i \leq ar(f)$. By induction hypothesis we obtain $\mu_x^{t_i}(\bar{\sigma}(t_i)) = \bar{\sigma}(x)$. Note that since $x \in vars(t_i)$ we have the equation $\mu_x^t(\bar{\sigma}(t)) = \mu_x^{t_i}(\pi_i(\bar{\sigma}(t)))$. Since $\pi_i(\bar{\sigma}(t)) = \pi_i(\bar{\sigma}(f(t_1, \dots, t_{ar(f)}))) = \pi_i(f(\bar{\sigma}(t_1), \dots, \bar{\sigma}(t_{ar(f)}))) = \bar{\sigma}(t_i)$, we then also have $\mu_x^t(\bar{\sigma}(t)) = \mu_x^{t_i}(\pi_i(\bar{\sigma}(t))) = \mu_x^{t_i}(\bar{\sigma}(t_i)) = \bar{\sigma}(t_i)$, which was to be shown.

Lemma 2. For all $t \in \mathcal{T}(\Sigma)$, and substitutions $\bar{\sigma}: \mathcal{V} \rightarrow \mathcal{T}(\Sigma)$

$$\sigma^t(\bar{\sigma}(t)) = true$$

Proof. By induction on the structure of term t .

1. t is a variable, say x . Then $\sigma^t(\bar{\sigma}(t)) = \sigma^x(\bar{\sigma}(x)) = true$.
2. t is of the form $f(t_1, \dots, t_{ar(f)})$ for some $f \in \Sigma$ and $t_1, \dots, t_{ar(f)} \in \mathcal{T}(\Sigma)$. By induction hypothesis we have $\sigma^{t_i}(\bar{\sigma}(t_i)) = true$ for all i such that $1 \leq i \leq ar(f)$.

$$\begin{aligned} \text{Then } \sigma^t(\bar{\sigma}(t)) &= is_f(f(t_1, \dots, t_{ar(f)})) \wedge \bigwedge_{i=1}^{ar(f)} \sigma^{t_i}(\pi_i(\bar{\sigma}(f(t_1, \dots, t_{ar(f)})))) = \\ &= true \wedge \bigwedge_{i=1}^{ar(f)} \sigma^{t_i}(\bar{\sigma}(t_i)) = true. \end{aligned}$$

Lemma 3. For all $t \in \mathcal{T}(\Sigma)$ and $p \in \mathcal{C}(\Sigma)$ such that $\sigma^t(p)$. If $\mu_x^t(p) = \bar{\sigma}(x)$ for all $x \in vars(t)$, then $\bar{\sigma}(t) = p$.

Proof. By induction on the structure of term t . Assume that $\mu_x^t(p) = \bar{\sigma}(x)$ for all $x \in vars(t)$.

1. t is a variable, say x . Since $\mu_x^t(p) = \bar{\sigma}(x)$ by assumption, and $\mu_x^t(p) = \mu_x^x(p) = p$ we have $\bar{\sigma}(t) = \bar{\sigma}(x) = \mu_x^t(p) = p$.
2. t is of the form $f(t_1, \dots, t_{ar(f)})$ for some $f \in \Sigma$ and $t_1, \dots, t_{ar(f)} \in \mathcal{T}(\Sigma)$. From $\sigma^t(p)$ it follows that $\sigma^{t_i}(\pi_i(p))$ for all $1 \leq i \leq ar(f)$. From the assumption that $\mu_x^t(p) = \bar{\sigma}(x)$ and the fact that $\mu_x^t(p) = \mu_x^{f(t_1, \dots, t_{ar(f)})}(p) = \mu_x^{t_i}(\pi_i(p))$ for those t_i in which x occurs it follows that $\mu_x^{t_i}(\pi_i(p)) = \bar{\sigma}(x)$ for all variables $x \in vars(t_i)$. Hence, by induction hypothesis we have $\bar{\sigma}(t_i) = \pi_i(p)$ for all $1 \leq i \leq ar(f)$. Then we have, $\bar{\sigma}(t) = \bar{\sigma}(f(t_1, \dots, t_{ar(f)})) = f(\bar{\sigma}(t_1), \dots, \bar{\sigma}(t_{ar(f)})) = f(\pi_1(p), \dots, \pi_{ar(f)}(p)) = p$. Note that we have used that $is_f(p)$ implies $f(\pi_1(p), \dots, \pi_{ar(f)}(p)) = p$ for all $p \in \mathcal{C}(\Sigma)$. This is easily proven by induction on the structure of p .

A.4 Proof of the main theorem

Theorem 1. Let $(\Sigma, \mathcal{D})$ be an mCRL2-restrictive TSS in the De Simone format. Then for every $p \in \mathcal{C}(\Sigma)$, the labeled transition system associated with p and the labeled transition system associated with $X(p)$ are isomorphic.

Proof. Obviously, it suffices to show that for all $p, p' \in \mathcal{C}(\Sigma)$ and $l \in \mathcal{A}$

$$p \xrightarrow{l} p' \quad \Rightarrow \quad \text{sol}(l, p') \in R(p) \quad (1)$$

and

$$\text{sol}(l, p') \in R(p) \quad \Rightarrow \quad p \xrightarrow{l} p' \quad (2)$$

since $\text{sol}(l, p') \in R(p)$ iff $X(p) \xrightarrow{l} X(p')$ follows directly from the semantics of an LPS.

First we give a proof for equation 1. We prove this part by induction on the depth of the proof tree of $p \xrightarrow{l} p'$. Now assume that the last step in this proof tree is the application of deduction rule $d \in \mathcal{D}$ of the form

$$\frac{\{x_i \xrightarrow{l_i} y_i \mid i \in I\}}{f(x_1, \dots, x_{\text{ar}(f)})} [\text{Cond}_d]$$

and let $\bar{\sigma}$ be a substitution such that $\bar{\sigma}(f(x_1, \dots, x_{\text{ar}(f)})) = p$, $\bar{\sigma}(t) = p'$, $\bar{\sigma}(x_i) \xrightarrow{l_i} \bar{\sigma}(y_i)$ for all $i \in I$, and Cond_d .

1. $\text{is}_f(p) = \text{is}_f(\bar{\sigma}(f(x_1, \dots, x_{\text{ar}(f)}))) = \text{is}_f(f(\bar{\sigma}(x_1), \dots, \bar{\sigma}(x_{\text{ar}(f)}))) = \text{true}$
2. $\sigma^t(p') = \sigma^t(\bar{\sigma}(t)) = \text{true}$. The last step is due to Lemma 2.
3. – Cond_d holds for the labels of the premises and the label for the conclusion.
 – For every $i \in I$ with $y_i \in \text{vars}(t)$: $\mu_{y_i}^t(p') = \mu_{y_i}^t(\bar{\sigma}(t)) = \bar{\sigma}(y_i)$ according to Lemma 1. Note that $\pi_i(p) = \pi_i(\bar{\sigma}(f(x_1, \dots, x_{\text{ar}(f)}))) = \pi_i(f(\bar{\sigma}(x_1), \dots, \bar{\sigma}(x_{\text{ar}(f)}))) = \bar{\sigma}(x_i)$. By induction hypothesis, for each $i \in I$, $\text{sol}(l_i, \bar{\sigma}(y_i)) \in R(\bar{\sigma}(x_i))$.
 Therefore,

$$\bigwedge_{i \in I} y_i \in \text{vars}(t) \quad \Rightarrow \quad \text{sol}(l_i, \mu_{y_i}^t(p')) \in R(\pi_i(p))$$

- For every $i \in I$ with $y_i \notin \text{vars}(t)$, by induction hypothesis, $\text{sol}(l_i, \bar{\sigma}(y_i)) \in R(\bar{\sigma}(x_i))$. As before $\bar{\sigma}(x_i) = \pi_i(p)$. Therefore,

$$\bigwedge_{i \in I} y_i \notin \text{vars}(t) \quad \Rightarrow \quad \exists_{z_i} \text{sol}(l_i, z_i) \in R(\pi_i(p))$$

4. For every j such that $1 \leq j \leq \text{ar}(f)$ and $j \notin I$ and $x_j \in \text{vars}(t)$ we have

$$\mu_{x_j}^t(p') = \mu_{x_j}^t(\bar{\sigma}(t)) = \bar{\sigma}(x_j) = \pi_j(p)$$

using Lemma 1. Therefore,

$$\bigwedge_{j \notin I} x_j \in \text{vars}(t) \quad \Rightarrow \quad \mu_{x_j}^t(p') = \pi_j(p)$$

From this we can conclude that $\text{sol}(l, p') \in R_d(p)$ and therefore also $\text{sol}(l, p') \in R(p)$.

Next we prove equation 2 by induction on closed term p . Assume that $X(p) \xrightarrow{l} X(p')$. Then this must be due to the fact that $\text{sol}(l, p') \in R(p)$. By definition this means that there exists a deduction rule $d \in \mathcal{D}$ such that $\text{sol}(l, p') \in R_d(p)$.

As d is a deduction rule in De Simone format, iff it is of the form

$$\frac{\{x_i \xrightarrow{l_i} y_i \mid i \in I\}}{f(x_1, \dots, x_{\text{ar}(f)})} [\text{Cond}_d]$$

From the definition of R_d it follows that there exist q_i , for $i \in I$ with $y_i \notin \text{vars}(t)$ such that $\text{sol}(l_i, q_i) \in R(\pi_i(p))$. Now, define a substitution $\bar{\sigma}$ such that

- $\bar{\sigma}(x_i) = \pi_i(p)$ for $1 \leq i \leq \text{ar}(f)$,
- $\bar{\sigma}(y_i) = \mu_{y_i}^t(p')$ for $i \in I$ such that $y_i \in \text{vars}(t)$,
- $\bar{\sigma}(y_i) = q_i$ for $i \in I$ such that $y_i \notin \text{vars}(t)$.

Now we can establish the following facts:

1. Cond_d holds for the labels of the premises and the label for the conclusion.
2. $\bar{\sigma}(f(x_1, \dots, x_{\text{ar}(f)})) = f(\bar{\sigma}(x_1), \dots, \bar{\sigma}(x_{\text{ar}(f)})) = f(\pi_1(p), \dots, \pi_{\text{ar}(f)}(p)) = p$. The last step follows from $\text{is}_f(p)$ (since $\text{sol}(l, p') \in R_d(p)$).
3. From $\text{sol}(l, p') \in R_d(p)$ it follows that $\mu_{x_j}^t(p') = \pi_j(p) = \bar{\sigma}(x_j)$ for all $1 \leq j \leq \text{ar}(f)$ such that $j \notin I$. By definition of $\bar{\sigma}$, for $i \in I$ and $y_i \in \text{vars}(t)$, also $\mu_{y_i}^t(p') = \bar{\sigma}(y_i)$. Therefore, as $\sigma^t(p)$ also follows from $\text{sol}(l, p') \in R_d(p)$, by Lemma 3 we have $\bar{\sigma}(t) = p'$.
4. From $\text{sol}(l, p') \in R_d(p)$ it follows that $\text{sol}(l_i, \mu_{y_i}^t(p')) \in R(\pi_i(p))$, for $i \in I$ and $y_i \in \text{vars}(t)$ and $\text{sol}(l_i, q_i) \in R(\pi_i(p))$, for $i \in I$ and $y_i \notin \text{vars}(t)$. By induction we then have $\pi_i(p) \xrightarrow{l_i} \mu_{y_i}^t(p')$, for $i \in I$ and $y_i \in \text{vars}(t)$ and $\pi_i \xrightarrow{l_i} q_i$, for $i \in I$ and $y_i \notin \text{vars}(t)$. Since $\pi_i(p) = \bar{\sigma}(x_i)$ and $\mu_{y_i}^t(p') = \bar{\sigma}(y_i)$, the premises of the deduction rule d are all derivable.

We can conclude that $p \xrightarrow{l} p'$. □

B Implementation

```
% This mCRL2 model describes the implementation of the Structural
% Operational Semantic deduction rules as described in:
%
% J.C.M. Baeten, T. Basten, and M.A. Reniers, M. A.
% Process Algebra: Equational Theories of Communicating Processes
% (Cambridge Tracts in Theoretical Computer Science)
%
% The corresponding deduction rules belonging the equations
% can be found on the pages mentioned above the equations.
```

```

% Set of Action/Predicate labels
sort AL = struct a0 | a1 | a2 | term;

% Term signature
sort T = struct zero?is_zero
          | one?is_one
          | a0( pi_1: T )?is_a0
          | a1( pi_1: T )?is_a1
          | a2( pi_1: T )?is_a2
          | alt( pi_1: T, pi_2: T )?is_alt
          | par( pi_1: T, pi_2: T )?is_par
          | seq( pi_1: T, pi_2: T )?is_seq
          ;

sort Solution = struct sol( pi_1: AL, pi_t: T );

map R, R_a0, R_a1, R_a2, R_alt_1, R_alt_2: T -> Set(Solution);
R_par_0, R_par_1, R_par_2 : T -> Set(Solution);
R_t1, R_t2, R_t3, R_seq_1, R_seq_2, R_seq_3 : T -> Set( Solution );
TR: T -> Set( Solution );
var p: T;
eqn R(p) = R_a0(p) + R_a1(p) + R_a2(p) + R_alt_1(p) + R_alt_2(p)
      + R_par_1(p) + R_par_2(p) + R_seq_2(p) + R_seq_3(p);

TR(p) = R_par_0(p) + R_t1(p) + R_t2(p) + R_t3(p) + R_seq_1(p);

map sigma_a0, sigma_a1, sigma_a2, sigma_alt_1: T -> Bool;
sigma_alt_2, sigma_par_1, sigma_par_2: T -> Bool;
Cond_a0, Cond_a1, Cond_a2, Cond_alt_1, Cond_alt_2: List(AL)#AL-> Bool;
Cond_par_1, Cond_par_2: List(AL)#AL-> Bool;
mu_a0_x0, mu_a1_x0, mu_a2_x0, mu_alt_1_x0, mu_alt_2_x0: T -> T;
mu_par_1_y0, mu_par_1_x1, mu_par_2_x0, mu_par_2_y1: T -> T;
var l: AL;
ls: List(AL);
p: T;

% Rule page 74.1
eqn Cond_a0( ls, l ) = l == a0;
sigma_a0(p) = true;
mu_a0_x0(p) = p;
R_a0(p)      = { t: Solution | is_a0(p)
                  && sigma_a0(pi_t(t))
                  && Cond_a0([], pi_l(t))
                  && mu_a0_x0(pi_t(t)) == pi_1(p)
                };

% Rule page 74.1
Cond_a1( ls, l ) = l == a1;
sigma_a1(p) = true;

```

```

mu_a1_x0(p) = p;
R_a1(p)      = { t: Solution | is_a1(p)
                  && sigma_a1(pi_t(t))
                  && Cond_a1([], pi_l(t))
                  && mu_a1_x0(pi_t(t)) == pi_1(p)
                };

% Rule page 74.1
Cond_a2( ls, l ) = l == a2;
sigma_a2(p) = true;
mu_a2_x0(p) = p;
R_a2(p)      = { t: Solution | is_a2(p)
                  && sigma_a2(pi_t(t))
                  && Cond_a2([], pi_l(t))
                  && mu_a2_x0(pi_t(t)) == pi_1(p)
                };

% Rule page 74.2
sigma_alt_1(p) = true;
mu_alt_1_x0(p) = p;
R_alt_1(p)     = { t: Solution | is_alt(p)
                  && sigma_alt_1(pi_t(t))
                  && sol( pi_l(t) , mu_alt_1_x0(pi_t(t))) in R(pi_1(p))
                };

% Rule page 74.3
sigma_alt_2(p) = true;
mu_alt_2_x0(p) = p;
R_alt_2(p)     = { t: Solution | is_alt(p)
                  && sigma_alt_2(pi_t(t))
                  && sol( pi_l(t) , mu_alt_2_x0(pi_t(t))) in R(pi_2(p))
                };

% Rule page 84.1
R_t1(p)        = { t: Solution | is_one(p) && pi_l(t) == term && (pi_t(t)) == p };

% Rule page 84.2
R_t2(p)        = { t: Solution | is_alt(p)
                  && p == pi_t(t)
                  && term == pi_l(t)
                  && sol( term, pi_1(p) ) in TR( pi_1( p ) ) };

% Rule page 84.3
R_t3(p)        = { t: Solution | is_alt(p)
                  && p == pi_t(t)
                  && term == pi_l(t)
                  && sol( term, pi_2(p) ) in TR( pi_2( p ) ) };

% Rule page 175.1
R_seq_1(p)     = { t: Solution | is_seq(p)
                  && sol(pi_l(t), pi_1(pi_t(t))) in R(pi_1(p))
                };

```

```

                                && sol(pi_1(t), pi_2(pi_t(t))) in R(pi_2(p))
                                && pi_1(t) == term
                                && is_seq(pi_t(t))
};

% Rule page 175.2
R_seq_2(p) = { t: Solution | is_seq(p)
                                && sol(pi_1(t), pi_1(pi_t(t))) in R(pi_1(p))
                                && pi_2(pi_t(t)) == pi_2(p)
                                && is_seq(pi_t(t))
};

% Rule page 175.3
R_seq_3(p) = { t: Solution | is_seq(p)
                                && sol(term, pi_1(p)) in TR(pi_1(p))
                                && t in R(pi_2(p))
};

% Rule page 216.1
R_par_0(p) = { t: Solution | is_par(p)
                                && p == pi_t(t)
                                && sol(term, pi_1(p)) in TR(pi_1(p))
                                && sol(term, pi_2(p)) in TR(pi_2(p))
                                && pi_1(t) == term
};

% Rule page 216.3
sigma_par_1(p) = is_par(p);
mu_par_1_y0(p) = pi_1(p);
mu_par_1_x1(p) = pi_2(p);
R_par_1(p) = { t: Solution | is_par(p)
                                && sigma_par_1(pi_t(t))
                                && sol( pi_1(t) , mu_par_1_y0(pi_t(t))) in R(pi_1(p))
                                && mu_par_1_x1(pi_t(t)) == pi_2(p)
};

% Rule page 216.4
sigma_par_2(p) = is_par(p) ;
mu_par_2_x0(p) = pi_1(p);
mu_par_2_y1(p) = pi_2(p);
R_par_2(p) = { t: Solution | is_par(p)
                                && sigma_par_2(pi_t(t))
                                && mu_par_2_x0(pi_t(t)) == pi_1(p)
                                && sol( pi_1(t) , mu_par_2_y1(pi_t(t))) in R(pi_2(p))
};

act tr: AL;
pr: AL;

```



```

proc X(p: T) = sum s: Solution. (s in R( p ) ) -> tr( pi_l(s) ) . X( pi_t( s ))
      + sum s: Solution. (s in TR( p ) ) -> pr( pi_l(s) ) . X( pi_t( s )) ;

% Models
%
% Figure 1a
%init X( a0( a1( a2( zero ) ) ) );

% Figure 1b
%init X( par( a1 ( zero )) , a2( zero ) ) );

% Figure 1c
%init X( alt( a0 ( zero ) , a1( a2 ( zero ) ) ) );

% Figure 2
%init X( alt( a0(zero) , a1( one )));

init X( seq( a0 ( a1 (one) ) , a2( one ) ) );

```