Journal of Logical and Algebraic Methods in Programming 85 (2016) 200–226

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

# Product line process theory

Fatemeh Ghassemi [a,*], Mohammad Reza Mousavi [b,1]

[a] School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran
[b] Centre for Research on Embedded Systems (CERES), School of IT, Halmstad University, Sweden

ARTICLE INFO

*Article history:*
Received 4 August 2014
Received in revised form 3 August 2015
Accepted 17 September 2015
Available online 4 November 2015

*Keywords:*
Software product line
Process theory
Product line bisimulation
Strict strong bisimulation
$\mu$-Calculus
Axiomatization

ABSTRACT

Software product lines (SPLs) facilitate reuse and customization in software development by genuinely addressing the concept of variability. Product Line Calculus of Communicating Systems (PL-CCS) is a process calculus for behavioral modeling of SPLs, in which variability can be explicitly modeled by a binary variant operator. In this paper, we study different notions of behavioral equivalence for PL-CCS, based on Park and Milner's strong bisimilarity. These notions enable reasoning about the behavior of SPLs at different levels of abstraction. We study the compositionality property of these notions and the mutual relationship among them. We further show how the strengths of these notions can be consolidated in an equational reasoning method. Finally, we designate the notions of behavioral equivalence that are characterized by the property specification language for PL-CCS, called multi-valued modal $\mu$-calculus.


## 1. Introduction

*Software product line (SPL)* engineering has become an established trend in software development, where a family of similar software products with minor differences are developed in tandem, instead of developing each specific software product separately [1]. SPL engineering benefits from systematic reuse throughout the system life cycle and enables mass development and customization of numerous products. Hence, the development cost and the time to market for an SPL is substantially decreased, compared to the cumulative development cost and time of the isolated products [2]. To this aim, various software engineering activities have to be adapted to cope with the differences among the artifacts for different products, called *variability*. Variability introduces a new complexity dimension and hence, this calls for a genuine treatment of variability in different artifacts (such as requirement specification, architectural design, detailed design, and implementation artifacts). Such a treatment should also allow for a collective analysis of product line behavior (e.g., in testing and verification [3,4]) to deal with the inherent complexity of SPLs.

At the highest level of abstraction, an SPL can be specified by a set of features that satisfy the specific needs of a particular market segment or mission [5]. A feature identifies "a prominent or distinctive unit of requirement which can be either a user-visible behavior, aspect, quality, or characteristic of a software system" [6]. Hence, a product can be specified by a subset of features. To specify an SPL, the features are organized in a hierarchical model, called a *feature model*.

* Corresponding author.
*E-mail address:* fghassemi@ut.ac.ir (F. Ghassemi).
[1] The work of M.R. Mousavi has been partially supported by the Swedish Research Council (Vetenskapsrådet) award number: 621-2014-5057 (Effective Model-Based Testing of Concurrent Systems) and the Swedish Knowledge Foundation (Stiftelsen för Kunskaps- och Kompetensutveckling) in the context of the AUTO-CAAS HöG project (number: 20140312).

http://dx.doi.org/10.1016/j.jlamp.2015.09.008
2352-2208/

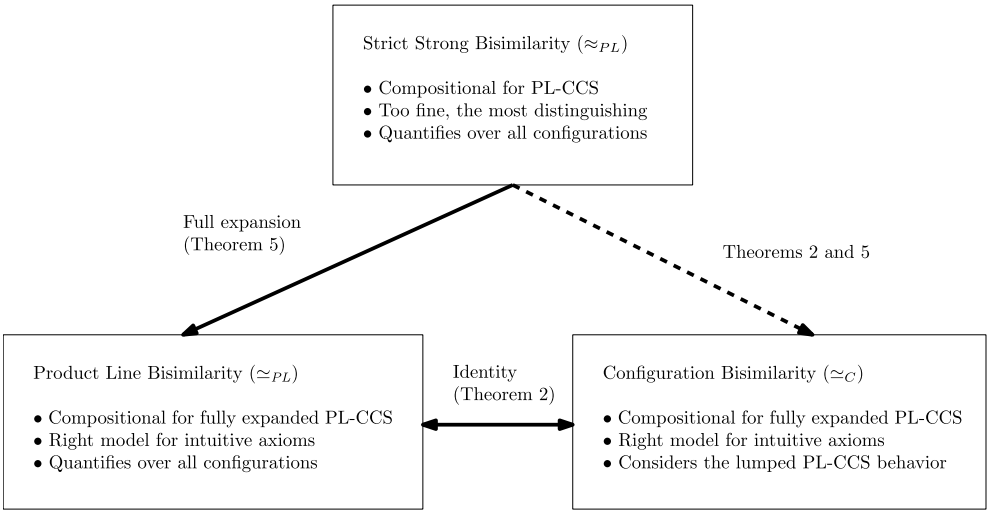


**Fig. 1.** Our notions of behavioral equivalence for PL-CCS specifications.

It identifies commonalities and differences among the products of the SPL in terms of their features and it identifies suitable relations among features, such as optional, mandatory, or mutually exclusive. More concrete specifications capture structural and behavioral aspects of an SPL. For instance, the architecture description languages Koala [7] and xADL [8] concentrate on structural modeling of SPLs. Modal transition systems (MTSs) [9] and Featured transition systems (FTSs) [10], however, concentrate on behavioral modeling (we refer to [11] for an overview of such behavioral models). MTSs capture the behavior of SPLs by defining state transitions as optional or mandatory, while FTSs annotate transitions with a set of features. Behavioral models typically come equipped with a product derivation method; e.g., a product, derived from a feature model, can project an FTS into a labeled transition system (LTS).

Formal verification techniques provide strong tools to analyze complex systems to guarantee their correctness. Process algebra is a formal approach to describe the behavior of communicating concurrent systems in a compositional manner. Product Line Calculus of Communicating Systems (PL-CCS) [12,13] is an extension of Milner's Calculus of Communicating Systems (CCS) [14]. PL-CCS extends CCS by adding the binary variant operator $\oplus_i$ to model behavioral variability in SPLs. More specifically, process term $p_1 \oplus_1 p_2$, where $p_1$ and $p_2$ are CCS process terms, specifies a family of two alternative products, namely $p_1$ or $p_2$ (the index $i$ in $\oplus_i$ is used to designate repeated choices that have to be made in the same way; when no repetition of indices is present, the indices can be safely ignored). The semantics of a PL-CCS specification is given in terms of three different models: the *flat semantics*, the *unfolded semantics*, and the *configured-transition semantics* [12]. A PL-CCS specification can be turned into a product, specified by a CCS term, by resolving the variability points, i.e., the variant operators, by deciding on whether their right or left process is chosen. The flat semantics of a PL-CCS term is given in terms of the semantics of all derivable products, denoted by CCS terms. A product family LTS (PF-LTS) is an extension of an LTS, where labels and states are paired with configuration vectors that maintain the configuration of variants. PF-LTSs provide the unfolded semantics of PL-CCS terms and are derived through a set of structural rules in a systematic way. The structural rules given in [12] work on a restricted set of PL-CCS terms in order to be compositional. The configured-transition semantics is defined over the unfolded semantics by merging all states that only differ in their configuration parts. This provides the most succinct model of PL-CCS terms. Hence, in the developments to come, we mostly focus on the configured-transition semantics of PL-CCS. In particular, we provide a set of structural rules that derive a configured-transition semantics for PL-CCS terms directly.

Equational reasoning is the cornerstone of the algebraic approach to process theory. To furnish PL-CCS with a proper equational theory, we study a number of notions of behavioral equivalence, based on strong bisimilarity [15]. A summary of these notions, their properties and the results establishing their relationship is depicted in Fig. 1. We start with a set of axioms that we expect to be sound for a model of PL-CCS and define the notion of strict strong bisimilarity, which is a natural extension of strong bisimilarity in the SPL setting. Namely, strict strong bisimilarity requires bisimilar product lines to behave bisimilarly for all common configurations. This turns out to be a fully compositional notion for PL-CCS, but too strong of a notion for some of our intuitive axioms. For example, strict strong bisimilarity rejects $p \oplus_i q = q \oplus_i p$, which is an intuitive axiom. Subsequently, we introduce a strictly coarser notion, called product line bisimilarity, which does satisfy the axioms we defined for PL-CCS. However, this notion is shown to satisfy a weaker compositionality property. Namely, it is compositional for a subset of PL-CCS terms, called fully expanded terms. To remedy the latter issue, we show that all PL-CCS term can be rewritten into this subset using a sound transformation, thanks to the strong compositional notion of strict strong bisimilarity. Since strict strong bisimilarity implies product line bisimilarity this transformation is also sound for the latter notion and hence, resolves its compositionality issue. Finally, we introduce configuration bisimilarity, which is an alternative yet equivalent notion for product line bisimilarity. The main motivation for introducing configuration bisimilarity

is that it allows for reasoning about the product line behavior as a whole and hence, dispenses with scrutinizing individual products' behaviors.

Our axiomatization is useful to identify common parts (i.e., the mandatory parts) among the products of a family, to reorganize the functionality of a family specification to behaviors for which appropriate components exist, to derive products of a family to validate a model in terms of its intended systems (where each product is specified by a CCS term), and to manipulate functionalities assigned to products of a product line at the syntactic level.

Regarding an equational theory for PL-CCS, our work improves upon [13], where a number of algebraic laws to restructure a product line specification were given. However, we are not aware of any complete axiomatization of PL-CCS to date. For example, the laws of [13] are restricted to families with the same number of variants and exclude intuitive equalities like $p \oplus_i q = q \oplus_i p$ and $p \oplus_i p = p$. It is worth noting that our notion of configuration bisimilarity is reminiscent of the notion of branching bisimilarity introduced in [16].

PL-CCS also comes equipped with a property specification language that is a variant of the multi-valued modal $\mu$-calculus [12]. A configured transition system can also be viewed as a multi-valued modal Kripke structure [12] and hence, formulae in the multi-valued modal $\mu$-calculus can be evaluated naturally in this semantic domain. The corresponding model checking method verifies a family at once, and its result defines the set of products that meet the given property. In this paper, we show that the multi-valued modal $\mu$-calculus of [12] is the logical characterization of our notion of product line bisimilarity (and hence, also our notion of configuration bisimilarity). This provides another evidence of suitability for our main notions of behavioral equivalence.

The contributions of this paper can be summarized as follows:

- We introduce a means to specify product lines with infinite behavior, following the approach of [17], by extending PL-CCS with recursive specifications.
- We provide a set of structural rules to derive the configured-transition semantics of PL-CCS directly.
- We study different notions of bisimilarity over the configured-transition semantics. To this end, we provide a set of intuitive axioms that should be satisfied. We prove different compositionality (congruence) results for the notions of bisimilarity and relate them to each other.
- We provide a sound axiomatization of PL-CCS terms modulo product line bisimilarity, which additionally allows one to derive any sound equation on closed terms with finite-state behavior (in technical terms, it is a ground-complete axiomatization).
- We show that the multi-valued modal $\mu$-calculus is the characterizing logic for our product line bisimilarity.

The rest of this paper is structured as follows. Section 2 introduces the PL-CCS syntax and semantics. Section 3 defines our notions of behavioral equivalence and relates them. Section 4 provides a set of sound and complete axioms to syntactically manipulate product lines. Section 5 illustrates the applicability of our axiomatization in the analysis of product lines. Section 6 presents the multi-valued modal $\mu$-calculus as well as a logical characterization of product line bisimilarity. Section 7 provides an overview of existing approaches on modeling and verification of SPLs. Finally, Section 8 concludes the paper.

## 2. Product line CCS

To our knowledge, PL-CCS [12,13] is the first process algebra introduced to formally specify and verify product lines in an algebraic manner. To give a semantics to PL-CCS terms in [12,13], the binary variant operators are assigned an index in a pre-processing step. Following the same principle, we index the binary variant operator of PL-CCS with a natural number. This allows for defining a unique semantic model for each PL-CCS term and also specify multiple variation points that should be resolved in the same manner. Moreover, any unnumbered PL-CCS term can be considered an indexed PL-CCS term by assigning arbitrary distinct natural numbers to each and every binary variant operator.

To specify product lines with infinite behavior, we extend PL-CCS with the recursion operator $\langle X|E\rangle$ taken from [17]. It encompasses both the CCS recursion operator $recX.t$ (which is specified in our syntax as $\langle X|X \stackrel{def}{=} t\rangle$) and the standard way to express recursion in ACP (where usually only guarded recursion is considered via systems of equations $E$) [17].

### 2.1. PL-CCS: syntax

Let $\mathcal{A}$ be the set of process names which are used as recursion variables in recursive specifications and ranged over by $A$ and $B$. Moreover assume that $\Sigma$ a finite set of input action labels, $\overline{\Sigma} = \{\overline{a} \mid a \in \Sigma\}$ is the set of output actions, and $\tau \notin \Sigma \cup \overline{\Sigma}$ the unobservable action. Then, the set of all actions $Act$ is defined as $\Sigma \cup \overline{\Sigma} \cup \{\tau\}$. By definition, we have that $\overline{\overline{a}} = a$.

The core syntax of PL-CCS comprises deadlock 0, action prefix $a.-$ (for each $a \in Act$), choice $+$, binary variant $\oplus_i$ where $i \in \mathbb{N}$, and parallel composition $\|$. It also includes process renaming $[f]$ where $f : Act \mapsto Act$ is a renaming function with $f(\overline{a}) = \overline{f(a)}$, and $f(\tau) = \tau$, restriction $\backslash L$ where $L \subseteq Act$. Additionally, it has process names, and recursion operator $\langle A|E\rangle$ where $E$ is a *recursive specification* over $\mathcal{A}$, denoted by $E(\mathcal{A})$ for short. A recursive specification is defined by a set of recursive equations that contains precisely one recursive equation $A \stackrel{def}{=} t_A$ for each process name $A \in \mathcal{A}$, where $t_A$ is a term over the PL-CCS signature and process names from $\mathcal{A}$. The PL-CCS syntax is summarized by the following grammar:

$$t ::= \ 0 \mid a.t \mid t + t \mid t \oplus_i t \mid t \parallel t \mid t[f] \mid t \setminus L \mid A \mid \langle A|E\rangle$$

Process term $a.t$ denotes a process that first performs action $a$ and then behaves as $t$. Alternative composition $t_1 + t_2$ nondeterministically behaves as $t_1$ or $t_2$. Variant operator $\oplus_i$ defines a behavioral variation point. Family $t_1 \oplus_i t_2$ consists of the two alternative families specified by $t_1$ and $t_2$. Binary variant operators $\oplus_i$, or *variants* for short, with identical indices $i$ are resolved in the same manner. PL-CCS terms can be composed using the parallel composition operator $\parallel$. Process term $t_1 \parallel t_2$ denotes the concurrent execution of two processes $t_1$ and $t_2$, of which the actions can be interleaved or synchronized whenever $t_1$ and $t_2$ are ready to execute an input and the corresponding output action simultaneously. The process term $t[f]$ behaves as $t$, with every action renamed according to the renaming function $f$. The process term $t \setminus L$ can perform any action that is not included in $L$. A process name $A$ denotes a specific process, and the recursion operator $\langle A|E\rangle$ represents a solution of the recursive specification $E(\mathcal{A})$ where $A$ acts as the initial variable. A solution of a recursive specification $E(\mathcal{A})$ is a set of process terms $\{s_A \mid A \in \mathcal{A}\}$ such that if for all $A \in \mathcal{A}$, $s_A$ is substituted for $A$, the equations of $E$ correspond to equal elements (in the model of our equational theory), i.e., $s_A = t\{s_X/X|X \in \mathcal{A}\}$, where $A \stackrel{def}{=} t \in E$. The guardedness criterion for recursive specifications ensures that this solution is unique. As far as unguarded recursions are concerned, following the approach of CCS and ACP [18], we consider the solution that has the least set of transitions. In Section 4, we explain the guardedness criterion. In the remainder of this paper, we use the notions of process term, product line, and family interchangeably.

Note that the term defining process name $A$ in a recursive specification may include recursive specifications. A term is called *closed*, if every process name $A$ occurs in the scope of a binding recursive specification $E(\mathcal{A})$ such that $A \in \mathcal{A}$. For instance, in the closed term $\langle X|\{X \stackrel{def}{=} a.0 \oplus_1 b.\langle Y|\{Y \stackrel{def}{=} Y + c.X\}\rangle\}\rangle$, $X$ is bound by the outer recursive specification. As usual, we use the notation $t\{s/A\}$ to denote the substitution of a closed term $s$ for every free occurrence of process name $A$ in $t$. We use $\langle t|E\rangle$, where $E$ is a recursive specification over $\mathcal{A}$, to denote $t\{\langle A|E\rangle/A \mid A \in \mathcal{A}\}$, i.e., $t$ where, for all $A \in \mathcal{A}$, all free occurrences of $A$ in $t$ are replaced by $\langle A|E\rangle$.

By adopting $\langle A|E\rangle$ instead of the CCS recursion operator, we can easily specify SPLs in which a process name definition is shared. For instance, $\langle p_1|\{p_1 \stackrel{def}{=} p_2 \parallel p_2,\ p_2 \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle$ is equivalent to the CCS notation $\mu X.(\mu Y.b.0 \oplus_1 c.0 \parallel \mu Y.b.0 \oplus_1 c.0)$.

Index $i$ is called *bounded* in $t_1 \oplus_i t_2$. The bounded indices of term $t$, denoted by $bi(t)$, are those that are reachable from the root of its parse tree. We define $bi(t) = fbi(t, \emptyset)$, where the auxiliary function *fbi* is defined inductively as follows:

$$\begin{array}{ll}
fbi(0, \mathcal{S}) = \emptyset & fbi(t \setminus L, \mathcal{S}) = fbi(t, \mathcal{S}) \\
fbi(a.t, \mathcal{S}) = fbi(t, \mathcal{S}) & fbi(t[f], \mathcal{S}) = fbi(t, \mathcal{S}) \\
fbi(t_1 + t_2) = fbi(t_1, \mathcal{S}) \cup fbi(t_2, \mathcal{S}) & fbi(t_1 \parallel t_2, \mathcal{S}) = fbi(t_1, \mathcal{S}) \cup fbi(t_2, \mathcal{S}) \\
fbi(t_1 \oplus_i t_2, \mathcal{S}) = \{i\} \cup fbi(t_1, \mathcal{S}) \cup fbi(t_2, \mathcal{S}) & fbi(A, \mathcal{S}) = \emptyset \\
fbi(\langle A|E \cup \{A \stackrel{def}{=} t\}\rangle, \mathcal{S}) = fbi(\langle t|E \cup \{A \stackrel{def}{=} t\}\rangle, \mathcal{S} \cup \{A\}), \text{ if } A \notin \mathcal{S} & \\
fbi(\langle A|E \cup \{A \stackrel{def}{=} t\}\rangle, \mathcal{S}) = \emptyset, \text{ if } A \in \mathcal{S}. &
\end{array}$$

An index $i$ is *free* in $t$, iff it is not bounded. We denote by $t[i/j]$ the term that is obtained by replacing all $\oplus_j$ by $\oplus_i$ in $t$ (we dispense with the inductive definition as it is straightforward).

### 2.2. PL-CCS semantics

Intuitively, the behavior of a product line family is defined by the cumulative behavior of its products. These products are obtained by resolving the choice in the binary variant operators. The resolution may take place at various points of execution and hence, to record such choices the semantics needs to record whether the choice is unresolved (denoted by ?), resolved in favor of the left-hand-side product (denoted by $L$), or resolved in favor of the left-hand-side product (denoted by $R$). Also resolving one instance of a binary variant operator may resolve the choice for other instances. For instance, configuring the variant $j$ as $R$ in $(a.0 \oplus_i b.0) \oplus_j c.0$, makes it unnecessary to configure the variant $i$. Configuration status of variation points bounded in a process term with maximum index $n$ are recorded in a configuration vector $\nu \in \{L, R, ?\}^n$, where the $i$th element of the vector is denoted by $\nu|_i$. We denote by *Config* the set of all possible configuration vectors, ranged over by $\nu$ and $\lambda$. Expression $\nu|_{i/x}$ denotes the result of replacing the $i$th element of $\nu$ by $x \in \{L, R\}$. A configuration is called *full* when all its elements are configured, i.e., are in $\{L, R\}$. Two configurations $\nu$ and $\lambda$ that do not have any conflict on a variation point are called *consistent*. This concept is formalized below.

**Definition 1** *(Consistent configuration vectors [12]).* Configuration vectors $\nu, \lambda \in \{L, R, ?\}^n$ are *consistent*, denoted by $\nu \asymp \lambda$, if and only if $\forall i \in \{1, \ldots, n\} : ((\nu|_i = ?) \vee (\lambda|_i = ?) \vee (\nu|_i = \lambda|_i))$.

Given two consistent configuration vectors $\nu$ and $\lambda$, their *unification*, denoted by $\nu \odot \lambda$ merges the configurations of their variation points as follows: $(\nu \odot \lambda)|_i = X \in \{L, R, ?\}$ iff either $\nu|_i = X \wedge \lambda|_i = ?$ or $\nu|_i = ? \wedge \lambda|_i = X$ or $\nu|_i = \lambda|_i = X$.

Configuration vector $\nu'$ is *more concrete* than $\nu$ (or $\nu$ is *more abstract* than $\nu'$), denoted by $\nu \sqsubseteq \nu'$, iff $\forall i \in \{1, \ldots, n\} : ((\nu|_i = ?) \vee (\nu|_i = \nu'|_i))$ [12]. Hence, each configuration vector $\nu$ represents a set of configuration vectors $\{\nu' \mid \nu \sqsubseteq \nu'\}$.

$$\frac{}{a.t \xrightarrow{a,\nu_?} t}\ :Prefix \qquad \frac{t_1 \xrightarrow{a,\nu} t_1'}{t_1 + t_2 \xrightarrow{a,\nu} t_1'}\ :Choice$$

$$\frac{t \xrightarrow{a,\nu} t' \qquad a \notin L}{t \setminus L \xrightarrow{a,\nu} t' \setminus L}\ :Res \qquad \frac{t_1 \xrightarrow{a,\nu} t_1' \qquad \nu|_i \neq R}{t_1 \oplus_i t_2 \xrightarrow{a,\nu|_{i/L}} t_1'}\ :Select$$

$$\frac{t_1 \xrightarrow{a,\nu} t_1'}{t_1 \parallel t_2 \xrightarrow{a,\nu} t_1' \parallel t_2}\ :Par \qquad \frac{\langle t|E\rangle \xrightarrow{a,\nu} t' \qquad A \stackrel{def}{=} t \in E}{\langle A|E\rangle \xrightarrow{a,\nu} t'}\ :Call$$

$$\frac{t_1 \xrightarrow{a,\nu} t_1' \qquad t_2 \xrightarrow{\overline{a},\nu'} t_2' \qquad \nu \asymp \nu'}{t_1 \parallel t_2 \xrightarrow{\tau,\nu\odot\nu'} t_1' \parallel t_2'}\ :Sync \qquad \frac{t \xrightarrow{a,\nu} t'}{t[f] \xrightarrow{f(a),\nu} t'[f]}\ :Rename$$

**Fig. 2.** Operational semantics to derive configured-transition system.

We briefly explained the three different semantic models of PL-CCS terms in Section 1: the *flat semantics*, the *unfolded semantics*, and the *configured-transition semantics* [12]. Since our equivalence relation and the multi-valued modal $\mu$-calculus are defined over the configured-transition semantics, we next elaborate on how this semantics is derived directly using our structural operational semantics rules.

The configured-transition semantics induces an LTS, in which the labels are pairs in $Act \times Config$. Formally, a configured-transition system (CTS) is a tuple $\langle S, s_0, Act \times Config, \rightarrow\rangle$, where $S$ is a set of states, $s_0 \in S$ is an initial state and $\rightarrow \subseteq S \times Act \times Config \times S$ is a set of transition relations. The notation $s \xrightarrow{\alpha,\nu} s'$ is used for $(s, (\alpha, \nu), s') \in \rightarrow$ and is representative for all transitions $s \xrightarrow{\alpha,\nu'} s'$, where $\nu \sqsubseteq \nu'$. The CTS semantics of a PL-CCS term $t$ has the set of all terms as its states, $t$ as the initial states, and the least relation satisfying the rules in Fig. 2 as its transition relation.

The rule *Prefix* indicates the execution of a prefix action, where $\nu_?$ denotes a configuration vector in which no element is configured. *Choice* specifies the non-deterministic behavior of the choice operator in terms of its operands. *Res* defines that term $t \setminus L$ is only allowed to do actions that are not in $L$. *Select* defines the behavior of a family in terms of its products, by deciding about the $i$th variant operator. The side condition prohibits any reconfiguration, if it was previously configured. *Call* defines the behavior of $\langle A|E\rangle$ in terms of the behavior of the right-hand side of the equation $A \stackrel{def}{=} t$ in the recursive specification $E$. *Par* explains that a process in a parallel composition can proceed independently of the other parallel component. *Sync* states that two processes in a parallel composition can be synchronized on an action, if both are ready to perform input and output counterparts simultaneously. Since $t_1$ and $t_2$ resolve variants in their scopes, their resolutions are unified for their parallel composition in *Sync*. Finally, *Rename* renames all actions using a function $f$.

The symmetric versions of rules *Choice*, *Select*, and *Par* are also present, but are not given explicitly here for the sake of brevity.

**Example 2.** Using the rules in Fig. 2, the configured-transition semantics of $\langle X|E\rangle$, where $E = \{X \stackrel{def}{=} (a.(b.X \oplus_1 c.0) + d.0) \oplus_2 e.0\}$, is given in Fig. 3a. The derivation tree inducing the transition labeled by $a, \langle ?, L\rangle$ is given below:

$$\cfrac{\cfrac{\cfrac{\cfrac{}{a.(b.\langle X|E\rangle \oplus_1 c.0) \xrightarrow{a,\langle ?,?\rangle} b.\langle X|E\rangle \oplus_1 c.0}\ :Prefix}{a.(b.\langle X|E\rangle \oplus_1 c.0) + d.0 \xrightarrow{a,\langle ?,?\rangle} b.\langle X|E\rangle \oplus_1 c.0}\ :Choice}{(a.(b.\langle X|E\rangle \oplus_1 c.0) + d.0) \oplus_2 e.0 \xrightarrow{a,\langle ?,L\rangle} b.\langle X|E\rangle \oplus_1 c.0}\ :Select}{\langle X|E\rangle \xrightarrow{a,\langle ?,L\rangle} b.\langle X|E\rangle \oplus_1 c.0}\ :Call$$

Other transitions are derived similarly. On deriving the transitions of $b.\langle X|E\rangle \oplus_1 c.0$ with the help of *Prefix* and *Select*, only the first variant point can be configured, and consequently it returns to state $\langle X|E\rangle$ with the action $b, \langle L, ?\rangle$ or state 0 with the action $c, \langle R, ?\rangle$, as shown in Fig. 3a.

Returning to state $\langle X|E\rangle$ in Example 2, makes it possible to reconfigure any previously configured variant. For the sake of compositionality, resolutions of variants are open to any possible configuration in our SOS rules. However, as it is explained in the next paragraph and Section 3.1, in deriving the behavior of a product/a set of related products only consistent resolutions are followed.

The semantic model of each product of $t$, identified by the full configuration $\nu^f$, can be derived by removing the transitions from $t$ whose configuration vector is not consistent with $\nu^f$. Let $\Pi(t, \nu^f)$ denote the resulting LTS. Formally speaking, $\Pi(t, \nu^f) \xrightarrow{a} \Pi(t', \nu^f)$ iff $t \xrightarrow{a,\nu} t'$ and $\nu \sqsubseteq \nu^f$. Therefore, only resolutions that are consistent with the full configuration, i.e., $\nu \sqsubseteq \nu^f$, are allowed and consequently reconfiguration is prohibited. See Fig. 3b, 3c, 3d, and 3e for the semantic models of

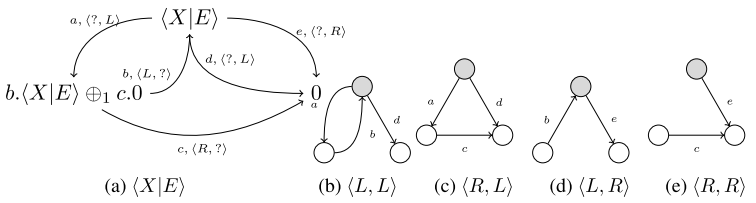


**Fig. 3.** The configured-transition system of $\langle X|E\rangle$, and $\Pi(\langle X|E\rangle, \nu)$ for the given configuration vectors.

**Table 1**
The axioms that product line bisimilarity should support.

| | | | |
|---|---|---|---|
| $p \oplus_i q = q \oplus_i p,\ \ i \notin bi(p) \cup bi(q)$ | $A_1$ | $(p \oplus_i q) + r = (p + r) \oplus_i (q + r)$ | $A_5$ |
| $(p \oplus_i q) \oplus_j r = p \oplus_i (q \oplus_j r), i \notin bi(r) \wedge j \notin bi(p)$ | $A_2$ | $r + (p \oplus_i q) = (r + p) \oplus_i (r + q)$ | $A_6$ |
| $p \oplus_i p = p$ | $A_3$ | $r \parallel (p \oplus_i q) = (r \parallel p) \oplus_i (r \parallel q)$ | $D_1$ |
| $a.(p \oplus_i q) = a.p \oplus_i a.q$ | $A_4$ | $(p \oplus_i q) \parallel r = (p \parallel r) \oplus_i (q \parallel r)$ | $D_2$ |

products derived from $X$ for the given configuration vectors $\langle L, L\rangle$, $\langle R, L\rangle$, $\langle L, R\rangle$, and $\langle R, R\rangle$ respectively (initial states are highlighted in gray). It should be noted that the semantic models derived for the configuration vectors $\langle L, R\rangle$ and $\langle R, R\rangle$ have only one reachable state from the initial state through the action $e$.

## 3. Bisimilarity for product lines

Following the approach of ACP [18], we define a set of axioms (as the main part of our *process theory* or *equational theory*) as primary and then investigate the models that they have. The most intuitive model of a process theory is the term algebra (the algebra with the same operators of equational theory) modulo a congruence. In this section, we first discuss about different notions of equivalence relation to reason about product lines. Next, we discuss about the congruence property of the previously defined relations.

Table 1 summarizes the axioms we have in mind for PL-CCS. We look for an appropriate notion of bisimilarity that supports the given equations. Axioms $A_{1-3}$ define commutativity, associativity and idempotency for the binary variant operator. Axioms $A_{1,2}$ ensure that two families are equivalent when they produce the same set of products, irrespective of their orders in variant operators. However, their application is restricted: $i$ should be free in $p$ and $q$ for $A_1$, while $i$ should be free in $r$ and $j$ should be free in $p$ for $A_2$. For instance, $a.0 \oplus_1 (b.0 \oplus_1 c.0)$ produces two products $a.0$ and $c.0$, but $(b.0 \oplus_1 c.0) \oplus_1 a.0$ produces $a.0$ and $b.0$, and consequently, as expected they are not equivalent. Axiom $A_3$ removes a repeated product from a family, and implies that two product families are equivalent iff they produce similar products, irrespective of their multiplicity. Axiom $A_4$ defines distributivity for prefix over binary variant, while axioms $A_{5,6}$ define distributivity for choice over binary variant. These rules allow for postponing the product selection by factorizing the common initial action/behavior respectively. Axioms $D_{1,2}$ define distributivity for parallel over binary variant. These two axioms reveal the difference between alternative choice and binary variant. Axioms $A_{5,6}$ and $D_{1,2}$ are useful to reduce redundancy by factorizing common parts.

### 3.1. Equivalence relation

Strong bisimulation [15] is very efficient to check and affords a neat theory: many other notions in the branching spectrum can be reduced to it by adding a standard set of axioms [19]. An LTS over a set of labels $L$ is defined by $\langle S, s_0, L, \rightarrow\rangle$, where $S$ is a set of states, $s_0 \in S$ is an initial state, and $\rightarrow \subseteq S \times L \times S$ is a set of transitions. The notation $s \xrightarrow{a} t$ is used for $(s, a, t) \in \rightarrow$. We typically identify LTSs with their initial states. Intuitively two labeled transition systems are equivalent, if they produce the same set of actions (observable behavior) and have the same branching structure:

**Definition 3** *(Strong bisimulation).* Two LTSs $s$ and $t$ are *strongly bisimilar*, notation $s \sim t$, iff there is a strong bisimulation relation $\mathcal{R}$ over their states such that:

- $(s, t) \in \mathcal{R}$, and
- for each pair $(s', t') \in \mathcal{R}$:
  - $\forall s'' \cdot s' \xrightarrow{\alpha} s'' \Rightarrow \exists t'' \cdot t' \xrightarrow{\alpha} t''$ and $(s'', t'') \in \mathcal{R}$, and
  - $\forall t'' \cdot t' \xrightarrow{\alpha} t'' \Rightarrow \exists s'' \cdot s' \xrightarrow{\alpha} s''$ and $(s'', t'') \in \mathcal{R}$.

Definition 3 can be readily used for CTSs (since CTSs can be considered LTSs with a structure on the set of labels $L$). However, this simple adoption of Definition 3 can lead to some counter-intuitive observations.

For example, according to this definition, $a.0 + b.0$ is strongly bisimilar to $a.0 \oplus_1 b.0$. However, these two processes should not be considered equivalent intuitively. The family $a.0 + b.0$ produces one product which has a non-deterministic

behavior in performing actions $a$ and $b$. By contrast, the family $a.0 \oplus_1 b.0$ produces two products, namely $a.0$ and $b.0$, and each product has deterministic behavior by performing solely $a$ or solely $b$. As another concern, this relation cannot identify $a.(b.0 \oplus_1 c.0)$ and $a.b.0 \oplus_1 a.c.0$, while both have two products $a.b.0$ and $a.c.0$; consequently, this bisimulation relation does not support axiom $A_4$. Therefore, the appropriate notion of bisimilarity over families must relate any product in one family to a product in the other such that their behaviors are strongly bisimilar.

A full configuration is called *valid* with respect to term $t$, if its length is not less than the maximum index in $bi(t)$. For instance, $\langle L\rangle$ is not valid for $b.\langle X|E\rangle \oplus_1 c.0$, where $E = \{(a.(b.X \oplus_1 c.0) + d.0) \oplus_2 e.0\}$, while $\langle L, ?, L\rangle$ is valid. Let $VFConfig(t)$ denote the set of all valid full configurations with respect to $t$. Intuitively, two product families are equivalent when they produce bisimilar sets of products:

**Definition 4** *(Strict strong bisimulation).* Two product line terms $s$ and $t$ are *strictly strongly bisimilar*, denoted by $s \approx_{PL} t$, iff for any valid full configuration $\nu^f \in VFConfig(s) \cap VFConfig(t)$, $\Pi(s, \nu^f) \sim \Pi(t, \nu^f)$.

However, strict strong bisimilarity does not support axioms $A_{1,2}$; to see this, observe that $\Pi(a.0 \oplus_1 b.0, \langle L\rangle) \nsim \Pi(b.0 \oplus_1 a.0, \langle L\rangle)$, $\Pi((a.0 \oplus_1 b.0) \oplus_2 c.0, \langle L, R\rangle) \nsim \Pi(a.0 \oplus_1 (b.0 \oplus_2 c.0), \langle L, R\rangle)$ and $\Pi((a.0 \oplus_2 b.0) \oplus_1 c.0, \langle R, L\rangle) \nsim \Pi(a.0 \oplus_2 (b.0 \oplus_1 c.0), \langle R, L\rangle)$. However, axiom $A_3$ is supported by strict strong bisimilarity, e.g., $(a.0 \oplus_1 b.0) \oplus_2 (a.0 \oplus_1 b.0) \approx_{PL} a.0 \oplus_1 b.0$ since for any $\nu^f \in \{\langle L, R\rangle, \langle L, L\rangle, \langle R, R\rangle, \langle R, L\rangle\}$, $\Pi((a.0 \oplus_1 b.0) \oplus_2 (a.0 \oplus_1 b.0), \nu^f) \sim \Pi(a.0 \oplus_1 b.0, \nu^f)$.

To make Definition 4 be insensitive to the placement of families in a binary variant composition, and consequently to support axioms $A_{1,2}$, the notion of bisimulation can be revised as follows.

**Definition 5** *(Product line bisimulation).* Two product line terms $s$ and $t$ are *product line bisimilar*, denoted by $s \simeq_{PL} t$, if and only if:

- $\forall \nu_1^f \in VFConfig(s) \cdot \exists \nu_2^f \in VFConfig(t) \cdot \Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$, and
- $\forall \nu_2^f \in VFConfig(t) \cdot \exists \nu_1^f \in VFConfig(s) \cdot \Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$.

**Theorem 6.** *Product line bisimilarity is an equivalence relation.*

See Appendix A for the proof. Confining the behaviors of two CTSs into full configurations disallows any reconfiguration of variants whose resolutions are left open in the semantic models (as it was explained in Section 2.2).

Matching each and every valid full configuration in the product line bisimilarity is very tedious. This process may require examining all possible pairs of configurations to find a suitable match. (Yet it does eventually reduce to checking strong bisimilarity for a large number of finite-state behaviors, which is decidable [20].) Hence, it would be appealing to have an appropriate notion of bisimilarity that decides the equivalence of PL-CCS terms at once (without considering all full configurations individually). We provide such a relation, called configuration bisimulation next and prove that it coincides with product line bisimilarity. Therefore, all the results for product line bisimilarity also hold for configuration bisimilarity. Also, to facilitate reasoning about product lines and also establish an algebraic theory for product line processes, we provide a sound and complete axiomatization for product line and configuration bisimilarity. Using this axiomatization, a term can be restructured at the syntactic level to its equivalent terms without the need to generate and compare the state spaces. In the process of providing a sound and complete axiomatization, we use product line bisimilarity which makes our proofs much simpler.

First, we define the notion of configuration inspired by [16]. As a first step, we would like to classify the transitions in terms of their configuration vectors; all transitions with consistent configurations are called a *consistent transition set* and they can potentially belong to a family. Subsequently, we define a bisimulation relation that relates states in terms of their consistent transition sets. In other words, it is required to relate not only states, but also their consistent transition sets. Consider Fig. 4, which illustrates this by an example. The terms in this figure are not product line bisimilar, as the left one consists of four (behaviorally different) products while the right one consists of two. However, a notion of bisimulation that only considers consistent transition sets in each state (as shown in Fig. 4) cannot distinguish them. The consistent transition set $\{(d, \langle L, ?\rangle), (a, \langle ?, ?\rangle)\}$ can be enabled together for a product/family. For instance, consider the product $d.0 + a.c.0$, which is identified by the full configuration $\langle L, R\rangle$ in the left CTS. It is derived from the consistent transition sets highlighted in Fig. 4. These consistent transition sets are matched to the corresponding sets in the right CTS. However, the matched sets do not belong to a family, and consequently cannot specify a product. The reason stems from the related states $b.0 \oplus_2 c.0$ and $b.0$. State $b.0 \oplus_2 c.0$ is reachable by the consistent transition set $\{(d, \langle L, ?\rangle), (a, \langle ?, ?\rangle)\}$ for two products (i.e., $\{\langle L, R\rangle, \langle L, L\rangle\}$). Therefore, its consistent transition set $\{(c, \langle ?, R\rangle)\}$ is enabled for product $\langle L, R\rangle$. However, its related state $b.0$ is only reachable for one product, which does not generate a matching consistent transition set. To summarize, we need to note for which family a state is reachable in order to match its enabled consistent transition sets with respect to that family. Furthermore, this book-keeping family disallows any reconfiguration of variants which have been previously resolved.

A *partitioning* of a configuration vector $\nu$ consists of configuration vectors $\nu_1, \cdots, \nu_n$ such that $\forall i, j \leq n \cdot ((i \neq j) \Rightarrow \nu_i \not\asymp \nu_j)$, and $\forall \nu^f \cdot (\nu \sqsubseteq \nu^f \Rightarrow \exists j \leq n \cdot (\nu_j \sqsubseteq \nu^f))$. For instance, $\{\langle ?, L\rangle, \langle L, R\rangle, \langle R, R\rangle\}$ is a partitioning of $\langle ?, ?\rangle$. A par-

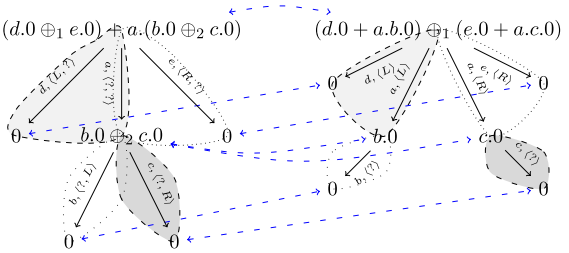


**Fig. 4.** An example on defining a bisimulation regarding states and consistent transition sets.

titioning of $\nu$, denoted by $p_\nu$, regards configurations over transitions of $s$, denoted by $Par(s, \nu, p_\nu)$, if and only if $\forall \nu_1 \in p_\nu \Rightarrow \exists \alpha, \nu_1', s'(s \xrightarrow{\alpha,\nu_1'} s' \wedge \nu_1' \sqsubseteq \nu_1) \vee \nexists \alpha, \nu_1', s'(s \xrightarrow{\alpha,\nu_1'} s' \wedge \nu_1' \asymp \nu_1)$. Transitions of $s$ whose configurations are more abstract than $\nu \in p_{\nu_1}$, where $Par(s, \nu_1, p_{\nu_1})$, constitute a consistent transition set. Assume $s$ and $t$ are related for families $\nu_1$ and $\nu_2$, respectively. In the below-defined notion of configuration bisimulation, we match consistent transition sets of $s$ defined by $p_{\nu_1}$, where $Par(s, \nu_1, p_{\nu_1})$, to consistent transition sets of $t$ defined by $p_{\nu_2}$, where $Par(t, \nu_2, p_{\nu_2})$.

**Definition 7** *(Configuration bisimulation).* A class of binary relations $R_{\nu_1,\nu_2} \subseteq S \times S$, where $\nu_1, \nu_2 \in Config$, is a configuration simulation relation if and only if for each $s, t \in S$ such that $(s, t) \in R_{\nu_1,\nu_2}$, there exists $p_{\nu_1}$ and $p_{\nu_2}$, where $Par(s, \nu_1, p_{\nu_1})$ and $Par(t, \nu_2, p_{\nu_2})$ such that for any $\nu_1' \in p_{\nu_1}$, there exists $\nu_2' \in p_{\nu_2}$ and:

- $s \xrightarrow{\alpha,\nu_1''} s'$ and $\nu_1'' \sqsubseteq \nu_1' \Rightarrow \exists t',\ \nu_2'' \cdot t \xrightarrow{\alpha,\nu_2''} t',\ \nu_2'' \sqsubseteq \nu_2'$, and $(s', t') \in R_{\nu_1',\nu_2'}$;
- $t \xrightarrow{\alpha,\nu_2''} t'$ and $\nu_2'' \sqsubseteq \nu_2' \Rightarrow \exists s',\ \nu_1'' \cdot s \xrightarrow{\alpha,\nu_1''} s',\ \nu_1'' \sqsubseteq \nu_1'$, and $(s', t') \in R_{\nu_1',\nu_2'}$.

$R_{\nu_1,\nu_2}$ is a configuration bisimulation if $R_{\nu_1,\nu_2}$ and $R^{-1}_{\nu_1,\nu_2}$ are configuration simulations. Two states $s, t \in S$ are called configuration bisimilar, denoted by $s \simeq_C t$, if and only if $(s, t) \in R_{\nu_?,\nu_?}$ for some configuration bisimulation relation $R_{\nu_?,\nu_?}$.

For instance, $a.(b.0 \oplus_1 c.0) \simeq_C a.c.0 \oplus_1 a.b.0$ is witnessed by the configuration bisimulation relations $R_{\langle ? \rangle,\langle ? \rangle} = \{(a.(b.0 \oplus_1 c.0), a.c.0 \oplus_1 a.b.0)\}$, $R_{\langle R \rangle,\langle L \rangle} = \{(b.0 \oplus_1 c.0, c.0), (0, 0)\}$, $R_{\langle L \rangle,\langle R \rangle} = \{(b.0 \oplus_1 c.0, b.0), (0, 0)\}$. However $a.(b.0 \oplus_1 0) \not\simeq_C a.b.0$ as the only partitioning of $\langle ? \rangle$ regarding the state transitions of $b.0 \oplus_1 0$ is $\{\langle R \rangle, \langle L \rangle\}$ and then the behavior $b.0 \oplus_1 0$ for family $\langle R \rangle$ cannot be matched to any behavior of $a.b.0$.

**Theorem 8.** *For any PL-CCS $s$ and $t$, $s \simeq_{PL} t \Leftrightarrow s \simeq_C t$.*

**Proof.** We assume $s \simeq_{PL} t$, we show that $s \simeq_C t$. For all valid full configurations $\nu_1^f \in VFConfig(s)$ and $\nu_2^f \in VFConfig(t)$, construct $R_{\nu_1^f,\nu_2^f}$ as:

$$R_{\nu_1^f,\nu_2^f} = \{(s', t') \mid \Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f) \text{ witnessed by } R' \wedge (\Pi(s', \nu_1^f), \Pi(t', \nu_2^f)) \in R'\}$$

It is easy to check that $R$ is a configuration bisimulation relation.

We assume $s \simeq_C t$ is witnessed by the class of configuration bisimulation relations $R_{\nu_1,\nu_2}$, where $\nu_1, \nu_2 \in Config$, we show that $s \simeq_{PL} t$. To this aim, for any $\nu_1^f \in VFConfig(s)$, we find $\nu_2^f \in VFConfig(t)$ such that $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$ and vice versa. Let $\mathcal{B} = \{\nu_2 \mid (s', t') \in R_{\nu_1,\nu_2} \wedge \nu_1 \sqsubseteq \nu_1^f\}$. Choose $b \in \mathcal{B}$ such that $\nexists \nu \in \mathcal{B} \setminus \{b\} \cdot b \sqsubseteq \nu$. Take a configuration $\nu_2^f$ such that $b \sqsubseteq \nu_2^f$. It is trivial that $R' = \{(\Pi(s', \nu_1^f), \Pi(t', \nu_2^f)) \mid (s', t') \in R_{\nu_1,\nu_2} \wedge \nu_1 \sqsubseteq \nu_1^f \wedge \nu_2 \sqsubseteq \nu_2^f\}$ is a strong bisimulation witnessing $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$. □

Partitioning each family and then matching its consistent transition sets is very complex and may need to examine all possible partitionings to find a suitable match. Since product line and configuration bisimilarity coincide, with the aim to provide a sound and complete axiomatization, we use product line bisimilarity which makes our proofs much simpler. Therefore, all results for product line bisimilarity also hold for configuration bisimilarity.

### *3.2. Congruence property*

In this section, we study the congruence property of strict and product line bisimulation relations, and provide a syntactic restriction over PL-CCS terms that makes product line bisimilarity a congruence with respect to the PL-CCS operators.

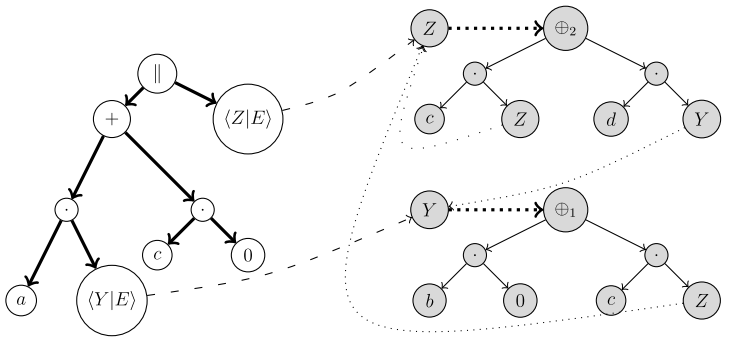


**Fig. 5.** The term-dependency graphs of $(a.\langle Y|E\rangle + e.0) \parallel \langle Z|E\rangle$, where $E = \{Y \stackrel{def}{=} b.0 \oplus_1 c.Z, Z \stackrel{def}{=} c.Z \oplus_2 d.Y\}$.

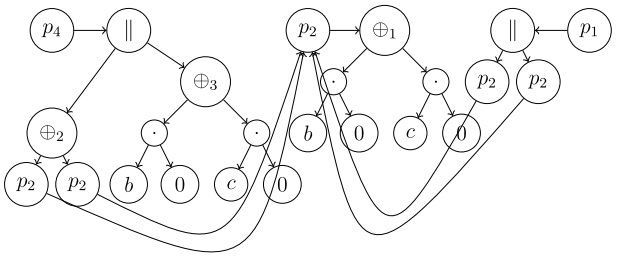


**Fig. 6.** The term-dependency graphs of $p_1$ and $p_4$; the term-dependency graph of $p_2$ is shared.

Strict strong bisimilarity is a congruence for PL-CCS terms. For instance, $\langle p_2'|\{p_2' \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle \approx_{PL} b.0 \oplus_1 c.0$, induces that $\langle p_2'|\{p_2' \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle \parallel \langle p_2'|\{p_2' \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle \approx_{PL} (b.0 \oplus_1 c.0) \parallel (b.0 \oplus_1 c.0)$.

**Theorem 9.** *Strict strong bisimilarity is an equivalence and a congruence for the PL-CCS term algebra.*

See Appendix A for the proof.

The case for product line bisimulation is a bit more intricate. To illustrate the involved issues, observe that $(d.0 \oplus_1 e.0) \parallel (b.0 \oplus_1 c.0) \not\simeq_{PL} (d.0 \oplus_1 e.0) \parallel (c.0 \oplus_1 b.0)$, while $b.0 \oplus_1 c.0 \simeq_{PL} c.0 \oplus_1 b.0$. The reason is that the configurations $\langle R\rangle$ and $\langle L\rangle$ of $b.0 \oplus_1 c.0$ are matched to the configurations $\langle L\rangle$ and $\langle R\rangle$ of $c.0 \oplus_1 b.0$ respectively, but each pair of matched configurations chooses a different product in $d.0 \oplus_1 e.0$. However, $(d.0 \oplus_1 e.0) \oplus_2 (b.0 \oplus_1 c.0) \simeq_{PL} (d.0 \oplus_1 e.0) \oplus_2 (c.0 \oplus_1 b.0)$, since $\oplus_2$ makes the configurations of its operands independent of each other. To guarantee congruence for product line bisimilarity, we impose a constraint on the PL-CCS syntax. In $(d.0 \oplus_1 e.0) \parallel (b.0 \oplus_1 c.0)$, there are two binary variants indexed by 1. Hence, once one resolves the choice between $d.0$ and $e.0$, the same choice has to made between $b.0$ and $c.0$. We call a product line *fully expanded* when all its variants can be configured independently from the configuration of other variation points.

This constraint was first introduced in [12] with the different intention of compositionality for their structural rules. We revise their definition to enforce that different binary variant choices can be made independent of each other. To formally define a fully expanded term, we use its *term-dependency graph* which is a directed labeled graph. Its construction for a term $t$ is explained on the term $(a.\langle Y|E\rangle + e.0) \parallel \langle Z|E\rangle$, where $E = \{Y \stackrel{def}{=} b.0 \oplus_1 c.Z, Z \stackrel{def}{=} c.Z \oplus_2 d.Y\}$. Its nodes comprise the nodes of the parse tree of $t$ together with additional nodes labeled $\langle A_i|E_i\rangle$ and $E_i$ (i.e., the gray and the white nodes in Fig. 5, respectively). Its edges comprise the edges of the parse tree of $t$ (i.e., the thick solid edges in Fig. 5) plus edges connecting $\langle A_i|E_i\rangle$ to the node labeled $A_i$ in the term-dependency graph of $E_i$ (i.e., the dashed edges in Fig. 5), and the edges of recursive specifications $E_i$. The term-dependency graph of $E_i(\mathcal{A}_i)$ consists of nodes labeled $A_i$ for each $A_i \in \mathcal{A}_i$, together with the nodes of the parse trees for term $t_i$ for each $A_i \stackrel{def}{=} t_i \in E_i$. Its edges comprise the edges of the parse trees (i.e., the thin solid edges in Fig. 5) plus the edges connecting $A_i$ to the roots of the parse trees of the corresponding right-hand sides, i.e., $t_i$ (i.e., the thick dashed edges in Fig. 5). Additionally, we add edges from leaves of the parse trees labeled $A_i$ to the node labeled with $A_i$ in its binding recursive specification (i.e., the dotted edges in Fig. 5).

**Definition 10** *(Fully expanded terms).* A PL-CCS term is *fully expanded* if and only if in its term-dependency graph, for each two distinct simple paths starting at an arbitrary common node and ending at (common or distinct) nodes labeled with $\oplus_i$, they have both passed through a common node that is labeled with $\oplus_j$, for some $j \neq i$.

$$\frac{t_1 \xrightarrow{a,\nu} t_1'}{t_1 \between t_2 \xrightarrow{a,\nu} t_1' \parallel t_2} : LMerge \qquad \frac{t_1 \xrightarrow{a,\nu} t_1' \quad t_2 \xrightarrow{\bar{a},\nu'} t_2' \quad \nu \asymp \nu'}{t_1 \mid t_2 \xrightarrow{\tau,\nu\odot\nu'} t_1 \parallel t_2} : Merge$$

$$\frac{t \xrightarrow{a,\nu} t' \quad \nu|_i \neq R}{L(t,i) \xrightarrow{a,\nu} L(t',i)} : LSelect \qquad \frac{t \xrightarrow{a,\nu} t' \quad \nu|_i \neq L}{R(t,i) \xrightarrow{a,\nu} R(t',i)} : RSelect$$

**Fig. 7.** Operational semantic rules for auxiliary operators.

Recall that a simple path is a path in a graph which does not have repeating vertices. This constraint rules out systems such as $\langle p_1 | \{p_1 \stackrel{def}{=} p_2 \parallel p_2,\ p_2 \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle$, since the parallel composition in the equation of $p_1$ has two simple paths to $\oplus_1$. However, these paths do not pass through a node labeled with $\oplus_j$, $j \neq 1$, beforehand. Nevertheless, it accepts systems such as $\langle p_4 | \{p_4 \stackrel{def}{=} (p_2 \oplus_2 p_2) \parallel (b.0 \oplus_3 c.0),\ p_2 \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle$, since all simple paths of the parallel composition in the equation $p_4$ to $\oplus_1$ have already passed through $\oplus_2$, as shown in Fig. 6. The same holds for simple paths of the node labeled with $p_4$. Such systems were not accepted by the Definition given in [12]. The term-dependency graph of Fig. 5 satisfies the condition of Definition 10.

**Theorem 11.** *Product line bisimilarity is a congruence on the fully expanded PL-CCS term algebra.*

See Appendix A for the proof. Restricting to fully expanded PL-CCS terms is not important in practice (when terms are manipulated at the syntactic level), since a term can be rewritten using our axioms supported by strict strong bisimilarity into a fully expanded form (see Theorem 12 in Section 4.2). Note that the side condition of axiom $A_2$ guarantees that a fully expanded term remains fully expanded after being restructured by this axiom. For instance, although $(a.0 \oplus_1 b.0) \oplus_2 (c.0 \oplus_1 d.0)$ is fully expanded, $a.0 \oplus_1 (b.0 \oplus_2 (c.0 \oplus_1 d.0))$ is not.

## 4. Equational reasoning on PL-CCS terms

We extend the axioms given in Section 3 to reason about parallel, recursive behaviors with finite-state models, and indexed binary variants. To axiomatize the interleaving behavior of parallel composition and terms with variants with an identical index, we extend the PL-CCS syntax and semantics with new operators in Section 4.1.

We provide PL-CCS axioms that are sound with respect to product line bisimilarity in Section 4.2. Furthermore, we identify those that are also valid with respect to strict strong bisimilarity. Our axiomatization in Table 2 subsumes standard axioms of CCS for choice operator ($C_{1-4}$, $R_{1-4}$, $E_{1-4}$), axioms of ACP corresponding to the well-known elimination theorem [21] ($P_{1-3,5,6}$ and $S_{1,2,4-6}$), and axioms of CCS to reason about recursive behaviors [22] (*Fold*, *UnFold*, and *Ung*). We explain that a term can be manipulated using axioms supported by strict strong bisimilarity (without changing the order of operands in binary variants) to be rewritten into a fully expanded form, and then manipulated with all axioms valid for product line bisimilarity. We prove ground-completeness (completeness over closed terms) of our axiomatization for a subset of PL-CCS terms, namely, those with finite-state behaviors in Section 4.3.

### 4.1. Extending PL-CCS framework

Our axiomatization borrows from the process algebra *ACP* [23] two auxiliary operators (left merge and communication merge) to axiomatize the interleaving and the synchronizing behavior of parallel composition, respectively. Furthermore, we extend the process theory with two new sets of indexed operators (left and right selector), to restrict the behavior of a term regarding configuration of the variant indexed by that number.

In the left merge composition $t_1 \between t_2$, the left operand ($t_1$) performs an action and then continues in parallel with $t_2$. In the communication merge $t_1 \mid t_2$, both operands are synchronized on their initial actions and then continue in parallel composition. The left selector operator $L(t,i)$ makes all variants indexed by $i$ be configured as left. Therefore, it only allows behaviors whose configurations on the variant indexed by $i$ are consistent with left. The right selector operator $R(t,i)$ behaves symmetrically. The structural operational semantics rules of operators to derive CTSs are given in Fig. 7.

### 4.2. PL-CCS axiomatization

We proceed to complete the axiomatization of PL-CCS modulo product line bisimilarity. The axioms are given in Table 2. Axioms $C_{1-4}$ define commutativity, associativity, idempotence, and unit element for the choice operator.

Axiom $P_1$ defines the parallel composition of two families in an interleaving semantics, as in the process algebra *ACP* [23]; Axiom $P_2$ explains the behavior of the left merge operator in terms of its left operand, if it can do an action. However, if it cannot do any action, the result is a deadlock, as explained by $P_6$. Axioms $P_{3,4}$ define left-distributivity of choice and binary variant over the left merge operator, respectively. Axiom $P_5$ defines right-distributivity of binary variant over the left merge operator. Axiom $S_1$ defines the commutativity property for the communication merge operator. Axioms $S_{2,3}$ (together with $S_1$) define distributivity of choice and binary variant over the communication merge operator respectively. Axioms

**Table 2**
The axiomatization of PL-CCS terms.

| | | | |
|---|---|---|---|
| $p+q=q+p$ | $C_1$ | $(p+q)+r=p+(q+r)$ | $C_2$ |
| $p=p+p$ | $C_3$ | $0+p=p$ | $C_4$ |
| $p \parallel q=(p \between q)+(q \between p)+(p \mid q)$ | $P_1$ | $p \mid q=q \mid p$ | $S_1$ |
| $a.p \between q=a.(p \parallel q)$ | $P_2$ | $(p+q) \mid r=(p \mid r)+(q \mid r)$ | $S_2$ |
| $(p+q) \between r=(p \between r)+(q \between r)$ | $P_3$ | $(p \oplus_i q) \mid r=(p \mid r) \oplus_i (q \mid r)$ | $S_3$ |
| $(p \oplus_i q) \between r=(p \between r) \oplus_i (q \between r)$ | $P_4$ | $(a.p) \mid (\overline{a}.q)=\tau.(p \parallel q)$ | $S_4$ |
| $p \between (q \oplus_i r)=(p \between q) \oplus_i (p \between r)$ | $P_5$ | $(a.p) \mid (b.q)=0,\ (b \neq \overline{a}) \vee (a=\tau)$ | $S_5$ |
| $0 \between p=0$ | $P_6$ | $0 \mid p=0$ | $S_6$ |
| $(a.p)[f]=f(a).(p[f])$ | $R_1$ | $(a.p) \setminus L=a.(p \setminus L),\ a \notin L$ | $E_1$ |
| $(p+q)[f]=(p[f])+(q[f])$ | $R_2$ | $(a.p) \setminus L=0,\ a \in L$ | $E_2$ |
| $(p \oplus_i q)[f]=(p[f]) \oplus_i (q[f])$ | $R_3$ | $(p+q) \setminus L=(p \setminus L)+q \setminus L)$ | $E_3$ |
| $0[f]=0$ | $R_4$ | $0 \setminus L=0$ | $E_4$ |
| | | $(p \oplus_i q) \setminus L=(p \setminus L) \oplus_i (q \setminus L)$ | $E_5$ |
| $L(p \oplus_i q, i)=L(p,i)$ | $N_1$ | $L(p \oplus_j q, i)=L(p,i) \oplus_j L(q,i),\ i \neq j$ | $N_2$ |
| $R(p \oplus_i q, i)=R(q,i)$ | $N_3$ | $R(p \oplus_j q, i)=R(p,i) \oplus_j R(q,i),\ i \neq j$ | $N_4$ |
| $p \oplus_i q=L(p,i) \oplus_i R(q,i)$ | $N_5$ | $p=p[k/j],\ k \notin bi(p)$ | $N_6$ |
| $\langle A \mid E \cup \{B \stackrel{def}{=} t\} \rangle=\langle \langle A \mid E \rangle \mid \{B \stackrel{def}{=} t\} \rangle$ | | | *Dec* |
| $\langle A \mid \{A \stackrel{def}{=} t\} \rangle=\langle t \mid \{A \stackrel{def}{=} t\} \rangle$ | | | *UnFold* |
| $s=t\{s/A\} \Rightarrow s=\langle A \mid \{A \stackrel{def}{=} t\} \rangle$, $A$ is guarded in $t$ | | | *Fold* |
| $\langle A \mid \{A \stackrel{def}{=} A+t\} \rangle=\langle A \mid \{A \stackrel{def}{=} t\} \rangle$ | | | *Ung* |
| $\langle A \mid \{A \stackrel{def}{=} t_1 \oplus_i t_2\} \rangle=\langle A \mid \{A \stackrel{def}{=} t_1\} \rangle \oplus_i \langle A \mid \{A \stackrel{def}{=} t_2\} \rangle$ | | | *Dri* |

$S_{4,5,6}$ define the behavior of communication merge operator; when the operands are ready to do matched input and output communication actions, they can be synchronized and the result of their synchronization is the unobservable action $\tau$ as explained by $S_4$. However, if either they are not matched ($S_5$) or one of the operands cannot do any action ($S_6$), the result is a deadlock.

Axioms $R_{1,4}$ and $E_{1,2,4}$ define the behavior of renaming and restriction operators respectively. Axioms $R_{2,3}$ and $E_{3,5}$ define distributivity of choice and binary variant over the renaming and restriction operators, respectively.

Axiom *Dec* decomposes a recursive specification $E$ made up of multiple (finitely-many) equations into several nested recursive specifications made up of a single equation. Axiom *UnFold* expresses the existence of a solution for any recursive specification $E$: the constant $\langle A \mid E \rangle$ is a solution of the recursive specification $E$. *Fold* expresses uniqueness of a solution for a guarded recursive specification: if $y$ is a solution for $A$ in $E$, and $E$ is guarded, then $y=\langle A \mid E \rangle$. Note that *UnFold* and *Fold* correspond to the *Recursive Definition Principle* (*RDP*) and *Recursive Specification Principle* (*RSP*) in *ACP*. An occurrence of a process name $A$ in $t$ is called *guarded*, if and only if this occurrence is in the scope of an action prefix operator. A recursive specification is called *guarded*, if and only if all occurrences of all its process names in the right-hand sides of all its equations are guarded, or it can be rewritten to such a recursive specification using the axioms of the theory and the equations of the specification [17]. This guardedness criterion ensures that any product line has a unique solution. Axiom *Ung* makes it possible to turn the unguarded recursive specification $\{A \stackrel{def}{=} A+t\}$ into a guarded one. Axiom *Dri* derives the products of a recursive specification: the solution of a recursive specification $\langle A \mid \{A \stackrel{def}{=} t_1 \oplus_i t_2\} \rangle$ is $\langle A \mid \{A \stackrel{def}{=} t_1\} \rangle \oplus_i \langle A \mid \{A \stackrel{def}{=} t_2\} \rangle$. Axioms *UnFold*, *Fold*, and *Ung* are standard for CCS terms (with finite state behaviors modulo branching bisimulation) [22].

Axioms $N_{1-5}$ handle binary variants with an identical index. Whenever the left (right) operand is selected in $p \oplus_i q$, then all $i$-indexed variants in $p$ ($q$), should select their left (right) operands accordingly. Axiom $N_5$ removes all occurrences of $\oplus_i$ at the root of $p \oplus_i q$ in operands $p$ and $q$. In other words, with the help of $N_{1-5}$, the occurrence of $i$ in subtrees of $p \oplus_i q$ becomes unique. For instance, consider the product line $\langle X \mid \{X \stackrel{def}{=} b.X \oplus_1 a.0\} \rangle$, with two products $\langle X \mid \{X \stackrel{def}{=} b.X\} \rangle$ and $\langle X \mid \{X \stackrel{def}{=} a.0\} \rangle$ obtained using axiom *Dri*. However, by applying axiom *UnFold* and substituting of $X$ with its defining term, one can derive $\langle X \mid \{X \stackrel{def}{=} b.X \oplus_1 a.0\} \rangle=(b.(\langle X \mid \{X \stackrel{def}{=} b.X \oplus_1 a.0\} \rangle)) \oplus_1 a.0)=b.(\langle X \mid \{X \stackrel{def}{=} b.X\} \rangle \oplus_1 \langle X \mid \{X \stackrel{def}{=} a.0\} \rangle) \oplus_1 a.0$. By axioms $A_4$ and $N_{1,5}$, $b.(\langle X \mid \{X \stackrel{def}{=} b.X\} \rangle \oplus_1 \langle X \mid \{X \stackrel{def}{=} a.0\} \rangle) \oplus_1 a.0$ is reduced to $b.\langle X \mid \{X \stackrel{def}{=} b.X\} \rangle \oplus_1 a.0$, which derives two products $b.\langle X \mid \{X \stackrel{def}{=} b.X\} \rangle$ and $a.0$ (that are strongly bisimilar to the products of $\langle X \mid \{X \stackrel{def}{=} b.X \oplus_1 a.0\} \rangle$). Axiom $N_6$ changes the index of a binary variant term. For example, $(a.0 \oplus_1 b.0) \oplus_2 (c.0 \oplus_1 d.0)=(a.0 \oplus_1 b.0) \oplus_2 (c.0 \oplus_3 d.0)$. Consequently, axiom $A_2$ can be applied, resulting $(a.0 \oplus_1 b.0) \oplus_2 (c.0 \oplus_1 d.0)=a.0 \oplus_1 (b.0 \oplus_2 (c.0 \oplus_3 d.0))$.

It should be noted that $t \approx_{PL} s$ implies $t \simeq_{PL} s$. All axioms, except for $A_{1,2}$ and $N_6$, are supported by strict strong bisimilarity. Generally speaking, to manipulate a PL-CCS term, as stated in Theorem 12, first it can be converted into a fully expanded form with the help of axioms supported by strict strong bisimilarity, and then manipulated with all axioms.

**Theorem 12.** *Any PL-CCS term $t$ can be rewritten by axioms $A_{4-6}$, $D_{1,2}$, $P_{4,5}$, $S_{1,3}$, $R_3$, $E_5$, $Dri$, and $N_{1-5}$ into a form that is fully expanded.*

See Appendix B for the proof. With the help of axioms $A_{4-6}$, $P_{4,5}$, $S_{1,3}$, $R_3$, $E_5$, and $Dri$, one can convert a PL-CCS term into a term such that its binary variants do not occur in the scope of any CCS operators. Then, by axioms $N_{1-5}$, the term becomes fully expanded. Therefore, it can be manipulated using all the axioms. The soundness of derivations, when the order of binary variants operands are fixed, follows from Theorem 9, and when terms are fully expanded, follows from Theorem 11.

**Theorem 13** *(Soundness). The axiomatization, given in Tables 1 and 2, is sound for the term algebra $\mathbb{P}(\text{PL-CCS})/\simeq_{PL}$, i.e., for all closed PL-CCS terms $t_1$ and $t_2$, $t_1 = t_2$ implies $t_1 \simeq_{PL} t_2$.*

See Appendix C for the proof.

**Example 14.** Axioms $D_{1,2}$ in Table 1 can be derived from the other remaining axioms. The derivation for $D_1$ is:

$$\begin{aligned}
r \parallel (p \oplus_i q) &=^{P_1} r \between (p \oplus_i q) + (p \oplus_i q) \between r + r \mid (p \oplus_i q) \\
&=^{P_{4,5},S_{2,3}} (r \between p \oplus_i r \between q) + (p \between r \oplus_i q \between r) + (r \mid p \oplus_i r \mid q) \\
&=^{A_{5,6}} (((r \between p + p \between r + r \mid p) \oplus_i (r \between p + p \between r + r \mid q)) \oplus_i \\
&((r \between p + q \between r + r \mid p) \oplus_i (r \between p + q \between r + r \mid q))) \oplus_i \\
&(((r \between q + p \between r + r \mid p) \oplus_i (r \between q + p \between r + r \mid q)) \oplus_i \\
&((r \between q + q \between r + r \mid p) \oplus_i (r \between q + q \between r + r \mid q))) \\
&=^{N_{1,3,5}} (r \between p + p \between r + r \mid q) \oplus_i (r \between q + q \between r + r \mid q) \\
&=^{P_1} (r \parallel p) \oplus_i (r \parallel q).
\end{aligned}$$

In [13], a set of algebraic laws was provided including $A_4$, $A_6$, $D_1$, and $Dri$. Their algebraic laws are sensitive to the numbering of variants and the placement of their operands. Therefore, the laws do not support idempotence and commutativity properties of the binary variant (axioms $A_{1,3}$). By prohibiting application of axioms $A_{1,2}$ and $N_6$ while removing variants with an identical index with the help of axioms $N_{1-5}$, we can rewrite a term into a fully expanded one. Subsequently, with the help of axioms $A_{1,2}$ and $N_6$, our axiomatization becomes insensitive to the placement of operands in binary variants or binary variant indices.

### 4.3. Completeness of the axiomatization for finite-state behaviors

We prove that the axiomatization in Tables 1 and 2 is ground-complete for PL-CCS terms with finite-state models modulo product line bisimilarity. Following the approach of [17], to restrict to PL-CCS terms with finite-state transition systems, we provide a syntactical restriction for constants $\langle A|E\rangle$. We consider so-called *finite-state PL-CCS*, denoted by PL-CCS$_f$, which is obtained by extending PL-CCS with essentially finite-state recursive specifications: a recursive specification $E$ is essentially finite-state, if it has only finitely many equations and in the right-hand sides of all equations of $E$, no process name occurs in the scope of static operators, namely, parallel composition, left- and communication merge, restriction, and renaming operators.

For instance, PL-CCS term $\langle Y|\{Y \stackrel{def}{=} (a.0 \oplus_1 b.0) \parallel c.Y\}\rangle$ is not a finite-state PL-CCS process. To see this, observe that it can derive the sequence of transitions $\langle Y|\{Y \stackrel{def}{=} (a.0 \oplus_1 b.0) \parallel c.Y\}\rangle \xrightarrow{c,\langle ?\rangle} (a.0 \oplus_1 b.0) \parallel \langle Y|\{Y \stackrel{def}{=} (a.0 \oplus_1 b.0) \parallel c.Y\}\rangle \xrightarrow{c\langle ?\rangle} (a.0 \oplus_1 b.0) \parallel (a.0 \oplus_1 b.0) \parallel \langle Y|\{Y \stackrel{def}{=} (a.0 \oplus_1 b.0) \parallel c.Y\}\rangle \xrightarrow{c,\langle ?\rangle} \ldots$, which leads to an infinite state space. This sequence results from the occurrence of process name $Y$ in the context of a parallel composition.

**Proposition 15** *(Finite-state behaviors). Consider a PL-CCS$_f$ term $t$; the transition system for $t$ generated by the operational rules has only finitely many states.*

This Proposition can be proved by resolving the binary variants, which are finite. Therefore, a finite set of CCS terms are derived such that each CCS term generates finitely many states [22].

**Theorem 16** *(Completeness). The axiomatization, given in Tables 1 and 2 is ground-complete for the term algebra $\mathbb{P}(\text{PL-CCS}_f)/\simeq_{PL}$, i.e., for all closed finite-state PL-CCS$_f$ terms $t_1$ and $t_2$, $t_1 \simeq_{PL} t_2$ implies $t_1 = t_2$.*

See Appendix D for the proof.

## 5. Product line analysis

The advantage of our sound and complete axiomatization is that we can prove equality of PL-CCS terms at a syntactic level by transforming one term to another. Hence, one does not need to generate the huge state space, which was required to check the notions of bisimilarity introduced in Section 3. This process can be facilitated and mechanized with the help of theorem provers, or term rewriting systems. Consequently, terms can be transformed by our axiomatization into a basic form, such as *linear process specifications* in the mCRL2 language [24,25], over which different analyses can be performed, either manually or using tools. This basic form acts as a symbolic representation of the state space of a model, which is comparatively small. A set of tools such as model checker, state space visualizer, and behavioral simulator exist that run over this basic representation and can be adapted to our setting. Furthermore, a number of optimization approaches such as $\tau$-confluence reduction [26], that work on the level of this basic format, can simplify it prior to any analysis. The transformation process into a basic form can be mechanized in the same way as [27] within a small amount of time. Similarly, PL-CCS terms can be reduced to their possible products (which are simple CCS terms) in a syntactic way to be validated in terms of their intended properties.

A formal framework for modeling and analyzing SPLs should support modular design, derivation (configuration) of individual systems from a product line model, and restructuring them into various syntactic forms [13]. Our process theory supports them all. In our case, we support a few different forms of restructuring: for example, we support "moving variation points throughout the hierarchical specification of an SPL towards its leaves or its root" [13]. We also support "modeling individual systems using a higher or lower degree of common parts" [13]. Therefore, a designer can model the functionality of an SPL irrespective of the existing components. Later with the aim of reuse, the functionality can be restructured to behaviors for which appropriate components exist. A restructuring mechanism is also appealing when a new functionality (which corresponds to a new feature) is added. We illustrate how our framework supports deriving products or restructuring of SPLs in following sections.

### 5.1. Deriving products of a family

Using our axiomatization, one can derive the products of a family, i.e., rewrite a process term into a term which comprises binary variants of CCS terms.

**Example 17.** For instance, consider CCS terms $p$, $q$, and $s$ (which naturally do not contain the binary variant operator); the family $(q \oplus_1 (a.(s \oplus_2 p))) \oplus_3 a.s$ can generate three pairwise non-bisimilar products:

$$
\begin{aligned}
&(q \oplus_1 (a.(s \oplus_2 p))) \oplus_3 a.s =^{A_2} \\
&\quad q \oplus_1 ((a.(s \oplus_2 p)) \oplus_3 a.s) =^{A_4} q \oplus_1 ((a.s \oplus_2 a.p) \oplus_3 a.s) =^{A_1} \\
&\quad q \oplus_1 ((a.p \oplus_2 a.s) \oplus_3 a.s) =^{A_2} q \oplus_1 (a.p \oplus_2 (a.s \oplus_3 a.s)) =^{A_3} \\
&\quad q \oplus_1 (a.p \oplus_2 a.s)
\end{aligned}
$$

**Example 18.** We can compare product lines $\langle p_1 | \{p_1 \stackrel{def}{=} p_2 \parallel p_2,\ \ p_2 \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle$ and $\langle p'_1 | \{p'_1 \stackrel{def}{=} b.0 \oplus_1 c.0 \parallel b.0 \oplus_2 c.0\}\rangle$. $p_1$ generates two non-bisimilar products, while $p'_1$ generates three pairwise non-bisimilar products, concluding $p_1 \neq p'_1$. To see this, observe the following derivations:

$$
\begin{aligned}
p_1 &=^{UnFold} \langle p_2 | \{p_1 \stackrel{def}{=} p_2 \parallel p_2, \\
&\qquad p_2 \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle \parallel \langle p_2 | \{p_1 \stackrel{def}{=} p_2 \parallel p_2,\ p_2 \stackrel{def}{=} b.0 \oplus_1 c.0\}\rangle \\
&=^{UnFold} (b.0 \oplus_1 c.0) \parallel (b.0 \oplus_1 c.0) \\
&=^{P_1,C_3} ((b.0 \oplus_1 c.0) \between (b.0 \oplus_1 c.0)) + ((b.0 \oplus_1 c.0) \mid (b.0 \oplus_1 c.0)) \\
&=^{P_{4,5},S_3} ((b.0 \between b.0 \oplus_1 c.0 \between b.0) \oplus_1 (b.0 \between c.0 \oplus_1 c.0 \between c.0)) + \\
&\qquad ((b.0 \mid b.0 \oplus_1 c.0 \mid b.0) \oplus_1 (b.0 \mid c.0 \oplus_1 c.0 \mid c.0)) \\
&=^{N_1,3,5} (b.0 \between b.0) \oplus_1 (c.0 \between c.0) + (b.0 \mid b.0) \oplus_1 (c.0 \mid c.0) \\
&=^{P_2,S_5,C_{3,4}} b.(0 \parallel b.0) \oplus_1 c.(0 \parallel c.0) \\
&=^{P_{1,6},S_6} b.b.0 \oplus_1 c.c.0
\end{aligned}
$$

$$
\begin{aligned}
p'_1 &=^{UnFold} (b.0 \oplus_1 c.0) \parallel (b.0 \oplus_2 c.0) \\
&=^{D_{1,2}} ((b.0 \parallel b.0) \oplus_2 (b.0 \parallel c.0)) \oplus_1 ((c.0 \parallel b.0) \oplus_2 (c.0 \parallel c.0)) \\
&=^{P_{1,6},S_6} (b.b.0 \oplus_2 (b.c.0 + c.b.0)) \oplus_1 ((c.b.0 + b.c.0) \oplus_2 c.c.0) \\
&=^{A_{1-3},C_1} b.b.0 \oplus_1 ((b.c.0 + c.b.0) \oplus_2 c.c.0)
\end{aligned}
$$

Next, we show that every PL-CCS term can be rewritten into a normal form comprising binary choices over CCS products.

**Theorem 19.** *Let* $\bigoplus_{i\leq n} p_i$ *denote* $(p_0 \oplus_1 (p_1 \oplus_2 (\ldots \oplus_n p_n)\ldots))$ *if* $n > 0$, *and* $p_0$ *if* $n = 0$. *Using the axiomatization in Tables 1 and 2, each PL-CCS term* $t$ *can be rewritten into the form* $\bigoplus_{i\leq n} p_i$, *where* $p_i$*s are CCS processes.*

See Appendix B for the proof. With the help of axioms $A_{4-6}$, $D_{1,2}$, $P_{4,5}$, $S_{1,3}$, $R_3$, $E_5$, and *Dri*, one can convert a PL-CCS term into another term such that its binary variants do not occur in scope of any CCS operators. Later by axiom $N_{1-6}$, the indices of variants become unique. Therefore, by $A_{1,2}$, it can be rewritten into the desired format.

### 5.2. Restructuring a product family

With the help of our axiomatization, we can factorize the common parts and simplify the structure of product line terms. We can also identify the mandatory parts of a product line; the parts that exist in any product.

**Example 20.** Consider a *Sensor* process that is replicated in different parts of a car windscreen *WindScreen*, such as wiper *WipFam* and fog remover *FogFam* [12]:

$$\begin{aligned} &WindScreen \stackrel{def}{=} WipFam \oplus_1 FogFam\\ &WipFam \stackrel{def}{=} Sensor \parallel Wiper\\ &FogFam \stackrel{def}{=} Sensor \parallel FogRem \end{aligned}$$

where *Sensor* detects the different conditions of precipitation, *Wiper* and *FogRem* offer different operational modes for wiper arm movement, and windscreen warmer concerning environmental conditions, respectively. Using our axioms, *WindScreen* specification is restructured as follows:

$$\begin{aligned} WindScreen &\stackrel{def}{=} WipFam \oplus_1 FogFam\\ &=^{UnFold} (Sensor \parallel Wiper) \oplus_1 (Sensor \parallel FogRem)\\ &=^{D_1} Sensor \parallel (Wiper \oplus_1 FogRem) \end{aligned}$$

The new specification for *WindScreen* reveals that *Sensor* is the mandatory part of our windscreen family. The structural specification (i.e., the architecture) of *WindScreen* consists of two components, a *Sensor* and a component, which can be either *Wiper* or *FogRem*.

Assume that the *Sensor* has two qualities, namely, high and low. The low quality sensor cannot distinguish between *heavy* and *little* rain (specified by *hvy* and *ltl* actions) and can only discriminate between *no rain* and *rain* [12]. The high quality sensor can, however, make such distinctions as specified below:

$$\begin{aligned} &Sensor \stackrel{def}{=} SensL \oplus_2 SensH\\ &SensL \stackrel{def}{=} non.SensL + ltl.Raining + hvy.Raining + \overline{noRain}.SensL\\ &Raining \stackrel{def}{=} non.SensL + ltl.Raining + hvy.Raining + \overline{rain}.Raining\\ &SensH \stackrel{def}{=} non.SensH + ltl.Medium + hvy.Heavy + \overline{noRain}.SensH\\ &Medium \stackrel{def}{=} non.SensH + ltl.Medium + hvy.Heavy + \overline{rain}.Medium\\ &Heavy \stackrel{def}{=} non.SensH + ltl.Medium + hvy.Heavy + \overline{hvyRain}.Heavy \end{aligned}$$

The specification of *Sensor* can be similarly examined to reveal the common behaviors. To this aim, we first restructure $SensL \oplus_2 SensH$ to factor out the common behaviors:

$$\begin{aligned} SensL \oplus_2 SensH =^{UnFold}&\\ &(non.SensL + ltl.Raining + hvy.Raining + \overline{noRain}.SensL) \oplus_2\\ &(non.SensH + ltl.Medium + hvy.Heavy + \overline{noRain}.SensL) =^{N_{1,3,5}}\\ &((non.SensL + ltl.Raining + hvy.Raining + \overline{noRain}.SensL) \oplus_2\\ &\quad (non.SensH + ltl.Raining + hvy.Raining + \overline{noRain}.SensL)) \oplus_2\\ &((non.SensL + ltl.Medium + hvy.Heavy + \overline{noRain}.SensH) \oplus_2\\ &\quad (non.SensH + ltl.Medium + hvy.Heavy + \overline{noRain}.SensH)) =^{A_{4,5}}\\ &(non.(SensL \oplus_2 SensH) + ltl.Raining + hvy.Raining + \overline{noRain}.SensL) \oplus_2\\ &(non.(SensL \oplus_2 SensH) + ltl.Medium + hvy.Heavy + \overline{noRain}.SensH) =^{UnFold} \end{aligned}$$

$$(non.Sensor + ltl.Raining + hvy.Raining + \overline{noRain}.SensL) \oplus_2$$
$$(non.Sensor + ltl.Medium + hvy.Heavy + \overline{noRain}.SensH) =^{N_{1,3,5}, A_{4,5}}$$
$$non.Sensor + ltl.(Raining \oplus_2 Medium) + hvy.(Raining \oplus_2 Heavy) +$$
$$\overline{noRain}.Sensor$$

Similarly $Raining \oplus_2 Medium$ and $Raining \oplus_2 Heavy$ can be examined:

$$Raining \oplus_2 Medium =^{UnFold, N_{1,3,5}, A_{4,5}}$$
$$non.(SensL \oplus_2 SensH) + ltl.(Raining \oplus_2 Medium) + hvy.(Raining \oplus_2 Heavy) +$$
$$\overline{rain}.(Raining \oplus_2 Medium)$$
$$Raining \oplus_2 Heavy =^{UnFold, N_{1,3,5}, A_{4,5}}$$
$$non.(SensL \oplus_2 SensH) + ltl.(Raining \oplus_2 Medium) + hvy.(Raining \oplus_2 Heavy) +$$
$$(\overline{rain}.(Raining \oplus_2 Heavy) \oplus_2 \overline{hvyRain}.(Raining \oplus_2 Heavy))$$

By applying *Fold*, the new specification of *Sensor* is obtained as follows:

$$Sensor \stackrel{def}{=} non.Sensor + ltl.RainMed + hvy.RainHvy + \overline{noRain}.Sensor$$
$$RainMed \stackrel{def}{=} non.Sensor + ltl.RainMed + hvy.RainHvy + \overline{rain}.RainMed$$
$$RainHvy \stackrel{def}{=} non.Sensor + ltl.RainMed + hvy.RainHvy +$$
$$(\overline{rain}.RainHvy \oplus_2 \overline{hvyRain}.RainHvy)$$

The new specification explains that resolving variability between *SensL* and *SensH* can be postponed until the variability between performing output actions $\overline{rain}$ and $\overline{hvyRain}$ is resolved in case it is possible to have heavy rain.

In [28], PL-CCS was compared with FTS in addressing variability via modeling the above-mentioned example. There, it was concluded that modeling in PL-CCS can result in verbose descriptions since common parts have to be duplicated [29]. However, in our experience, PL-CCS facilitates modular design without forcing the designer to factor out common parts. Later the specification can be restructured as illustrated by the above-given example. For instance, actions *non*, *ltl*, *hvy*, and $\overline{noRain}$ are common among *SensL* and *SensR*, while their behaviors does not change after performing actions *non* and $\overline{noRain}$. Such common actions and behaviors are factored our by rewriting $SensL \oplus_2 SensR$ to $non.Sensor + ltl.(Raining \oplus_2 Medium) + hvy.(Raining \oplus_2 Heavy) + \overline{noRain}.Sensor$. Therefore, the modeler is not forced to identify common actions and behaviors to derive its model. The semantics of our resulting specification is even more compact than the FTS model of [28] by factorizing out common behaviors as much as possible; the part of behavior in which both sensors does not change their behaviors as long as action *hvy* is performed, is factored out in our case.

## 6. Logical characterization

In this section, we show that product line bisimilarity induces the same identification of PL-CCS terms as the multi-valued modal $\mu$-calculus. In this section, we first review the logic and then explain how it characterizes product line bisimilarity.

### 6.1. Multi-valued modal $\mu$-calculus

The multi-valued modal $\mu$-calculus [12] combines Kozen's modal $\mu$-calculus [30] and multi-valued $\mu$-calculus as defined by Grumberg and Shoham [31]. Let $\mathcal{V}$ be the set of propositional variables. The set of multi-valued modal $\mu$-calculus formulae is given by the following grammar:

$$\varphi \quad ::= \quad true \mid false \mid Z \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid \eta Z.\varphi,\ \eta \in \{\nu, \mu\}$$

where $a \in Act$ and $Z \in \mathcal{V}$, and the fixed point quantifiers $\mu$ and $\nu$ are variable binders.

Let $mv - \mathfrak{L}_\mu$ denote the set of closed formulae generated by the above grammar. The semantics of a formula for a PL-CCS term is the set of configurations satisfying it. All configurations satisfy *true* in all states. A configuration $\nu'$ satisfies a formula $\phi_1 \vee \phi_2$ in state $s$ if it satisfies either $\phi_1$ or $\phi_2$ in state $s$. A configuration $\nu'$ satisfies a formula $\langle a \rangle \varphi$ in state $s$ if it has a transition $s \xrightarrow{a,\nu} s'$ such that $\nu \sqsubseteq \nu'$ and $\nu'$ satisfies $\varphi$ in state $s'$. A configuration $\nu'$ satisfies a formula $[a]\varphi$ in state $s$ if for all transitions $s \xrightarrow{a,\nu} s'$ such that $\nu \sqsubseteq \nu'$, $\nu'$ satisfies $\varphi$ in state $s'$. Equations with recursive variables are used to describe properties of behaviors with an infinite depth. For instance, $X \stackrel{def}{=} \langle a \rangle X \vee \langle b \rangle true$ specifies configurations

**Table 3**
Semantics of multi-valued modal $\mu$-calculus [12].

| | |
|---|---|
| $[\![true]\!]_\rho = \lambda s.\mathbb{P}(Config)$ | $[\![\langle a\rangle\varphi]\!]_\rho = \lambda s.\bigcup\{\{\nu \mid \exists s' \cdot s \xrightarrow{\nu,a} s'\} \cap [\![\varphi]\!]_\rho(s')\}$ |
| $[\![false]\!]_\rho = \lambda s.\emptyset$ | $[\![[a]\varphi]\!]_\rho = \lambda s.\bigcap\{\{\nu \mid \nexists s' \cdot s \xrightarrow{\nu,a} s'\} \cup [\![\varphi]\!]_\rho(s')\}$ |
| $[\![Z]\!]_\rho = \rho(Z)$ | $[\![\mu Z.\varphi]\!]_\rho = \bigcap\{f \mid [\![\varphi]\!]_{\rho[Z\mapsto f]} \subseteq f\}$ |
| $[\![\varphi_1 \wedge \varphi_2]\!]_\rho = [\![\varphi_1]\!]_\rho \cap [\![\varphi_2]\!]_\rho$ | $[\![\nu Z.\varphi]\!]_\rho = \bigcup\{f \mid f \subseteq [\![\varphi]\!]_{\rho[Z\mapsto f]}\}$ |
| $[\![\varphi_1 \vee \varphi_2]\!]_\rho = [\![\varphi_1]\!]_\rho \cup [\![\varphi_2]\!]_\rho$ | |

(i.e., products) that they satisfy $\langle b\rangle true$ either in the initial state, or the state reached after performing a (possibly infinite) sequence of $a$-actions (with consistent configurations). Since an equation may have many solutions, the maximum and minimum solutions are selected by $\nu Z.\phi$ and $\mu Z.\phi$, respectively. Considering $Z$ as a mapping from the states to a set of configurations, $\mu Z.\phi$ is valid for the smallest mapping $Z$ that satisfies the equation $Z = \phi$. Similarly $\nu Z.\phi$ is valid for the largest mapping $Z$ that satisfies equation $Z = \phi$. For instance, the property $\mu X.\langle a\rangle X \vee \langle b\rangle true$ specifies that "eventually an action $b$ follows a (possibly empty) sequence of $a$ actions". This property holds for the configuration $\langle L, L\rangle$ in the state $\langle X|E\rangle$ of CTS in Fig. 3a.

The semantics of $\varphi$, denoted by $[\![\varphi]\!]$, is a function $\mathcal{S} \to \mathbb{P}(Config)$ that defines the set of configurations that satisfy formula $\varphi$ for each given state. Given an environment $\rho : \mathcal{V} \to (\mathcal{S} \to \mathbb{P}(Config))$, which maps free variables in $\varphi$ to $\mathcal{S} \to \mathbb{P}(Config)$, $[\![\varphi]\!]_\rho$ defines the semantics of $\varphi$ with respect to $\rho$ in Table 3.

**Example 21.** Regarding the rules in Table 3, the semantics of $[\![\langle b\rangle true]\!]$ in the state $(b.\langle X|E\rangle \oplus_1 c)$ of the CTS in Fig. 3a is $\{\langle L, R\rangle, \langle L, L\rangle\}$, since $\mathcal{R}_b(b.\langle X|E\rangle \oplus_1 c.0, \langle X|E\rangle) = \{\langle L, R\rangle, \langle L, L\rangle\}$ and $[\![true]\!](\langle X|E\rangle) = \mathbb{P}(Config)$. It can be shown that $f = \{(b.\langle X|E\rangle \oplus_1 c.0) \mapsto \{\langle L, R\rangle, \langle L, L\rangle\}, \langle X|E\rangle \mapsto \{\langle L, L\rangle\}, 0 \mapsto \emptyset\}$ is the semantics of $\mu X.\langle a\rangle X \vee \langle b\rangle true$ since $[\![\langle a\rangle X]\!]_{[X\mapsto f]} = \lambda s.\bigcup\{\mathcal{R}_a(s, s') \cap X(s')\} = \{\langle X|E\rangle \mapsto \{\langle L, L\rangle\}, b.\langle X|E\rangle \oplus_1 c \mapsto \emptyset, 0 \mapsto \emptyset\}$, $[\![\langle b\rangle true]\!] = \{b.\langle X|E\rangle \oplus_1 c \mapsto \{\langle L, R\rangle, \langle L, L\rangle\}, \langle X|E\rangle \mapsto \emptyset, 0 \mapsto \emptyset\}$, $f = [\![\langle a\rangle X \vee \langle b\rangle true]\!]_{[X\mapsto f]}$, and it is the minimum mapping that satisfies the equation $X = \langle a\rangle X \vee \langle b\rangle true$.

### 6.2. Relation to product line bisimilarity

Model checking logical formula $\phi$ over a PL-CCS term is supposed to result in the set of full configurations for which the property holds. Intuitively, two PL-CCS terms are logically equivalent when for each logical formula, there exists a non-empty set of products in one product line satisfying it if and only if there exists such a non-empty set in the other. For instance, $(a.0 + b.0) \oplus_1 b.0$ is not logically equivalent to $a.0 \oplus_1 b.0$ as the logical formula $\langle a\rangle true \wedge \langle b\rangle true$ is satisfied by the former for the configuration $\langle L\rangle$, but it is not satisfied by the latter for any configuration.

**Definition 22** *(Logical equivalence).* Two PL-CCS terms $s$ and $t$ are *logically equivalent*, denoted by $s \sim_L t$, iff $\forall\varphi \in mv - \mathfrak{L}_\mu \cdot ([\![\varphi]\!](s) \neq \emptyset \Leftrightarrow [\![\varphi]\!](t) \neq \emptyset)$.

As stated before, product line bisimilarity and logical equivalence coincide. The following Theorem states that if two PL-CCS terms are not product line bisimilar, then there is a logical formula that can distinguish them.

**Theorem 23.** *For any PL-CCS terms $s$ and $t$, $s \simeq_{PL} t$ iff $s \sim_L t$.*

See Appendix E for the proof.

## 7. Related work

There are a vast number of languages to specify software product lines for the purpose of specifying different aspects of variability [32–38,12,13,39,29,40,10,41–44]. Among them some frameworks, such as [32–35], do not directly address formal reasoning. On the other hand, there are formal frameworks to reason about some aspects of SPLs, such as [36–38,12,13,39, 29,40,10,41–44].

Regarding modeling issues, several approaches are classified by [29] in terms of treating variability as either a first class citizen [34,39,38,41–44] or as part of the behavioral model [35–37,32,12,33]. In the former approaches, variability is separately modeled and related to other models (data and behavior), called base models. Therefore, variability is explicitly traceable in base models and its evolution is automatically propagated to the base models. As opposed to other process-algebraic approaches [41,42,44], PL-CCS expresses variability as part of its behavioral model; In [41,42], the cross-tree constraints of feature models are related to the behavior of products using a CCS-like process algebra, while in [44], process terms are tagged with the sets of specific products where they are enabled using a CSP-like process algebra. In [11], some of the fundamental formal behavioral models for SPLs are compared in terms of their expressiveness.

Our approach follows the line of research on process algebra for SPLs [12,13,41,42,45,44]. It also offers several reasoning capability: model-checking based on a multi-valued modal $\mu$-calculus over CTSs given in [12], and equational reasoning based on a set of rules to restructure PL-CCS terms, extending the approach of [13]. Along these lines, the safety and

liveness properties of SPLs as well as consistency of configurations are checked in [41,42] by encoding their semantic rules in the Maude rewriting logic. In [45], the algebraic framework mCRL2 [24,25], is used for modular verification of SPLs. There, tailored property-preserving reductions were applied to a product line modeled in mCRL2 using the reduction modulo branching bisimulation of mCRL2 tool set. Two pre-congruence behavioral relations to compare the behavior of a product either against a product or a family are proposed in [44]. In contrast, we offer a set of behavioral equivalence relations over CTSs supported by a sound and complete axiomatization to reason about families at the syntactic level. To this aim, a family can be restructured to another, while its functionality is preserved. Our approach is hence complementary to the aforementioned approaches in the literature.

The work of [46] is analogous to us in that it takes an axiomatic approach to software product lines. However, there approach has a different intention, namely, to axiomatizes product family concepts that characterize a generic product line formalism. Their approach indicates that all operators of a formalism are distributive over the binary variant operator. Our axioms $D_{1,2}$, $A_{5,6}$, $P_{4,5}$, $S_3$ (together with $S_1$), $R_3$, and $E_5$ conform to this result. Furthermore, it expresses that binary variant operators with different indices are distributive and provides rules to simplify specifications when two $i$-indexed variants are directly nested. These rules are derivable in our setting by axioms $A_3$ and $N_{1-6}$ as follows:

$$
\begin{aligned}
(P \oplus_j Q) \oplus_i (P \oplus_j O) &=^{A_3} \\
&((P \oplus_j Q) \oplus_i (P \oplus_j O)) \oplus_j ((P \oplus_j Q) \oplus_i (P \oplus_j O)) =^{N_5} \\
&L((P \oplus_j Q) \oplus_i (P \oplus_j O), j) \oplus_j R((P \oplus_j Q) \oplus_i (P \oplus_j O), j) =^{N_{1-4}} \\
&(L(P, j) \oplus_i L(P, j)) \oplus_j (R(Q, j) \oplus_i R(O, j)) =^{N_2, N_4} \\
&L(P \oplus_i P, j) \oplus_j R(Q \oplus_i O, j) =^{A_3, N_5} P \oplus_j (Q \oplus_i O) \\
P \oplus_i (Q \oplus_i O) &=^{N_5} L(P, i) \oplus_i R(Q \oplus_i O, i) =^{N_4} \\
&L(P, i) \oplus_i R(O, i) =^{N_5} P \oplus_i O
\end{aligned}
$$

Similarly, $(P \oplus_j O) \oplus_i (Q \oplus_j O) = (P \oplus_j Q) \oplus_i O$ and $(P \oplus_i Q) \oplus_i O = P \oplus_i O$ can be derived from our equational theory.

Other related techniques to our behavioral equivalence are conformance notions that are used to iteratively refine partial behavioral models; they can consequently be used to relate a product behavior to a family model. Refinement as well as conformance between SPLs modeled by modal I/O automata is studied in [37]. A notion of behavioral conformance on MTS-based specifications is defined in [36], which preserves 3-valued weak $\mu$-calculus. Modal transition systems (MTS) [36] and I/O modal automata [37] capture variability by defining transitions as optional and mandatory. A notion of input–output conformance on FTSs is defined in [47,48] with the aim of devising a model-based testing trajectory for SPLs. In [49], pre-orders over FTSs preserving LTL properties are given with respect to specific products. As opposed to these approaches, our equivalence relation is defined over CTS and preserves multi-valued modal $\mu$-calculus. Providing a comprehensive and formal comparison of the different notions of SPL pre-orders in the literature is among our future work. In [16], a feature-oriented notion of branching bisimulation over FTSs and its associated minimization algorithm were introduced in order to reduce a model prior to its verification. Our configuration bisimulation was inspired by [16] and adopting its minimization algorithm is also among our future directions for research.

Remaining approaches mainly use a model checking technique to reason about SPLs; these include checking safety properties over Statecharts in [38], LTL over FTSs [29], CTL over modal I/O automata [39], MHML, a deontic logic interpreted over MTSs [40], and fLTL (an extension of LTL) over FTSs in [10].

## 8. Conclusions and future work

We proposed an equational reasoning technique to reason about software product lines at the syntactic level. To this aim, we defined product line bisimilarity by finding a mapping between products of two terms, identified by their configuration vectors. We also introduced a configuration-oriented bisimilarity that compares families at once. We proved that product line and configuration bisimilarity coincide. To facilitate checking the bisimilarity relations, we provided a sound and complete axiomatization over closed and finite-state behaviors. We characterized the distinguishing power of our equivalence relations in terms of a multi-valued modal $\mu$-calculus.

Instead of working at the semantic level and finding a mapping between products, one can use our axioms and restructure a term to its equivalent terms, e.g., such that the mandatory and optional parts are factored out separately. The restructuring mechanism can also be initiated to group the functionality of an SPL to behaviors for which appropriate components exist. Furthermore, one can derive the possible products of a term, specified by CCS terms to validate an SPL model in terms of its various products.

We intend to exploit the PL-CCS process theory as a formal framework for specifying the structural and behavioral aspects of product lines, following the approach of [50]. We intend to investigate a basic form, such as linear process specification in mCRL2, over which different analysis and optimizations can be executed. Then, PL-CCS terms can be automatically transformed into the basic form in the same way of [27]. Finding a minimization algorithm for configuration bisimulation is another line of research. Furthermore, pre-order notions of literature can be compared in the general setting of FTS, which is a very expressive model for SPLs [11].

**Acknowledgements**

We would like to thank Wan Fokkink for his valuable comments on the earlier version of this paper.

## Appendix A. Proofs of Theorems 6, 9, and 11

We first prove that product line bisimilarity is an equivalence relation, and then prove its congruence property on fully expanded PL-CCS terms. Later, we show that strict strong bisimilarity is an equivalence relation and constitutes a congruence on PL-CCS terms.

### A.1. Proof of Theorem 6

**Theorem 6.** *Product line bisimilarity is an equivalence relation.*

**Proof.** To show that product line bisimilarity is an equivalence, we must show that it is reflexive, symmetric, and transitive. Reflexivity and symmetry follow immediately from the reflexivity and symmetry properties of strong bisimilarity. Hence, it only remains to prove transitivity.

Consider PL-CCS terms $s, t, r$ such that $s \simeq_{PL} t$, and $t \simeq_{PL} r$. Following Definition 5, for any valid full configuration $\nu_1^f$ with respect to $s$, there exists a valid full configuration $\nu_2^f$ with respect to $t$ such that $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$. For any $\nu_2^f$, there exists a valid full configuration $\nu_3^f$ with respect to $r$ such that $\Pi(t, \nu_2^f) \sim \Pi(r, \nu_3^f)$. Transitivity of strong bisimilarity results in $\Pi(s, \nu_1^f) \sim \Pi(r, \nu_3^f)$. The same argument holds for any valid full configuration $\nu_3^f$ with respect to $r$, concluding that $s \simeq_{PL} r$. □

### A.2. Proof of Theorem 11

**Theorem 11.** *Product line bisimilarity constitutes a congruence on fully expanded PL-CCS terms.*

**Proof.** Consider arbitrary product line $r$ such that $s \odot r$ and $t \odot r$ are fully expanded, where $\odot \in \{+, \oplus, \|\}$ and $s \simeq_{PL} t$; also consider renaming function $\phi$, $L \subseteq Act$; to show that product line bisimilarity is a congruence on fully expanded PL-CCS terms we need to prove the following statements:

1. $\alpha.s \simeq_{PL} \alpha.t$;
2. $s + r \simeq_{PL} t + r$;
3. $s \oplus_i r \simeq_{PL} t \oplus_i r$;
4. $s \setminus L \simeq_{PL} t \setminus L$;
5. $s[\phi] \simeq_{PL} t[\phi]$;
6. $s \| r \simeq_{PL} t \| r$;

We only prove cases 1, 3, and 6 as the proof of remaining cases is almost identical. Let $\nu \cdot \lambda$ denote the concatenation of two configuration vectors $\nu$ and $\lambda$ by appending the elements of $\lambda$ at the end of $\nu$. Furthermore, assume that $|\nu|$ denotes the length of configuration vector $\nu$, and $max(S)$ denotes the maximum index in set $S$ such that $max(\emptyset) = 0$. Note that any full configuration $\nu^f$ with respect to $s \odot r$ can be written as either $\nu_s \cdot \lambda_1$ or $\nu_r \cdot \lambda_2$ for some $\nu_s \in VFConfig(s)$ and $\nu_r \in VFConfig(r)$. The following two lemmata are required for the proof. In following proofs, we use $\equiv$ to denote syntactic equivalence. □

**Lemma 24.** *For each PL-CCS term $t$, $t \xrightarrow{a,\nu} t'$, where $|\nu| \geq max(bi(t))$, implies $t \xrightarrow{a,\nu\cdot\lambda_?} t$.*

**Proof.** The proof is straightforward by induction on the structure of $t$. The only interesting cases are given below:

- $t \equiv t_1 \oplus_i t_2$: by SOS rule *Select*, $t \xrightarrow{a,\nu} t_1'$, since $t_1 \xrightarrow{a,\nu'} t_1'$ and $\nu'|_i \neq R$, and $\nu = \nu'|_{i/L}$. By induction, $t_1 \xrightarrow{a,\nu'\cdot\lambda_?} t_1'$ while $\nu'|_i \neq R \Rightarrow (\nu' \cdot \lambda_?)|_i \neq R$. Consequently by SOS rule *Select*, $t \xrightarrow{a,\nu\cdot\lambda_?} t_1'$ holds. The same discussion holds when $t \xrightarrow{a,\nu} t_2'$ as the result of $t_2 \xrightarrow{a,\nu'} t_2'$.
- $t \equiv t_1 \| t_2$: by SOS rule *Sync*, we have $t \xrightarrow{\tau,\nu'\odot\nu''} t_1' \| t_2'$, since $t_1 \xrightarrow{a,\nu'} t_1'$ and $t_2 \xrightarrow{\bar{a},\nu''} t_2'$, and $\nu' \asymp \nu''$. By induction, $t_1 \xrightarrow{a,\nu'\cdot\lambda_?} t_1'$ and $t_2 \xrightarrow{\bar{a},\nu''\cdot\lambda_?} t_2'$. Since $(\nu' \cdot \lambda_?) \asymp (\nu'' \cdot \lambda_?)$, then by SOS role *Sync*, we have $t \xrightarrow{\tau,(\nu'\odot\nu'')\cdot\lambda_?} t_1' \| t_2'$. The same argument holds when $t \xrightarrow{a,\nu} t_1' \| t_2$ or $t \xrightarrow{a,\nu} t_1 \| t_2'$ by SOS rule *Par* as the results of $t_1 \xrightarrow{a,\nu'} t_1'$ or $t_2 \xrightarrow{a,\nu'} t_2'$, respectively. □

**Lemma 25.** *For each PL-CCS term $t$, $t \xrightarrow{a,\nu} t'$ implies $t \xrightarrow{a,\nu'} t$, where $\nu = \nu' \cdot \lambda_?$ and $|\nu'| \geq max(bi(t))$.*

**Proof.** By induction on the structure of $t$. The only interesting cases are:

- $t \equiv t_1 \oplus_i t_2$: by SOS rule *Select*, we have $t \xrightarrow{a,\nu} t'_1$, since $t_1 \xrightarrow{a,\nu'} t'$ and $\nu'|_i \neq R$, and $\nu \equiv \nu'|_{i/L}$. By induction, we obtain $t_1 \xrightarrow{a,\nu''} t'$, where $\nu' \equiv \nu'' \cdot \lambda_?$ and $|\nu''| \geq max(bi(t))$. Since $i \in bi(t)$, then $i < |\nu''|$, and consequently $\nu''|_i \neq R$. Therefore by SOS rule *Select*, $t \xrightarrow{a,\nu''|_{i/L}} t'_1$ holds. The same argument holds when $t \xrightarrow{a,\nu} t'_2$ as the result of $t_2 \xrightarrow{a,\nu'} t'_2$.
- $t \equiv t_1 \parallel t_2$: by SOS rule *Sync*, we obtain $t \xrightarrow{\tau,\nu_1 \odot \nu_2} t'_1 \parallel t'_2$, since $t_1 \xrightarrow{a,\nu_1} t'_1$ and $t_2 \xrightarrow{\overline{a},\nu_2} t'_2$, and $\nu_1 \asymp \nu_2$. By induction, $t_1 \xrightarrow{a,\nu'_1} t'_1$ and $t_2 \xrightarrow{\overline{a},\nu'_2} t'_2$, where $\nu_1 = \nu'_1 \cdot \lambda_?$, $\nu_2 = \nu'_2 \cdot \lambda_?$, and $|\nu'_1|, |\nu'_2| \geq max(bi(t))$. Therefore, $\nu_1 \asymp \nu_2$ implies $\nu'_1 \asymp \nu'_2$, and by SOS rule *Sync*, we have that $t \xrightarrow{a,\nu'_1 \odot \nu'_2} t'_1 \parallel t'_2$. The same argument holds when $t \xrightarrow{a,\nu} t'_1 \parallel t_2$ or $t \xrightarrow{a,\nu} t_1 \parallel t'_2$ by SOS rule *Par* as the results of $t_1 \xrightarrow{a,\nu'} t'_1$ or $t_2 \xrightarrow{a,\nu'} t'_2$, respectively. □

We now proceed with the proof of Theorem 11. Following Definition 5, $s \simeq_{PL} t$ implies that for any full configuration $\nu_1^f$ with respect to $s$, there exists a valid full configuration $\nu_2^f$ with respect to $t$ such that $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$ holds. In the remainder of the proof, we assume that $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$ is witnessed by strong bisimulation relation $\mathcal{R}$.

**Case 1**. We have that $\nu_1^f \in VFConfig(s)$ and $\nu_2^f \in VFConfig(t)$ and hence, $\nu_1^f \in VFConfig(a.s)$ and $\nu_2^f \in VFConfig(a.t)$. Therefore, we only need to prove that $\Pi(a.s, \nu_1^f) \sim \Pi(a.t, \nu_2^f)$. To this end, we prove that the closure of $\mathcal{R}$ with action prefixing, denoted by $\mathcal{R}'$ is a strong bisimulation relation. We formally define $\mathcal{R}'$ as $\mathcal{R} \cup \{(\Pi(a.s, \nu_1^f), \Pi(a.t, \nu_2^f)) \mid (\Pi(s, \nu_1^f), \Pi(t, \nu_2^f)) \in \mathcal{R}\}$. It remains to show the transfer conditions of Definition 3 for each pair in $\mathcal{R}'$. The case for the pairs in $\mathcal{R}$ holds vacuously. We only need to show the transfer conditions for an arbitrary pair $(\Pi(a.s, \nu_1^f), \Pi(a.t, \nu_2^f)) \in \mathcal{R}'$. Assume that $\Pi(a.s, \nu_1^f) \xrightarrow{a} (s', \nu_1^f)$ for some $s'$. This transition can only be due to SOS rule *Prefix* and $s'$ must be $s$. Hence, we have that $a.s \xrightarrow{a,\nu_?} s$ and $\nu_? \sqsubseteq \nu_1^f$. Similarly, it follows from SOS rule *Prefix* and the definition of the LTS semantic of product lines that $\Pi(a.t, \nu_2^f) \xrightarrow{a} \Pi(t, \nu_2^f)$. Due to the definition of $\mathcal{R}'$, we have that $(\Pi(s, \nu_1^f), \Pi(t, \nu_2^f)) \in \mathcal{R}$ and hence $(\Pi(s, \nu_1^f), \Pi(t, \nu_2^f)) \in \mathcal{R}'$, which was to be shown.

**Case 3**. Following the Definition 5, we prove that for any valid full configuration ${\nu^f}'_1$ with respect to $s \oplus_i r$, there exists a valid full configuration ${\nu^f}'_2$ with respect to $t \oplus_i r$ such that $\Pi(s \oplus_i r, {\nu^f}'_1) \sim \Pi(t \oplus_i r, {\nu^f}'_2)$ holds. Since $s \oplus_i r$ and $t \oplus_i r$ are fully expanded, then $i$ is fresh in $s$, $t$, and $r$. Regarding the value of ${\nu^f}'_1|_i$, two cases can be considered:

1. ${\nu^f}'_1|_i = L$: Let ${\nu^f}'_1 = \nu_1^f \cdot \lambda_1$, then take ${\nu^f}'_2 = \nu_1^f \cdot \lambda_2$ such that $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$ and ${\nu^f}'_1|_i = {\nu^f}'_2|_i$. Construct $\mathcal{R}'_1$ as:

$$\mathcal{R}'_1 = \{(\Pi(s \oplus_i r, {\nu^f}'_1), \Pi(t \oplus_i r, {\nu^f}'_2))\} \cup \{(\Pi(s', {\nu^f}'_1), \Pi(t', {\nu^f}'_2)) | (\Pi(s', \nu_1^f), \Pi(t', \nu_2^f)) \in \mathcal{R}\}.$$

   We prove that $\mathcal{R}'_1$ satisfies the transfer conditions of Definition 3. We only examine the transfer conditions for an arbitrary pair $(\Pi(s \oplus_i r, {\nu^f}'_1), \Pi(t \oplus_i r, {\nu^f}'_2)) \in \mathcal{R}'_1$ as the pairs in $\mathcal{R}$ trivially satisfy the transfer conditions. Suppose that $\Pi(s \oplus_i r, {\nu^f}'_1) \xrightarrow{a} p$. This transition can only be due to SOS rule *Select* and transition $s \xrightarrow{a,\nu} s'$, where $\nu \sqsubseteq {\nu^f}'_1$, and $p \equiv \Pi(s', {\nu^f}'_1)$. Hence, we have that $s \oplus_i r \xrightarrow{a,\nu|_{i/L}} s'$. By Lemma 25, $s \xrightarrow{a,\nu_s} s'$, where $\nu = \nu_s \cdot \lambda_?$, $\nu_s \sqsubseteq \nu_1^f$ and $\nu_s|_i =?$. Therefore, it follows from the definition of the LTS semantic of product lines that $\Pi(s, \nu_1^f) \xrightarrow{a} \Pi(s', \nu_1^f)$. Furthermore, $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_2^f)$ implies that $\Pi(t, \nu_2^f) \xrightarrow{a} \Pi(t', \nu_2^f)$, and $\Pi(s', \nu_1^f)\ \mathcal{R}\ \Pi(t', \nu_2^f)$. Similarly, it follows from the definition of the LTS semantic of product lines that $t \xrightarrow{a,\nu_t} t'$, where $\nu_t \sqsubseteq \nu_2^f$, $\nu_t|_i =?$. Thus, by Lemma 24, $t \xrightarrow{a,\nu'} t'$, where $\nu' = \nu_t \cdot \lambda_?$ and $\nu' \sqsubseteq {\nu^f}'_2$. From SOS rule *Select* and the definition of the LTS semantic of product lines, it follows that $\Pi(t \oplus_i r, {\nu^f}'_2) \xrightarrow{a} \Pi(t', {\nu^f}'_2)$. Due to the definition of $\mathcal{R}'$, we have that $(\Pi(s', \nu_1^f), \Pi(t', \nu_2^f)) \in \mathcal{R}$ and hence $(\Pi(s', \nu_1^f), \Pi(t', \nu_2^f)) \in \mathcal{R}'$, which was to be shown. The same discussion holds when $\Pi(t \oplus_i r, {\nu^f}'_2) \xrightarrow{a} p$. Concluding that $\mathcal{R}'_1$ is a strong bisimulation.
2. ${\nu^f}'_1|_i = R$: Let ${\nu^f}'_1 = \nu^f \cdot \lambda_1$, where $\nu^f$ is a valid configured configuration with respect to $r$, then take ${\nu^f}'_2 = \nu^f \cdot \lambda_2$ such that ${\nu^f}'_1|_i = {\nu^f}'_2|_i$. Construct $\mathcal{R}'_2$ as:

$$\mathcal{R}'_2 = \{(\Pi(s \oplus_i r, {\nu^f}'_1), \Pi(t \oplus_i r, {\nu^f}'_2))\} \cup \{(\Pi(r', {\nu^f}'_1), \Pi(r', {\nu^f}'_2))\}.$$

We prove that $\mathcal{R}'_2$ satisfies the transfer conditions of Definition 3. We only examine the transfer conditions for an arbitrary pair $(\Pi(s \oplus_i r, \nu^{f'}_1), \Pi(t \oplus_i r, \nu^{f'}_2)) \in \mathcal{R}'_2$ as the pairs in $\{(\Pi(r', \nu^{f'}_1), \Pi(r', \nu^{f'}_2))\}$ trivially satisfy the transfer conditions. Suppose that $\Pi(s \oplus_i r, \nu^{f'}_1) \xrightarrow{a} p$. This transition can only be due to SOS rule *Select* and transition $r \xrightarrow{a,\nu} r'$, where $\nu \sqsubseteq \nu^{f'}_1$, and $p \equiv \Pi(r', \nu^{f'}_1)$. Hence, we have that $s \oplus_i r \xrightarrow{a,\nu|_{i/R}} r'$. By Lemma 25, $r \xrightarrow{a,\nu_r} r'$, where $\nu = \nu_r \cdot \lambda_?$ and $\nu_r \sqsubseteq \nu^f$. Hence, by Lemma 24, $r \xrightarrow{a,\nu'} r'$, where $\nu' = \nu_r \cdot \lambda_?$, and $\nu' \sqsubseteq \nu^{f'}_2$. From SOS rule *Select* and the definition of the LTS semantic of product lines, it follows that $\Pi(t \oplus_i r, \nu^{f'}_2) \xrightarrow{a} \Pi(r', \nu^{f'}_2)$, and by the definition of $\mathcal{R}'_2$ we have that $\Pi(r', \nu^{f'}_1)\ \mathcal{R}'_2\ \Pi(r', \nu^{f'}_2)$, which was to be shown. The same discussion holds when $\Pi(t \oplus_i r, \nu^{f'}_2) \xrightarrow{a} p$. Concluding that $\mathcal{R}'_2$ is a strong bisimulation.

The same discussion holds for any valid full configuration $\nu^{f'}_2$ with respect to $t \oplus_i r$. We conclude that $s \oplus_i r \simeq_{PL} t \oplus_i r$.

**Case 6**. Since $s \parallel r$ and $t \parallel r$ are fully expanded and the root of their parse tree is parallel composition, $s$ and $t$ cannot have any binary variant in common with $r$, i.e., $bi(s) \cap bi(r) = \emptyset$ and $bi(t) \cap bi(r) = \emptyset$. Consequently, for any $\nu^f_1 \in VFConfig(s)$, there exists a $\nu^f_2 \in VFConfig(t)$ such that $\forall i \in bi(r)(\nu^f_1|_i = \nu^f_2|_i)$. Following the Definition 5, we prove that for any $\nu^{f'}_1 \in VFConfig(s \parallel r)$, there exists $\nu^{f'}_2 \in VFConfig(t \parallel r)$ such that $\Pi(s \parallel r, \nu^{f'}_1) \sim \Pi(t \parallel r, \nu^{f'}_2)$ holds. To this aim, for any $\nu^{f'}_1 = \nu^f_1 \cdot \lambda_1$, take $\nu^{f'}_2 = \nu^f_2 \cdot \lambda_2$ such that $\Pi(s, \nu^f_1) \sim \Pi(t, \nu^f_1)$ and $\forall i \in bi(r)(\nu^f_1|_i = \nu^f_2|_i)$. Construct $\mathcal{R}'$ as

$$\mathcal{R}' = \{(\Pi(s' \parallel r', \nu^{f'}_1), \Pi(t' \parallel r', \nu^{f'}_2)) | (\Pi(s', \nu^f_1), \Pi(t', \nu^f_2)) \in \mathcal{R}\}$$

We show that it satisfies the transfer conditions of Definition 3.

For an arbitrary pair $\Pi(s' \parallel r', \nu^{f'}_1)\ \mathcal{R}'\ \Pi(t' \parallel r', \nu^{f'}_1)$, suppose that $\Pi(s' \parallel r', \nu^{f'}_1) \xrightarrow{a} p$. Using the SOS rules *Par* and *Sync*, three cases can be considered:

1. This transition can be due to SOS rule *Par* and $s' \xrightarrow{a,\nu'_1} s''$, where $\nu'_1 \sqsubseteq \nu^{f'}_1$. Hence, we have that $s' \parallel r' \xrightarrow{a,\nu'_1} s'' \parallel r'$, and $p \equiv \Pi(s'' \parallel r', \nu^{f'}_1)$. By Lemma 25, $s' \xrightarrow{a,\nu_s} s''$, where $\nu'_1 = \nu_s \cdot \lambda_?$ and $\nu_s \sqsubseteq \nu^f_1$. Thus, it follows from the definition of the LTS semantic of product lines that $\Pi(s', \nu^f_1) \xrightarrow{a} \Pi(s'', \nu^f_1)$. Furthermore, $\Pi(s', \nu^f_1)\ \mathcal{R}\ \Pi(t', \nu^f_2)$ implies that $\Pi(t', \nu^f_2) \xrightarrow{a} \Pi(t'', \nu^f_2)$ and $\Pi(s'', \nu^f_1)\ \mathcal{R}\ \Pi(t'', \nu^f_2)$. Similarly, it follows from the definition of the LTS semantic of product lines that $t' \xrightarrow{a,\nu_t} t''$, where $\nu_t \sqsubseteq \nu^f_2$ and consequently by Lemma 24, $t' \xrightarrow{a,\nu'_2} t''$, where $\nu'_2 = \nu_t \cdot \lambda_?$ and $\nu'_2 \sqsubseteq \nu^{f'}_2$. By SOS rule *Par*, $t' \parallel r' \xrightarrow{a,\nu'_2} t'' \parallel r'$ and consequently, $\Pi(t' \parallel r', \nu^{f'}_2) \xrightarrow{a} \Pi(t'' \parallel r', \nu^{f'}_2)$. Due to the definition of $\mathcal{R}'$ we have that $\Pi(s'', \nu^f_1)\ \mathcal{R}\ \Pi(t'', \nu^f_2)$, and hence $\Pi(s'' \parallel r', \nu^{f'}_1)\ \mathcal{R}'\ \Pi(t'' \parallel r', \nu^{f'}_2)$, which was to be shown.
2. This transition can be due to SOS rule *Par* and $r' \xrightarrow{a,\nu'} r''$, where $\nu' \sqsubseteq \nu^{f'}_1$. Hence, we have that $s' \parallel r' \xrightarrow{a,\nu'} s' \parallel r''$ and $p \equiv \Pi(s' \parallel r'', \nu^{f'}_1)$. Therefore, by Lemmas 25 and 24, $r' \xrightarrow{a,\nu''} r''$, where $\nu'' \sqsubseteq \nu^{f'}_2$. Thus, by SOS rule *Par*, $t' \parallel r' \xrightarrow{a,\nu''} t' \parallel r''$, where $\nu'' \sqsubseteq \nu^{f'}_2$, and consequently, $\Pi(t' \parallel r', \nu^{f'}_2) \xrightarrow{a} \Pi(t' \parallel r'', \nu^{f'}_2)$. Due to the definition of $\mathcal{R}'$ we have that $\Pi(s', \nu^f_1)\ \mathcal{R}\ \Pi(t', \nu^f_2)$, and hence $\Pi(s' \parallel r'', \nu^{f'}_1)\ \mathcal{R}'\ \Pi(t' \parallel r'', \nu^{f'}_2)$, which was to be shown.
3. This transition can be due to SOS rule *Sync* and transitions $s' \xrightarrow{a,\nu'_1} s''$ and $r' \xrightarrow{\bar{a},\nu'} r''$, where $\nu'_1 \sqsubseteq \nu^{f'}_1$, $\nu' \sqsubseteq \nu^{f'}_1$, and $\nu'_1 \asymp \nu'$, and consequently $p \equiv \Pi(s'' \parallel r'', \nu^{f'}_1)$. Hence, we have that $s' \parallel r' \xrightarrow{\tau,\nu'_1 \odot \nu'} s'' \parallel r''$. By Lemma 25, $s' \xrightarrow{a,\nu_s} s''$, where $\nu'_1 = \nu_s \cdot \lambda_?$ and $\nu_s \sqsubseteq \nu^f_1$. It follows from the definition of the LTS semantic of product lines that $\Pi(s', \nu^f_1) \xrightarrow{a} \Pi(s'', \nu^f_1)$. Furthermore, $\Pi(s', \nu^f_1)\ \mathcal{R}\ \Pi(t', \nu^f_2)$ implies that $\Pi(t', \nu^f_2) \xrightarrow{a} \Pi(t'', \nu^f_2)$ and $\Pi(s'', \nu^f_1)\ \mathcal{R}\ \Pi(t'', \nu^f_2)$. It follows that $t' \xrightarrow{a,\nu_t} t''$, where $\nu_t \sqsubseteq \nu^f_2$, and consequently, by Lemma 24, $t' \xrightarrow{a,\nu'_2} t''$, where $\nu'_2 = \nu_t \cdot \lambda_?$ and $\nu'_2 \sqsubseteq \nu^{f'}_2$. Since $\nu_s \sqsubseteq \nu^f_1$ and $\nu_t \sqsubseteq \nu^f_2$, then $\nu'_1 \asymp \nu'$ and $\forall i \in bi(r)(\nu^f_1|_i = \nu^f_2|_i)$ imply that $\nu'_2 \asymp \nu'$ and $\nu'_2 \odot \nu' \sqsubseteq \nu^{f'}_2$. By SOS rule *Sync*, we have that $t' \parallel r' \xrightarrow{\tau,\nu'_2 \odot \nu'} t'' \parallel r''$ and hence, $\Pi(t' \parallel r', \nu^{f'}_2) \xrightarrow{\tau} \Pi(t'' \parallel r'', \nu^{f'}_2)$. Due to the Definition of $\mathcal{R}'$, we have that $\Pi(s'', \nu^f_1)\ \mathcal{R}\ \Pi(t'', \nu^f_2)$ and hence, $\Pi(s'' \parallel r'', \nu^{f'}_1)\ \mathcal{R}'\ \Pi(t'' \parallel r'', \nu^{f'}_2)$.

The same discussion holds when $\Pi(t' \parallel r', \nu^{f'}_2) \xrightarrow{a} p$. Concluding that $\mathcal{R}'$ is a strong bisimulation, and consequently $s \parallel r \simeq_{PL} t \parallel r$.

### A.3. Proof of Theorem 9

**Theorem 9** *(Part 1). Strict strong bisimilarity is an equivalence relation.*

**Proof.** To show that strict strong bisimilarity is an equivalence, we must show that it is reflexive, symmetric, and transitive. Reflexivity and symmetry follow immediately from the reflexivity and symmetry properties of strong bisimilarity. Hence, it only remains to prove transitivity.

Consider PL-CCS terms $s$, $t$, and $r$ such that $s \approx_{PL} t$, and $t \approx_{PL} r$; following Definition 4 for any valid full configuration $\nu_1^f \in \mathit{VFConfig}(s) \cap \mathit{VFConfig}(t)$ and $\nu_2^f \in \mathit{VFConfig}(t) \cap \mathit{VFConfig}(r)$, it holds that $\Pi(s, \nu_1^f) \sim \Pi(t, \nu_1^f)$, and $\Pi(t, \nu_2^f) \sim \Pi(r, \nu_2^f)$. Without loss of generality, we assume that $max(bi(s)) \geq max(bi(t)) \geq max(bi(r))$ (the proof of all other cases is almost identical). Therefore, $\nu_1^f = \nu_r \cdot \lambda_1 \cdot \lambda_2$, $\nu_2^f = \nu_r \cdot \lambda_2$, where $\nu_r \in \mathit{VFConfig}(r)$. We prove that $\Pi(s, \nu_1^f) \sim \Pi(r, \nu_1^f)$. Construct $\mathcal{R}$ as

$$\mathcal{R} = \{(\Pi(s', \nu_1^f), \Pi(r', \nu_1^f)) \mid \Pi(s', \nu_1^f) \sim \Pi(t', \nu_1^f) \wedge \Pi(t', \nu_2^f) \sim \Pi(r', \nu_2^f)\}$$

We prove that $\mathcal{R}$ satisfies the transfer conditions of Definition 3.

For an arbitrary pair $(\Pi(s', \nu_1^f), \Pi(r', \nu_1^f)) \in \mathcal{R}$, consider that $\Pi(s', \nu_1^f) \xrightarrow{\alpha} \Pi(s'', \nu_1^f)$ since $s' \xrightarrow{\alpha, \nu_1'} s''$, where $\nu_1' \sqsubseteq \nu_1^f$. Hence, $\Pi(s', \nu_1^f) \sim \Pi(t', \nu_1^f)$ implies that $\Pi(t', \nu_1^f) \xrightarrow{\alpha} \Pi(t'', \nu_1^f)$ and $\Pi(s'', \nu_1^f) \sim \Pi(t'', \nu_1^f)$. It follows that $t' \xrightarrow{\alpha, \nu_2} t''$, where $\nu_2 \sqsubseteq \nu_1^f$. By Lemma 25, it holds that $t' \xrightarrow{\alpha, \nu_2'} t''$, where $\nu_2 = \nu_2' \cdot \lambda_?$ and $\nu_2' \sqsubseteq \nu_2^f$ and consequently $\Pi(t', \nu_2^f) \xrightarrow{\alpha} \Pi(t'', \nu_2^f)$. Therefore, $\Pi(t', \nu_2^f) \sim \Pi(r', \nu_2^f)$ implies that $\Pi(r', \nu_2^f) \xrightarrow{\alpha} \Pi(r'', \nu_2^f)$ and $\Pi(t'', \nu_2^f) \sim \Pi(r'', \nu_2^f)$. It follows that $r' \xrightarrow{\alpha, \nu_3'} r''$, where $\nu_3' \sqsubseteq \nu_2^f$. Consequently, by Lemma 24, $r' \xrightarrow{\alpha, \nu_3} r''$, where $\nu_3 = \nu_3' \cdot \lambda_?$ and $\nu_3 \sqsubseteq \nu_1^f$. Hence, $\Pi(r', \nu_1^f) \xrightarrow{\alpha} \Pi(r'', \nu_1^f)$. Due to the Definition of $\mathcal{R}$, we have that $\Pi(s'', \nu_1^f) \sim \Pi(t'', \nu_1^f)$ and $\Pi(t'', \nu_2^f) \sim \Pi(r'', \nu_2^f)$ and hence, $(\Pi(s'', \nu_1^f), \Pi(r'', \nu_1^f)) \in \mathcal{R}$, which was to be shown. The same discussion holds when $r' \xrightarrow{\alpha, \nu_3} r''$, where $\nu_3 \sqsubseteq \nu_1^f$. Concluding that $\mathcal{R}$ is a strong bisimulation, and consequently $s \approx_{PL} r$. □

**Theorem 9** *(Part 2). Strict strong bisimilarity is congruence on PL-CCS terms.*

**Proof.** To show that strict strong bisimilarity is a congruence on PL-CCS terms with respect to the PL-CCS operators, we need to prove that for any arbitrary product line $r$, renaming function $\phi$, and $L \subseteq Act$, $s \approx_{PL} t$ implies following cases:

1. $\alpha.s \approx_{PL} \alpha.t$;
2. $s + r \approx_{PL} t + r$;
3. $s \oplus_i r \approx_{PL} t \oplus_i r$;
4. $s \setminus L \approx_{PL} t \setminus L$;
5. $s[\phi] \approx_{PL} t[\phi]$;
6. $s \parallel r \approx_{PL} t \parallel r$;

We only prove cases 1, 3, and 6; the proof of remaining cases is almost identical. Note that $\forall \nu^{f'} \in \mathit{VFConfig}(s \odot r) \cap \mathit{VFConfig}(t \odot r) \Rightarrow \exists \nu^f \in \mathit{VFConfig}(s) \cap \mathit{VFConfig}(t)(\nu^{f'} = \nu^f \cdot \lambda)$, where $\odot \in \{+, \oplus, \parallel, \between, |\}$. For any valid full configuration $\nu^f$ with respect to $s$ and $t$, we assume $\Pi(s, \nu^f) \sim \Pi(t, \nu^f)$ is witnessed by strong bisimulation $\mathcal{R}$.

**Case 1**. We have that $\nu^f \in \mathit{VFConfig}(s) \cap \mathit{VFConfig}(t)$, and hence, $\nu^f \in \mathit{VFConfig}(a.s) \cap \mathit{VFConfig}(a.t)$. Therefore, we only need to prove that $\Pi(a.s, \nu^f) \sim \Pi(a.t, \nu^f)$. To this end, we prove that the closure of $\mathcal{R}$ with action prefixing, denoted by $\mathcal{R}'$ is a strong bisimulation relation. It remains to show the transfer conditions of Definition 3 for each pair in $\mathcal{R}'$. The case for the pairs in $\mathcal{R}$ holds vacuously. We only need to show the transfer conditions for an arbitrary pair $(\Pi(a.s, \nu_1^f), \Pi(a.t, \nu_2^f)) \in \mathcal{R}'$. Assume that $\Pi(a.s, \nu^f) \xrightarrow{a} (s', \nu^f)$ for some $s'$. This transition can only be due to SOS rule *Prefix* and $s'$ must be $s$. Hence, we have that $a.s \xrightarrow{a, \nu_?} s$ and $\nu_? \sqsubseteq \nu^f$. Similarly, it follows from SOS rule *Prefix* and the definition of the LTS semantic of product lines that $\Pi(a.t, \nu^f) \xrightarrow{a} \Pi(t, \nu^f)$. Due to the definition of $\mathcal{R}'$, we have that $(\Pi(s, \nu^f), \Pi(t, \nu^f)) \in \mathcal{R}$ and hence $(\Pi(s, \nu^f), \Pi(t, \nu^f)) \in \mathcal{R}'$, which was to be shown.

**Case 3**. Following the Definition 4, we prove that for any valid full configuration $\nu^{f'}$ with respect to $s \oplus_i r$ and $t \oplus_i r$, $\Pi(s \oplus_i r, \nu^{f'}) \sim \Pi(t \oplus_i r, \nu^{f'})$ holds. Construct $\mathcal{R}'$ as:

$$\mathcal{R}' = \{(\Pi(s \oplus_i r, \nu^{f'}), \Pi(t \oplus_i r, \nu^{f'}))\} \cup \{(\Pi(t', \nu^{f'}), \Pi(t', \nu^{f'}))\} \cup \{(\Pi(s', \nu^{f'}), \Pi(t', \nu^{f'})) \mid (\Pi(s', \nu^f), \Pi(t', \nu^f)) \in \mathcal{R}\}.$$

We prove that $\mathcal{R}'$ satisfies the transfer conditions of Definition 3. We only prove the transfer conditions for the pair $(\Pi(s \oplus_i r, \nu^{f'}), \Pi(t \oplus_i r, \nu^{f'})) \in \mathcal{R}'$ as for others, the proof is trivial (the proof for pairs such as $(\Pi(s', \nu^{f'}), \Pi(t', \nu^{f'})) \in \mathcal{R}'$ follows from Lemmas 24 and 25).

Suppose that $\Pi(s \oplus_i r, \nu^{f'}) \xrightarrow{a} p$. Regarding the value of $\nu^{f'}|_i$ two cases can be considered:

1. $\nu^{f'}|_i = L$: This transition can be due to SOS rule *Select* and $s \xrightarrow{a,\nu} s'$, where $\nu \sqsubseteq \nu^{f'}$, and $p \equiv \Pi(s', \nu^{f'})$. Hence, we have that $s \oplus_i r \xrightarrow{a,\nu|_{i/L}} s'$. By Lemma 25, $s \xrightarrow{a,\nu_s} s'$, where $\nu = \nu_s \cdot \lambda_?$, and $\nu_s \sqsubseteq \nu^f$. Therefore, from the definition of the LTS semantic of product lines it follows that $\Pi(s, \nu^f) \xrightarrow{a} \Pi(s', \nu^f)$. Furthermore, $\Pi(s, \nu^f) \sim \Pi(t, \nu^f)$ implies that $\Pi(t, \nu^f) \xrightarrow{a} \Pi(t', \nu^f)$ and $\Pi(s', \nu^f) \sim \Pi(t', \nu^f)$. Similarly, it follows that $t \xrightarrow{a,\nu_t} t'$, where $\nu_t \sqsubseteq \nu^f$. Thus, by Lemma 24, $t \xrightarrow{a,\nu'} t'$, where $\nu' = \nu_t \cdot \lambda_?$, and $\nu' \sqsubseteq \nu^{f'}$. By *Select*, $t \oplus_i r \xrightarrow{a,\nu'|_{i/L}} t'$ and consequently $\Pi(t \oplus_i r, \nu^{f'}) \xrightarrow{a} \Pi(t', \nu^{f'})$. Due to the definition of $\mathcal{R}'$, we have that $\Pi(s', \nu^f) \sim \Pi(t', \nu^f)$ and hence, $\Pi(s', \nu^{f'}) \mathcal{R}' \Pi(t', \nu^{f'})$, which was to be shown.
2. $\nu^f|_i = R$: This transition can be due to SOS rule *Select* and $r \xrightarrow{a,\nu} r'$, where $\nu \sqsubseteq \nu^{f'}$, and $p \equiv \Pi(r', \nu^{f'})$. Hence, we have that $s \oplus_i r \xrightarrow{a,\nu|_{i/R}} r'$ and similarly, $t \oplus_i r \xrightarrow{a,\nu|_{i/R}} t'$. It follows from the definition of the LTS semantic of product lines that $\Pi(t \oplus_i r, \nu^{f'}) \xrightarrow{a} \Pi(r', \nu^{f'})$. By the definition of $\mathcal{R}'$, $\Pi(r', \nu^{f'}) \mathcal{R}' \Pi(r', \nu^{f'})$.

The same discussion holds when $\Pi(t \oplus_i r, \nu^{f'}) \xrightarrow{a} p$. Concluding that $\mathcal{R}'$ is a strong bisimulation. Consequently $s \oplus_i r \approx_{PL} t \oplus_i r$.

**Case 6**. Following the Definition 4, we prove that for any valid full configuration $\nu^{f'}$ with respect to $s \parallel r$ and $t \parallel r$, $\Pi(s \parallel r, \nu^{f'}) \sim \Pi(t \parallel r, \nu^{f'})$ holds. Construct $\mathcal{R}'$ as

$$\mathcal{R}' = \{(\Pi(s' \parallel r', \nu^{f'}), \Pi(t' \parallel r', \nu^{f'}))|(\Pi(s', \nu^f), \Pi(t', \nu^f)) \in \mathcal{R}\}$$

We show that it satisfies the transfer conditions of Definition 3.

For an arbitrary pair $\Pi(s' \parallel r', \nu^{f'}) \mathcal{R}' \Pi(t' \parallel r', \nu^{f'})$, suppose that $\Pi(s' \parallel r', \nu^{f'}) \xrightarrow{a} p$. Using the SOS rules *Par* and *Sync*, three cases can be considered:

1. This transition can be due to SOS rule *Par* and $s' \xrightarrow{a,\nu'_1} s''$, where $\nu'_1 \sqsubseteq \nu^{f'}$, and $p \equiv \Pi(s'' \parallel r', \nu^{f'})$. Hence, we have that $s' \parallel r' \xrightarrow{a,\nu'_1} s'' \parallel r'$. By Lemma 25, $s' \xrightarrow{a,\nu_s} s''$, where $\nu'_1 = \nu_s \cdot \lambda_?$ and $\nu_s \sqsubseteq \nu^f$. It follows from the definition of the LTS semantic of product lines that $\Pi(s', \nu^f) \xrightarrow{a} \Pi(s'', \nu^f)$. Furthermore, $\Pi(s', \nu^f) \mathcal{R} \Pi(t', \nu^f)$ implies that $\Pi(t', \nu^f) \xrightarrow{a} \Pi(t'', \nu^f)$ and $\Pi(s'', \nu^f) \mathcal{R} \Pi(t'', \nu^f)$. It follows that $t' \xrightarrow{a,\nu_t} t''$, where $\nu_t \sqsubseteq \nu^f$, and consequently by Lemma 24, $t' \xrightarrow{a,\nu'_2} t''$, where $\nu'_2 = \nu_t \cdot \lambda_?$ and $\nu'_2 \sqsubseteq \nu^{f'}$. By SOS rule *Par*, $t' \parallel r' \xrightarrow{a,\nu'_2} t'' \parallel r'$ and hence, $\Pi(t' \parallel r', \nu^{f'}) \xrightarrow{a} \Pi(t'' \parallel r', \nu^{f'})$. Due to the Definition of $\mathcal{R}'$, we have that $\Pi(s'', \nu^f) \mathcal{R} \Pi(t'', \nu^f)$ and hence $\Pi(s'' \parallel r', \nu^{f'}) \mathcal{R}' \Pi(t'' \parallel r', \nu^{f'})$, which was to be shown.
2. This transition can be due to SOS rule *Par* and $r' \xrightarrow{a,\nu'} r''$, where $\nu' \sqsubseteq \nu^{f'}$, and $p \equiv \Pi(s' \parallel r'', \nu^{f'})$. Hence, we have that $s' \parallel r' \xrightarrow{a,\nu'} s' \parallel r''$ and similarly, $t' \parallel r' \xrightarrow{a,\nu'} t' \parallel r''$, where $\nu' \sqsubseteq \nu^{f'}$. Therefore, it follows that $\Pi(t' \parallel r', \nu^{f'}) \xrightarrow{a} \Pi(t' \parallel r'', \nu^{f'})$. Due to the Definition of $\mathcal{R}'$, we have that $\Pi(s' \parallel r'', \nu^{f'}) \mathcal{R}' \Pi(t' \parallel r', \nu^{f''})$, which was to be shown.
3. This transition can be due to SOS rule *Sync* and transitions $s' \xrightarrow{a,\nu'_1} s''$ and $r' \xrightarrow{\overline{a},\nu'} r''$, where $\nu'_1 \sqsubseteq \nu^{f'}$, $\nu' \sqsubseteq \nu^{f'}$, and $\nu'_1 \asymp \nu'$, and consequently $p \equiv \Pi(s'' \parallel r'', \nu^{f'})$. Hence, we have that $s' \parallel r' \xrightarrow{\tau,\nu'_1 \odot \nu'} s'' \parallel r''$. By Lemma 25, $s' \xrightarrow{a,\nu_s} s''$, where $\nu'_1 = \nu_s \cdot \lambda_?$ and $\nu_s \sqsubseteq \nu^f$. It follows from the definition of the LTS semantic of product lines that $\Pi(s', \nu^f) \xrightarrow{a} \Pi(s'', \nu^f)$. Furthermore, $\Pi(s', \nu^f) \mathcal{R} \Pi(t', \nu^f)$ implies that $\Pi(t', \nu^f) \xrightarrow{a} \Pi(t'', \nu^f)$ and $\Pi(s'', \nu^f) \mathcal{R} \Pi(t'', \nu^f)$. It follows that $t' \xrightarrow{a,\nu_t} t''$, where $\nu_t \sqsubseteq \nu^f$, and consequently, by Lemma 24, $t' \xrightarrow{a,\nu'_2} t''$, where $\nu'_2 = \nu_t \cdot \lambda_?$ and $\nu'_2 \sqsubseteq \nu^{f'}$. Since $\nu_s \sqsubseteq \nu^f$ and $\nu_t \sqsubseteq \nu^f$, then $\nu_s \asymp \nu_t$. Therefore, $\nu'_1 \asymp \nu'$ implies $\nu'_2 \asymp \nu'$ and $\nu'_2 \odot \nu' \sqsubseteq \nu^{f'}$. By SOS rule *Sync*, we have that $t' \parallel r' \xrightarrow{\tau,\nu'_2 \odot \nu'} t'' \parallel r''$ and hence, $\Pi(t' \parallel r', \nu^{f'}) \xrightarrow{\tau} \Pi(t'' \parallel r'', \nu^{f'})$. Due to the Definition of $\mathcal{R}'$, we have that $\Pi(s'', \nu^f) \mathcal{R} \Pi(t'', \nu^f)$ and hence, $\Pi(s'' \parallel r'', \nu^{f'}) \mathcal{R}' \Pi(t'' \parallel r'', \nu^{f'})$.

The same discussion holds when $\Pi(t' \parallel r', \nu^f)' \xrightarrow{a} p$. Concluding that $\mathcal{R}'$ is a strong bisimulation. □

## Appendix B. Proof of Theorems 12 and 19

In this section, we first prove Theorem 12 and then we provide a proof for Theorem 19. To this aim, we prove that each PL-CCS term can be rewritten by axioms $A_{4-6}$, $D_{1,2}$, $P_{4,5}$, $S_{1,3}$, $R_3$, $E_5$, *Dri*, and $N_{1-5}$ into a form where no binary variant occurs in the scope of a CCS-operator.

We prove this by structural induction on the syntax of term $t$. The base case of the induction, where $t \equiv 0$, holds trivially. We distinguish the following cases based on the structure of $t$:

- if $t \equiv a.t'$, then $t$ can be rewritten into the required form by applying the induction hypothesis on $t'$, and subsequently applying axiom $A_4$.

- if $t \equiv t'_1 + t'_2$, then $t$ can be rewritten into the required format by applying the induction hypothesis on $t'_1$ and $t'_2$, and then applying axioms $A_{5,6}$. Assume that $\vdash t'_1 = p_1 \oplus_i p_2$ and $\vdash t'_2 = q_1 \oplus_j q_2$. By axioms $A_{5,6} \vdash t = ((p_1 + q_1) \oplus_j (p_1 + q_2)) \oplus_i ((p_2 + q_1) \oplus_j (p_2 + q_2))$. These two axioms are applied to each $p_i + q_j$, where $i, j \in \{1, 2\}$ repeatedly until the operands of $+$ become CCS terms.
- if $t \equiv t'_1 \oplus t'_2$, $t$ can be rewritten into the required format by applying the induction hypothesis on $t'_1$ and $t'_2$.
- if $t \equiv t'_1 \parallel t'_2$, then $t$ can be rewritten into the required format by applying the induction hypothesis on $t'_1$ and $t'_2$, and then applying axioms $D_{1,2}$. Assume $\vdash t'_1 = p_1 \oplus_i p_2$ and $\vdash t'_2 = q_1 \oplus_j q_2$. By axioms $D_{1,2} \vdash t = ((p_1 \parallel q_1) \oplus_j (p_1 \parallel q_2)) \oplus_i ((p_2 \parallel q_1) \oplus_j (p_2 \parallel q_2))$. These two axioms are applied to each $p_i \parallel q_j$, where $i, j \in \{1, 2\}$ repeatedly until the operands of $\parallel$ become CCS terms.
- if $t \equiv t'_1 \between t'_2$, then $t$ can be rewritten into the required format by applying the induction hypothesis on $t'_1$ and $t'_2$, and then applying axioms $P_{4,5}$. Assume $\vdash t'_1 = p_1 \oplus_i p_2$ and $\vdash t'_2 = q_1 \oplus_j q_2$. By axioms $P_{4,5} \vdash t = ((p_1 \between q_1) \oplus_j (p_1 \between q_2)) \oplus_i ((p_2 \between q_1) \oplus_j (p_2 \between q_2))$. These two axioms are applied to each $p_i \between q_j$, where $i, j \in \{1, 2\}$ repeatedly until the operands of $\between$ become CCS terms.
- if $t \equiv t'_1 \mid t'_2$, then $t$ can be rewritten into the required format by applying the induction hypothesis on $t'_1$ and $t'_2$, and then applying axioms $S_{1,3}$. Assume $\vdash t'_1 = p_1 \oplus_i p_2$ and $\vdash t'_2 = q_1 \oplus_j q_2$. By axioms $S_{1,3} \vdash t = ((p_1 \mid q_1) \oplus_j (p_1 \mid q_2)) \oplus_i ((p_2 \mid q_1) \oplus_j (p_2 \mid q_2))$. These two axioms are applied to each $p_i \mid q_j$, where $i, j \in \{1, 2\}$ repeatedly until the operands of $\mid$ become CCS terms.
- if $t \equiv \langle A | \{A \stackrel{def}{=} t'\}\rangle$, then by applying the induction hypothesis, we obtain $\vdash t' = p_1 \oplus_i p_2$. By axiom $Dri \vdash t = \langle A | \{A \stackrel{def}{=} p_1\}\rangle \oplus_i \langle A | \{A \stackrel{def}{=} p_2\}\rangle$. We apply this axiom repeatedly to $p_1$ and $p_2$, until no binary variant operator occurs in the equations defining $A$.

Hence, $t$ is rewritten into a form where no binary variant is in the scope of CCS operator, i.e., $\vdash t = p_1 \oplus_i p_2$, where $p_1$ and $p_2$ are also in this form. For all $j \in bi(p_1 \oplus_i p_2)$, first apply $N_5$ to any subterm $p' \oplus_j q'$, i.e. $\vdash p' \oplus_j q' = L(p', j) \oplus_j L(q', j)$ and then $N_{1-4}$ to make $j$ free in $p'$ and $q'$. We claim that the resulting term $t'' \equiv p'_1 \oplus_i p'_2$ is fully expanded, otherwise there is a simple path from $\oplus_j$ to another $\oplus_j$. This is impossible due to application of $N_{1-5}$.

For all $j \in bi(p'_1) \cap bi(p'_2)$, apply $N_6$ to $p'_2$, and replace all occurrences of $j$ by a fresh index $k \notin bi(p'_1) \cup bi(p'_2)$ to derive $t'''$. Hence, all indices are unique, and the term is fully expanded. Theorem 19 is proved by applying axioms $A_{1,2}$ to $t'''$.

## Appendix C. Soundness of axiomatization

To prove the soundness of axioms, we show that all axioms except $A_{1,2}$ and $N_6$ are sound with respect to strict strong bisimilarity, while $A_{1,2}$ and $N_6$ are sound with respect to product line bisimilarity.

To show the soundness of $A_1$, for any $\nu_1^f \in VFConfig(p \oplus_i q)$, we define $\nu_2^f \in VFConfig(q \oplus_i p)$ such that $|\nu_1^f| = |\nu_2^f|$ and $\forall_{j \neq i}(\nu_1^f|_j = \nu_2^f|_j)$, $(\nu_1^f|_i = R \Rightarrow \nu_2^f|_i = L)$, and $(\nu_1^f|_i = L \Rightarrow \nu_1^f|_i = R)$. We show that $\Pi(p \oplus_i q, \nu_1^f) \sim \Pi(q \oplus_i p, \nu_2^f)$. To this aim, we show that $\mathcal{R} = \{(\Pi(p \oplus_i q, \nu_1^f), \Pi(q \oplus_i p, \nu_2^f))\} \cup \{(\Pi(t, \nu_1^f), \Pi(t, \nu_2^f))\}$ is a strong bisimulation. Regarding $\nu_1^f|_i$, two cases can be distinguished. We only discuss the case where $\nu_1^f|_i = L$, as the other can be dealt with in the same fashion.

We show that the transfer conditions hold for the pair $(\Pi(p \oplus_i q, \nu_1^f), \Pi(q \oplus_i p, \nu_2^f)) \in \mathcal{R}$, as for others, the proof is straightforward. Suppose $\Pi(p \oplus_i q, \nu_1^f) \xrightarrow{a} \Pi(p', \nu_1^f)$, since $p \xrightarrow{a, \nu_1} p'$, where $\nu_1 \sqsubseteq \nu_1^f$ and $\nu_1|_i =?$. Therefore, $\nu_1 \sqsubseteq \nu_2^f$. By SOS rule *Select*, $q \oplus_i p \xrightarrow{a, \nu_2} p'$, where $\nu_2 = \nu_1|_{i/R}$, $\nu_2 \sqsubseteq \nu_2^f$, and $\Pi(p', \nu_1^f) \mathcal{R} \Pi(p', \nu_2^f)$. The same discussion holds when $\Pi(q \oplus_i p, \nu_2^f) \xrightarrow{a} \Pi(p', \nu_2^f)$. Similarly, for any $\nu_2^f \in VFConfig(q \oplus_i p)$, we define $\nu_1^f \in VFConfig(q \oplus_i p)$ as discussed in above. Concluding that $p \oplus_i q \simeq_{PL} q \oplus_i p$. Axiom $A_2$ is proved similar to $A_1$: for any $\nu_1^f \in VFConfig((p \oplus_i q) \oplus_j r)$, we define $\nu_2^f \in VFConfig(p \oplus_i (q \oplus_j r))$ such that $|\nu_1^f| = |\nu_2^f|$ and $\forall_{k \neq i, k \neq j}(\nu_1^f|_k = \nu_2^f|_k)$, $(\nu_1^f|_i = R \wedge \nu_1^f|_j = R) \Rightarrow (\nu_2^f|_i = R \wedge \nu_2^f|_j = R)$, $(\nu_1^f|_i = L \wedge \nu_1^f|_j = R) \Rightarrow (\nu_2^f|_i = R \wedge \nu_2^f|_j = R)$, $(\nu_1^f|_i = R \wedge \nu_1^f|_j = L) \Rightarrow (\nu_2^f|_i = R \wedge \nu_2^f|_j = L)$, and $(\nu_1^f|_i = L \wedge \nu_1^f|_j = L) \Rightarrow (\nu_2^f|_i = L \wedge \nu_2^f|_j = L)$. It can be easily shown that $\Pi((p \oplus_i q) \oplus_j r, \nu_1^f) \sim \Pi(p \oplus_i (q \oplus_j r), \nu_2^f)$. Axiom $N_6$ is proved similar to $A_{1,2}$: for any $\nu_1^f \in VFConfig(p)$, we define $\nu_2^f \in VFConfig(p[k/i])$ such that $\nu_2^f = \nu_1^f \cdot \lambda$ and $\nu_1^f|_i = \nu_2^f|_k$.

For axiom $A_3$, we show that for any $\nu^f \in VFConfig(p \oplus_i p)$ (which implicitly implies $\nu^f \in VFConfig(p)$), $\Pi((p \oplus_i p), \nu^f) \sim \Pi(p, \nu^f)$. It is trivial that $\mathcal{R} = \{(\Pi((p \oplus_i p), \nu^f)\} \cup \{(\Pi(t, \nu^f), \Pi(t, \nu^f))\}$ is a strong bisimulation. Axioms $N_{1,3}$ are proved similarly.

For axiom $A_4$, for any $\nu^f \in VFConfig(a.(p \oplus_i q))$ (which implicitly implies $\nu^f \in VFConfig(a.p \oplus_i a.q)$), $\Pi(a.(p \oplus_i q), \nu^f) \sim \Pi(a.p \oplus_i a.q, \nu^f)$. Regarding the value of $\nu^f|_i$, two cases can be distinguished. We only discuss the case that $\nu^f|_i = L$, as the other cases can be proven identically. Therefore, we show that

$$\mathcal{R} = \{(\Pi(a.(p \oplus_i q), \nu^f), \Pi(a.p \oplus_i a.q, \nu^f)), (\Pi((p \oplus_i q), \nu^f), \Pi(p, \nu^f))\} \cup \{(\Pi(t, \nu^f), \Pi(t, \nu^f))\}$$

is a strong bisimulation and satisfies the transfer conditions of Definition 3. For the pair $(\Pi(a.(p \oplus_i q), \nu^f), \Pi(a.p \oplus_i a.q, \nu^f)) \in \mathcal{R}$, suppose $\Pi(a.(p \oplus_i q), \nu^f) \xrightarrow{a} \Pi(p \oplus q_i, \nu^f)$ since by SOS rule *Prefix* $a.(p \oplus_i q) \xrightarrow{a, \nu_?} p \oplus_i q$, where $\nu_? \sqsubseteq$

$\nu^f$. By SOS rules *Prefix* and *Select*, $a.p \oplus_i a.q \xrightarrow{a,\nu_?|_{i/L}} p$, $\nu_?|_{i/L} \sqsubseteq \nu^f$. Therefore, $\Pi(a.p \oplus_i a.q, \nu^f) \xrightarrow{a} \Pi(p, \nu^f)$, and $\Pi((p \oplus_i q), \nu^f)\ \mathcal{R}\ \Pi(p, \nu^f)$. The same discussion holds when $\Pi(a.p \oplus a.q, \nu^f) \xrightarrow{a} \Pi(p, \nu^f)$. Furthermore, for the pair $(\Pi((p \oplus_i q), \nu^f), \Pi(p, \nu^f)) \in \mathcal{R}$, assume that $\Pi((p \oplus_i q), \nu^f) \xrightarrow{a} \Pi(p', \nu^f)$ since $p \oplus_i q \xrightarrow{a,\nu} p'$ and $\nu \sqsubseteq \nu^f$. Therefore, $p \xrightarrow{a,\nu'} p'$, where $\nu'|_{i/L} = \nu$, and $\nu' \sqsubseteq \nu^f$. Consequently $\Pi(p, \nu^f) \xrightarrow{a} \Pi(p', \nu^f)$ and $\Pi(p', \nu^f)\ \mathcal{R}\ \Pi(p', \nu^f)$. The same discussion holds when $\Pi(p, \nu^f) \xrightarrow{a} \Pi(p', \nu^f)$. Transfer conditions trivially hold for pairs like $(\Pi(t, \nu^f), \Pi(t, \nu^f)) \in \mathcal{R}$. Consequently $\mathcal{R}$ is a strong bisimulation, concluding that $a.(p \oplus_i q) \approx_{PL} a.p \oplus_i a.q$.

To prove axiom $A_5$, for any $\nu^f \in \mathit{VFConfig}((p \oplus_i q) + r)$ (which implicitly implies $\nu^f \in \mathit{VFConfig}((p + r) \oplus_i (q + r))$), it is straightforward to show that $\Pi((p \oplus_i q) + r, \nu^f) \sim \Pi((p + r) \oplus_i (q + r), \nu^f)$ witnessed by the strong bisimulation relation $\mathcal{R} = \{(\Pi((p \oplus_i q) + r, \nu^f), \Pi((p + r) \oplus_i (q + r), \nu^f))\} \cup \{(\Pi(t, \nu^f), \Pi(t, \nu^f))\}$. Axioms $A_6$, $P_{4,5}$ $R_3$, $E_5$, $N_{2,4,5}$, $S_3$ are proved similar to $A_5$.

For axiom $C_1$, it can be proved that for any $\nu^f \in \mathit{VFConfig}(p + q)$ (which implicitly implies $\nu^f \in \mathit{VFConfig}(q + p)$) $\Pi(p + q, \nu^f) \sim \Pi(q + p, \nu^f)$. It is easy to show that $\mathcal{R} = \{(\Pi(p + q, \nu^f), \Pi(q + p, \nu^f))\} \cup \{(\Pi(t, \nu^f), \Pi(t, \nu^f))\}$ is a strong bisimulation. Axioms $C_{2-4}$, $P_{1-3,6}$, $S_{1,2,4,5,6}$, $R_{1,2,4}$, and $E_{1-4}$ are proved similarly.

Axioms *Dec*, *UnFold*, *Fold*, and *Ung* are standard [17]. We provide a full proof for the new axiom *Dri*. To prove axiom *Dri*, we show that for any $\nu^f \in \mathit{VFConfig}(\langle A|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle)$ (which implicitly implies $\nu^f \in \mathit{VFConfig}(\langle A|\{A \overset{def}{=} t_1\}\rangle \oplus_i \langle A|\{A \overset{def}{=} t_2\}\rangle)$), $\Pi(\langle A|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f) \sim \Pi(\langle A|\{A \overset{def}{=} t_1\}\rangle \oplus_i \langle A|\{A \overset{def}{=} t_2\}\rangle, \nu^f)$. We only discuss on the case when $\nu^f|_i = L$ as the other can be dealt with in the same way. To this aim, we prove that $\mathcal{R} = \{(\Pi(\langle t|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle), \Pi(\langle t|\{A \overset{def}{=} t_1\}\rangle \oplus_i \langle t|\{A \overset{def}{=} t_2\}\rangle, \nu^f))\} \cup \{(\Pi(\langle t|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle), \Pi(\langle t|\{A \overset{def}{=} t_1\}\rangle, \nu^f))\}$ is a strong bisimulation.

The transfer conditions of Definition 3 for the pair $(\Pi(\langle t|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f), \Pi(\langle t|\{A \overset{def}{=} t_1\}\rangle \oplus_i \langle t|\{A \overset{def}{=} t_2\}\rangle, \nu^f))$ can be examined by structural induction over the syntax of $t$. The base case of induction for $t \equiv 0$ is trivial.

- if $t \equiv a.t'$, then by rule *Prefix*, $\langle a.t'|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle \xrightarrow{a,\nu_?} \langle t'|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle$ and $\nu_? \sqsubseteq \nu^f$. Similarly, $\langle a.t'|\{A \overset{def}{=} t_1\}\rangle \oplus_i \langle a.t'|\{A \overset{def}{=} t_2\}\rangle \xrightarrow{a,\nu_?|_{i/L}} \langle t'|\{A \overset{def}{=} t_1\}\rangle$. Hence, $\Pi(\langle a.t'|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t'|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f)$, $\Pi(\langle a.t'|\{A \overset{def}{=} t_1\}\rangle \oplus_i \langle a.t'|\{A \overset{def}{=} t_2\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t'|\{A \overset{def}{=} t_1\}\rangle, \nu^f)$, and $\Pi(\langle t'|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f)\ \mathcal{R}\ \Pi(\langle t'|\{A \overset{def}{=} t_1\}\rangle, \nu^f)$.
- if $t \equiv t_1 + t_2$, then by rule *Choice*, either $\langle t_1 + t_2|\{A \overset{def}{=} p \oplus_i q\}\rangle \xrightarrow{a,\nu_1} \langle t'_1|\{A \overset{def}{=} p \oplus_i q\}\rangle$ or $\langle t_1 + t_2|\{A \overset{def}{=} p \oplus_i q\}\rangle \xrightarrow{b,\nu_2} \langle t'_2|\{A \overset{def}{=} p \oplus_i q\}\rangle$, and $\nu_1 \sqsubseteq \nu^f$ or $\nu_2 \sqsubseteq \nu^f$, since $\langle t_1|\{A \overset{def}{=} p \oplus_i q\}\rangle \xrightarrow{a,\nu_1} \langle t'_1|\{A \overset{def}{=} p \oplus_i q\}\rangle$ or $\langle t_2|\{A \overset{def}{=} p \oplus_i q\}\rangle \xrightarrow{b,\nu_2} \langle t'_2|\{A \overset{def}{=} p \oplus_i q\}\rangle$. Hence, $\Pi(\langle t_1 + t_2|\{A \overset{def}{=} p \oplus_i q\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t'_1|\{A \overset{def}{=} p \oplus_i q\}\rangle, \nu^f)$ or $\Pi(\langle t_1 + t_2|\{A \overset{def}{=} p \oplus_i q\}\rangle, \nu^f) \xrightarrow{b} \Pi(\langle t'_2|\{A \overset{def}{=} p \oplus_i q\}\rangle, \nu^f)$. By induction, $\Pi(\langle t_1|\{A \overset{def}{=} p \oplus_i q\}\rangle, \nu^f) \sim \Pi(\langle t_1|\{A \overset{def}{=} p\}\rangle \oplus_i \langle t_1|\{A \overset{def}{=} q\}\rangle, \nu^f)$ and $\Pi(\langle t_2|\{A \overset{def}{=} p \oplus_i q\}\rangle, \nu^f) \sim \Pi(\langle t_2|\{A \overset{def}{=} p\}\rangle \oplus_i \langle t_2|\{A \overset{def}{=} q\}\rangle, \nu^f)$. Therefore, $\Pi(\langle t_1|\{A \overset{def}{=} p\}\rangle \oplus_i \langle t_1|\{A \overset{def}{=} q\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t'_1|\{A \overset{def}{=} p\}\rangle, \nu^f)$, or $\Pi(\langle t_2|\{A \overset{def}{=} p\}\rangle \oplus_i \langle t_2|\{A \overset{def}{=} q\}\rangle, \nu^f) \xrightarrow{b} \Pi(\langle t'_2|\{A \overset{def}{=} p\}\rangle, \nu^f)$, and $\Pi(\langle t'_1|\{A \overset{def}{=} p \oplus_i q\}\rangle, \nu^f)\ \mathcal{R}\ \Pi(\langle t'_1|\{A \overset{def}{=} p\}\rangle, \nu^f)$, and $\Pi(\langle t'_2|\{A \overset{def}{=} p \oplus_i q\}\rangle, \nu^f)\ \mathcal{R}\ \Pi(\langle t'_2|\{A \overset{def}{=} p\}\rangle, \nu^f)$. Consequently, $\Pi(\langle t_1 + t_2|\{A \overset{def}{=} p\}\rangle \oplus_i \langle t_1 + t_2|\{A \overset{def}{=} q\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t'_1|\{A \overset{def}{=} p\}\rangle, \nu^f)$ or $\Pi(\langle t_1 + t_2|\{A \overset{def}{=} p\}\rangle \oplus_i \langle t_1 + t_2|\{A \overset{def}{=} q\}\rangle \xrightarrow{b} \Pi(\langle t'_2|\{A \overset{def}{=} p\}\rangle, \nu^f)$. The same discussion holds when $\langle t_1 + t_2|\{A \overset{def}{=} p\}\rangle \oplus_i \langle t_1 + t_2|\{A \overset{def}{=} q\}\rangle \xrightarrow{a,\nu_1} \langle t'_1|\{A \overset{def}{=} p\}\rangle$ or $\langle t_1 + t_2|\{A \overset{def}{=} p\}\rangle \oplus_i \langle t_1 + t_2|\{A \overset{def}{=} q\}\rangle \xrightarrow{b,\nu_2} \langle t'_2|\{A \overset{def}{=} q\}\rangle$.
- if $t \equiv t_1 \oplus_j t_2$, $t \equiv t_1 \parallel t_2$, $t \equiv t_1 \between t_2$, $t \equiv t_1 \mid t_2$, $t \equiv t' \setminus L$, or $t \equiv t'[f]$, then the proof is almost identical to the previous case.
- if $t \equiv \langle A|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle$, then $\Pi(\langle A|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t'_1|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f)$ since $\langle A|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle \xrightarrow{a,\nu} \langle t'_1|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle$, where $\nu \sqsubseteq \nu_f$, and by SOS rules *call* and *Select*, $\langle t_1|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle \xrightarrow{a,\nu'} \langle t'_1|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle$, where $\nu'|_{i/L} = \nu$. Therefore, by induction $\Pi(\langle t_1|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f) \sim \Pi(\langle t_1|\{A \overset{def}{=} t_1\}\rangle, \nu^f)$, and hence $\Pi(\langle t_1|\{A \overset{def}{=} t_1\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t'_1|\{A \overset{def}{=} t_1\}\rangle, \nu^f)$, and $\Pi(\langle t'_1|\{A \overset{def}{=} t_1\}\rangle, \nu^f) \sim \Pi(\langle t'_1|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f)$. Thus, by SOS rules *Select* and *Call*, $\Pi(\langle A|\{A \overset{def}{=} t_1\}\rangle \oplus_i \langle A|\{A \overset{def}{=} t_2\}\rangle, \nu^f) \xrightarrow{a} \Pi(\langle t'_1|\{A \overset{def}{=} t_1\}\rangle, \nu^f)$ and $\Pi(\langle t'_1|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f)\ \mathcal{R}\ \Pi(\langle t'_1|\{A \overset{def}{=} t_1\}\rangle, \nu^f)$. The same discussion holds when $\Pi(\langle A|\{A \overset{def}{=} t_1\}\rangle \oplus_i \langle A|\{A \overset{def}{=} t_2\}\rangle \xrightarrow{a} \langle t'_1|\{A \overset{def}{=} t_1\}\rangle$.

The transfer conditions of Definition 3 for the pair $(\Pi(\langle t|\{A \overset{def}{=} t_1 \oplus_i t_2\}\rangle, \nu^f), \Pi(\langle t|\{A \overset{def}{=} t_1\}\rangle))$ can be examined by structural induction over the syntax of $t$ as discussed in above.

## Appendix D. Ground-completeness of axiomatization

We are going to prove Theorem 16, that the axiomatization of PL-CCS is ground-complete for closed, finite-state terms modulo product line bisimilarity. The idea behind the ground-completeness proof, following the approach of [14], is to show that $t = s$ via the intermediate results $t = t \oplus_i s$ and $t \oplus_i s = s$, which imply that $t = t \oplus_i s = s$. The following lemmata are required before explaining the proof.

**Lemma 26.** *Let $t$, $s$, and $r$ be fully expanded. If $(t \oplus_i s) \oplus_j r \simeq_{PL} r$, where $i \neq j$, $i \notin bi(t) \cup bi(s)$ and $j \notin bi(r)$, then $t \oplus_j r \simeq_{PL} r$ and $s \oplus_j r \simeq_{PL} r$.*

**Proof.** We show that $t \oplus_j r \simeq_{PL} r$ as the other case is symmetric. Regarding Definition 5, we must show that

- for any $\nu^{f'}{}_1 \in \mathit{VFConfig}(t \oplus_j r)$, there exists $\nu^{f'}{}_2 \in \mathit{VFConfig}(r)$ such that $\Pi(t \oplus_j r, \nu^{f'}{}_1) \sim \Pi(r, \nu^{f'}{}_2)$.
  Assume $\nu^{f'}{}_1|_j = L$; hence, $\nu^{f'}{}_1$ can be written as $\nu_t \cdot \lambda$, where $\nu_t \in \mathit{VFConfig}(t)$. Consider $\nu^f{}_1 \in \mathit{VFConfig}((t \oplus_i s) \oplus_j r)$ such that $\nu^f{}_1|_i = L$, $\nu^f{}_1|_j = L$, and $\forall k \leq |\nu_t| \wedge (k \neq i)(\nu^f{}_1|_k = \nu^{f'}{}_1|_k)$. Hence, by Definition 5, $(t \oplus_i s) \oplus_j r \simeq_{PL} r$ implies that there exists $\nu^f_2 \in \mathit{VFConfig}(r)$ such that $\Pi((t \oplus_i s) \oplus_j r, \nu^f_1) \sim \Pi(r, \nu^f_2)$. Define $\nu^{f'}{}_2 = \nu^f{}_2$. It immediately follows that $\Pi(t \oplus_j r, \nu^{f'}{}_1) \sim \Pi(r, \nu^{f'}{}_2)$.
  Assume $\nu^{f'}{}_1|_j = R$, so it can be written as $\nu^{f'}{}_1 = \nu_r \cdot \lambda$, where $\nu_r \in \mathit{VFConfig}(r)$. Consider $\nu^f{}_1 \in \mathit{VFConfig}((t \oplus_i s) \oplus_j r)$ such that $\nu^f{}_1|_j = R$, and $\forall k \leq |\nu_r|(\nu^f{}_1|_k = \nu^{f'}{}_1|_k)$. Define $\nu^{f'}{}_2 = \nu_r$. It immediately follows that $\Pi(t \oplus_j r, \nu^{f'}{}_1) \sim \Pi(r, \nu^{f'}{}_2)$.
- for any $\nu^{f'}{}_2 \in \mathit{VFConfig}(r)$, there exists $\nu^{f'}{}_1 \in \mathit{VFConfig}(t \oplus_j r)$ such that $\Pi(t \oplus_j r, \nu^{f'}{}_1) \sim \Pi(r, \nu^{f'}{}_2)$. Take $\nu^{f'}{}_1$ such that $\nu^{f'}{}_1|_j = R$ and $\forall k \leq |\nu_r| \wedge (k \neq j)(\nu^{f'}{}_1|_k = \nu^{f'}{}_2|_k)$. It follows immediately that $\Pi(t \oplus_j r, \nu^{f'}{}_1) \sim \Pi(r, \nu^{f'}{}_2)$. □

**Lemma 27.** *Let $\bigoplus_{i \leq n} p_i$, where $n > 0$, be a PL-CCS term such that $p_i$s are CCS terms. Then $\Pi(\bigoplus_{i \leq n} p_i, \nu_f) \sim p_j$, where $\nu_f|_j = L$ and $\forall k < j(\nu_f|_k = R)$.*

**Proof.** It is straightforward to check that for a given CCS term $p$, $\Pi(p, \nu_f) \sim p$ since $\nu_f$ has no effect on deriving transitions of $p$ ($\nu_f$ is only considered in rules *Select*, *RSelect*, and *LSelect*). Therefore, $\Pi(\bigoplus_{i \leq n} p_i, \nu_f) \xrightarrow{a} \Pi(p', \nu_f)$, since $\bigoplus_{i \leq n} p_i \xrightarrow{a,\nu} p'$, where $\nu \sqsubseteq \nu_f$. Hence, by SOS rule *Select*, $p_j \xrightarrow{a,\nu_2} p'$ and $\forall k < j(\nu|_k) = R$ and $\nu|_j = L$. □

**Theorem.** *For all closed finite-state* PL-CCS$_f$ *terms $t_1$ and $t_2$, $t_1 \simeq_{PL} t_2$ implies $t_1 = t_2$.*

**Proof.** By Theorem 19, PL-CCS$_f$ terms $t$ and $s$ can be derived by our axiomatization into fully expanded terms $t' \equiv \bigoplus_{i \leq n} p_i$ and $s' \equiv \bigoplus_{j \leq m} q_j$, where $p_i$s and $q_i$s are CCS term such that every recursive specification $E$ included in $p_i$s or $q_i$s is essentially finite state. Since our axiomatization subsumes CCS axiomatization, completeness of CCS axiomatization for closed finite-state CCS terms [17] implies $p \sim q \Leftrightarrow p = q$. The soundness of our axiomatization yields $t \simeq_{PL} t'$ and $s \simeq_{PL} s'$. By transitivity of product line bisimilarity, $t' \simeq_{PL} s'$. Therefore, it is enough to prove that $t' \oplus_i s' \simeq_{PL} s' \Rightarrow t' \oplus_i s' = s'$ and $t' \simeq_{PL} t' \oplus_i s' \Rightarrow t' = t' \oplus_i s'$, where $i$ is free in $s'$ and $t'$. These properties are sufficient to prove the theorem as follows: If $t' \simeq_{PL} s'$, then, by the fact that product line bisimilarity is reflexive (i.e., $t' \simeq_{PL} t'$ and $s' \simeq_{PL} s'$) and the fact that product line bisimilarity is a congruence on fully expanded PL-CCS terms, we have $t' \oplus_i t' \simeq_{PL} s' \oplus_i t'$ and $t' \oplus_i s' \simeq_{PL} s' \oplus_i s'$ (note that $t' \oplus_i s'$ is still fully expanded). The soundness of axiomatization, more in particular the validity of Axiom $A_3$, implies that $t' \oplus_i t' \simeq_{PL} t'$ and $s' \oplus_i s' \simeq_{PL} s'$. Using symmetry and transitivity of product line bisimilarity, $t' \simeq_{PL} t' \oplus_i s'$ and $t' \oplus_i s' \simeq_{PL} s'$ are obtained. Thus, above properties yield that $t' = t' \oplus_i s'$ and $t' \oplus_i s' = s'$. These last results can be combined to show that $t' = t' \oplus_i s' = s'$. Consequently, $t = t'$ and $s = s'$ together with $t' = s'$ result $t = s$.

For all fully expanded PL-CCS$_f$ terms $t' \oplus_i s'$, $t'$ and $s'$, where $t' \equiv \bigoplus_{i \leq n} p_i$ and $s' \equiv \bigoplus_{j \leq m} q_j$, $t' \oplus_i s' \simeq_{PL} s'$ implies $t' \oplus_i s' = s'$, is proven by induction on number of CCS terms of $t'$, i.e. $n$. The proof of $t' \simeq_{PL} t' \oplus_i s'$ implies $t' = t' \oplus_i s'$ is similar and therefore omitted.

The base case of the induction corresponds to the case that $n = 0$. Therefore, $p_1 \oplus_i s' \simeq_{PL} s$ implies for $\nu^f{}_1$, where $\nu^f{}_1|_1 = L$, there exists $\nu^f{}_2$ such that $\Pi(p_1 \oplus_i s', \nu^f{}_1) \sim \Pi(s', \nu^f{}_2)$. By Lemma 27, $\Pi(p_1 \oplus_i s', \nu^f{}_1) \sim p_1$ and $\Pi(s', \nu^f{}_2) \sim q_k$, where $\nu^f{}_2|_k = L$ and $\forall j < k(\nu^f{}_2|_j = R)$. Therefore, $p_1 \sim q_k$ and consequently by completeness of axiomatization of CCS, $p_1 = q_k$. Thus, $p_1 \oplus_i s' = q_k \oplus_i \bigoplus_{j \leq m} q_j =^{A_1,A_2,A_3} s'$.

Assume that for all $0 < w < n$ such that $t' \equiv \bigoplus_{i \leq w} p_i$, $t' \oplus_i s' \simeq_{PL} s'$ implies $t' \oplus_i s' = s'$. We prove that $t' \oplus_i s' \simeq_{PL} s'$, where $t' \equiv \bigoplus_{i \leq n} p_i$, implies $t' \oplus_i s' = s'$. By Lemma 26, $t' \oplus_i s' \simeq_{PL} s'$ implies $p_1 \oplus_i s' \simeq_{PL} s'$ and $(\bigoplus_{1 < i \leq n} p_i) \oplus_i s' \simeq_{PL} s'$. By application of axiom $N_6$, $\bigoplus_{1 < i \leq n} p_i$ can be derived into $t'' \equiv \bigoplus_{k < n} p_k$. Therefore, by soundness of axiomatization and transitivity of product line bisimilarity, $(\bigoplus_{k < n} p_k) \oplus_i s' \simeq_{PL} s'$. By induction $p_1 \oplus_i s' = s'$, $t'' \oplus_i s' = s'$. Therefore, $t' \oplus_i s' = (p_1 \oplus_1 (\bigoplus_{1 < i \leq n} p_i)) \oplus_i s' =^{N_6} (p_1 \oplus_z (\bigoplus_{1 < i \leq n} p_i)) \oplus_i s' =^{A_1,A_2} (\bigoplus_{1 < i \leq n} p_i) \oplus_z (p_1 \oplus_i s') = (\bigoplus_{1 < i \leq n} p_i) \oplus_z s' = t'' \oplus_i s' = s'$, where $z$ is a fresh index such that $z > m$, $z > n$, and $z \neq i$.

## Appendix E. Proof of Theorem 23

We use the following so called *reduction* Lemma taken from [51] to complete the proof. The right-hand sides of the bi-implications express the unfolding of the fixed points: in case of minimum, a single element $p$, is removed while for the maximum it is added.

**Lemma 28.** *For $\psi$ a monotonic function on a powerset $Pow(D)$ with $p \in D$, we have:*

$$p \in \mu V.\psi(V) \Leftrightarrow p \in \psi(\mu V.(\psi(V) \setminus \{p\})),$$
$$p \in \nu V.\psi(V) \Leftrightarrow p \in \psi(\nu V.(\psi(V) \cup \{p\})).$$

The proof of Theorem 23 follows:

"$\Rightarrow$" Suppose $r \simeq_{PL} s$ and let $\varphi \in mv - \mathfrak{L}_\mu$. We prove that for all valid full configuration $\nu_1^f \in VFConfig(r)$ and $\nu_2^f \in VFConfig(s)$ that $\Pi(r, \nu_1^f) \sim \Pi(s, \nu_2^f)$, $\nu_1^f \in [\![\varphi]\!](r)$ iff $\nu_2^f \in [\![\varphi]\!](s)$. The proof is managed by induction on the structure of $\varphi$.

1. If $\varphi = true$, then obviously $\nu_1^f \in [\![\varphi]\!](r)$ and $\nu_2^f \in [\![\varphi]\!](s)$.
2. If $\varphi = \varphi_1 \wedge \varphi_2$, then by definition $\nu_1^f \in [\![\varphi_1]\!](r)$ and $\nu_1^f \in [\![\varphi_2]\!](r)$, and the claim follows by straightforward induction.
3. If $\varphi = \varphi_1 \vee \varphi_2$, it is proved similar to previous case.
4. If $\varphi = \langle a \rangle \varphi'$, then by definition $\nu_1^f \in [\![\varphi]\!](r)$, if $\nu_1^f \in \mathcal{R}_a(r, r')$ and $\nu_1^f \in [\![\varphi']\!](r')$ for some $r'$. Hence $\nu_1^f \in \mathcal{R}_a(r, r')$ implies that $r \xrightarrow{a, \nu_1} r'$ for some $\nu_1$ such that $\nu_1 \sqsubseteq \nu_1^f$. Consequently $\Pi(r, \nu_1^f) \xrightarrow{a} \Pi(r', \nu_1^f)$. By assumption, $\Pi(r, \nu_1^f) \sim \Pi(s, \nu_2^f)$ implies $\Pi(s, \nu_2^f) \xrightarrow{a} \Pi(s', \nu_2^f)$ and $\Pi(r', \nu_1^f) \sim \Pi(s', \nu_2^f)$. Thus $s \xrightarrow{a, \nu_2} s'$ for some $\nu_2$ such that $\nu_2 \sqsubseteq \nu_2^f$ and $\nu_2^f \in \mathcal{R}_a(s, s')$. Concluding by induction that $\nu_2^f \in [\![\varphi']\!](s')$, the claim follows.
5. If $\varphi = [a]\varphi'$, then by definition $\nu_1^f \in [\![\varphi]\!](r)$ implies that for all $s'$:
   - either $\nu_1^f \in \mathcal{R}_a(r, r')$ and $\nu_1^f \in [\![\varphi']\!](r')$: Hence $\nu_1^f \in \mathcal{R}_a(r, r')$ implies that $r \xrightarrow{a, \nu_1} r'$ for some $\nu_1$ such that $\nu_1 \sqsubseteq \nu_1^f$. Consequently $\Pi(r, \nu_1^f) \xrightarrow{a} \Pi(r', \nu_1^f)$. By assumption, $\Pi(r, \nu_1^f) \sim \Pi(s, \nu_2^f)$ implies $\Pi(s, \nu_2^f) \xrightarrow{a} \Pi(s', \nu_2^f)$ and $\Pi(r', \nu_1^f) \sim \Pi(s', \nu_2^f)$. Thus $s \xrightarrow{a, \nu_2} s'$ for some $\nu_2$ such that $\nu_2 \sqsubseteq \nu_2^f$ and $\nu_2^f \in \mathcal{R}_a(s, s')$. Concluding by induction that $\nu_2^f \in [\![\varphi']\!](s')$, the claim follows.
   - or $\nu_1^f \notin \mathcal{R}_a(r, r')$: Hence there exists no transition that $r \xrightarrow{a, \nu_1} r'$ for some $\nu_1$ such that $\nu_1 \sqsubseteq \nu_1^f$. Consequently $\Pi(r, \nu_1^f) \xrightarrow{a} \not\rightarrow$. By assumption, $\Pi(r, \nu_1^f) \sim \Pi(s, \nu_2^f)$ implies $\Pi(r, \nu_1^f) \xrightarrow{a} \not\rightarrow$, and hence $\nu_2^f \notin \mathcal{R}_a(s, s')$ for any $s'$.
6. If $\varphi = \mu Z.\phi$, then by Lemma 28, $\nu_1^f \in [\![\varphi]\!](r)$ implies $\nu_1^f \in [\![\phi]\!]_{\rho[Z \mapsto \mu Z.\phi \setminus \{\nu_1^f\}]}(r)$. By induction $\nu_2^f \in [\![\phi]\!]_{\rho[Z \mapsto \mu Z.\phi \setminus \{\nu_2^f\}]}(s)$, and hence $\nu_2^f \in [\![\varphi]\!](s)$.
7. If $\varphi = \nu Z.\phi$, then by Lemma 28, $\nu_1^f \in [\![\varphi]\!](r)$ implies $\nu_1^f \in [\![\phi]\!]_{\rho[Z \mapsto \nu Z.\phi \cup \{\nu_1^f\}]}(r)$. By induction $\nu_2^f \in [\![\phi]\!]_{\rho[Z \mapsto \nu Z.\phi \cup \{\nu_2^f\}]}(s)$, and consequently $\nu_2^f \in [\![\varphi]\!](s)$.

"$\Leftarrow$" Suppose $r \sim_L s$. We prove that for any valid full configuration $\nu_1^f \in VFConfig(r)$, there exists $\nu_2^f \in VFConfig(s)$ such that $\Pi(r, \nu_1^f) \sim \Pi(s, \nu_2^f)$. To this aim, we assume that $\Pi(r, \nu_1^f)$ is image finite, i.e., $\{\Pi(r', \nu_1^f) | \Pi(r, \nu_1^f) \xrightarrow{a} \Pi(r', \nu_1^f)\}$ is finite. The result can be lifted to general cases in the same vein as [14]. For $\nu_1^f \in VFConfig(r)$, we find $\varphi \in mv - \mathfrak{L}_\mu$ such that it characterizes the strong bisimulation class for $\Pi(r, \nu_1^f)$, called *characteristic formula*, following the approach of [52]. Since $\nu_1^f \in [\![\varphi]\!](r)$, by Definition 22, there exists $\nu_2^f \in [\![\varphi]\!](s)$. Therefore, $\Pi(r, \nu_1^f) \sim \Pi(s, \nu_2^f)$.

Similarly for any valid full configuration $\nu_2^f \in VFConfig(s)$, we can find $\nu_1^f \in VFConfig(r)$ that $\Pi(r, \nu_1^f) \sim \Pi(s, \nu_2^f)$.

## References


[1] K. Pohl, G. Bockle, F. van der Linden, Software Product Line Engineering: Foundations, Principles, and Techniques, Springer, 2005.
[2] K. Schmid, F. van der Linden, Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering, Springer, 2010.
[3] E. Engström, P. Runeson, Software product line testing – a systematic mapping study, Inf. Softw. Technol. 53 (1) (2011) 2–13.
[4] T. Thüm, S. Apel, C. Kästner, I. Schaefer, G. Saake, A classification and survey of analysis strategies for software product lines, ACM Comput. Surv. 47 (1) (2014) 6:1–6:45.
[5] P. Clements, L. Northrop, Software Product Lines: Practices and Patterns, Addison–Wesley, 2001.
[6] K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, Feature-oriented domain analysis (FODA) feasibility study, Tech. rep., Carnegie–Mellon University Software Engineering Institute, 1990, CMU/SEI-90-TR-21.
[7] R.C. van Ommering, Koala, a component model for consumer electronics product software, in: Proc. 2nd International Conference on Development and Evolution of Software Architectures for Product Families, in: LNCS, vol. 1429, Springer, 1998, pp. 76–86.
[8] E.M. Dashofy, A. van der Hoek, R.N. Taylor, An infrastructure for the rapid development of XML-based architecture description languages, in: Proc. 22nd International Conference on Software Engineering, ACM, 2002, pp. 266–276.
[9] K.G. Larsen, B. Thomsen, A modal process logic, in: Proc. 3rd Annual Symposium on Logic in Computer Science, IEEE, 1988, pp. 203–210.

[10] A. Classen, M. Cordy, P. Schobbens, P. Heymans, A. Legay, J.F. Raskin, Featured transition systems: foundations for verifying variability-intensive systems and their application to LTL model checking, IEEE Trans. Softw. Eng. 39 (8) (2013) 1069–1089.
[11] H. Beohar, M. Varshosaz, M.R. Mousavi, Basic behavioral models for software product lines: expressiveness and testing pre-orders, Sci. Comput. Program. (2015), http://dx.doi.org/10.1016/j.scico.2015.06.005, in press, available online.
[12] A. Gruler, M. Leucker, K.D. Scheidemann, Modeling and model checking software product lines, in: Proc. 10th International Conference on Formal Methods for Open Object-Based Distributed Systems, in: LNCS, vol. 5051, Springer, 2008, pp. 113–131.
[13] A. Gruler, M. Leucker, K.D. Scheidemann, Calculating and modeling common parts of software product lines, in: Proc. 12th International Software Product Line Conference, IEEE Computer Society, 2008, pp. 203–212.
[14] R. Milner, Communication and Concurrency, Prentice-Hall, 1989.
[15] D. Park, Concurrency and automata on infinite sequences, in: Theoretical Computer Science, 5th GI-Conference, Proceedings, Karlsruhe, Germany, March 23–25, 1981, in: LNCS, vol. 104, Springer, 1981, pp. 167–183.
[16] T. Belder, M.H. ter Beek, E.P. de Vink, Coherent branching feature bisimulation, in: Proc. 6th Workshop on Formal Methods and Analysis in SPL Engineering, in: EPTCS, vol. 182, 2015, pp. 14–30.
[17] J. Baeten, M. Bravetti, A ground-complete axiomatization of finite state processes in process algebra, in: Proc. 16th International Conference on Concurrency Theory, in: LNCS, vol. 3653, Springer, 2005, pp. 248–262.
[18] J. Bergstra, J.W. Klop, Algebra of communicating processes with abstraction, Theor. Comput. Sci. 37 (1985) 21–77.
[19] R. van Glabbeek, The Linear Time-Branching Time Spectrum I: The Semantics of Concrete, Sequential Processes, Handbook of Process Algebra, Elsevier, 2001, pp. 3–99.
[20] S. Christensen, H. Hüttel, C. Stirling, Bisimulation equivalence is decidable for all context-free processes, Inf. Comput. 121 (2) (1995) 143–148.
[21] J. Baeten, T. Basten, M. Reniers, Process Algebra: Equational Theories of Communicating Processes, Cambridge University Press, 2010.
[22] R. van Glabbeek, A complete axiomatization for branching bisimulation congruence of finite-state behaviours, in: Proc. 18th Symposium on Mathematical Foundations of Computer Science, in: LNCS, vol. 711, Springer, 1993, pp. 473–484.
[23] J. Bergstra, J.W. Klop, Process algebra for synchronous communication, Inf. Control 60 (1–3) (1984) 109–137.
[24] J.F. Groote, A. Mathijssen, M. Reniers, Y. Usenko, M. van Weerdenburg, The formal specification language mCRL2, in: Proc. Methods for Modelling Software Systems, in: Dagstuhl Seminar Proceedings, vol. 06351, Schloss Dagstuhl, 2006.
[25] J.F. Groote, M.R. Mousavi, Modeling and Analysis of Communicating Systems, MIT Press, 2014.
[26] S. Blom, Partial $\tau$-confluence for efficient state space generation, Tech. rep., CWI, 2001.
[27] Y. Usenko, Linearization in $\mu$CRL, Ph.D. thesis, Eindhoven University of Technology, 2002.
[28] A. Classen, Modelling with FTS: a collection of illustrative examples, Tech. rep., P-CS-TR SPLMC-00000001, PReCISE, Research Center, University of Namur, 2010.
[29] A. Classen, P. Heymans, P. Schobbens, A. Legay, J. Raskin, Model checking lots of systems: efficient verification of temporal properties in software product lines, in: Proc. 32nd International Conference on Software Engineering, ACM, 2010, pp. 335–344.
[30] D. Kozen, Results on the propositional $\mu$-calculus, Theor. Comput. Sci. 27 (1983) 333–354.
[31] S. Shoham, O. Grumberg, Multi-valued model checking games, J. Comput. Syst. Sci. 78 (2) (2012) 414–429.
[32] A. Fantechi, S. Gnesi, Formal modeling for product line families engineering, in: Proc. 12th International Conference on Software Product Lines, IEEE Computer Society, 2008, pp. 193–202.
[33] W. van der Aalst, M. Dumas, F. Gottschalk, A. ter Hofstede, M. Rosa, J. Mendling, Correctness-preserving configuration of business process models, in: Proc. 11th International Conference on Fundamental Approaches to Software Engineering, in: LNCS, vol. 4961, Springer, 2008, pp. 46–61.
[34] K. Czarnecki, M. Antkiewicz, Mapping features to models: a template approach based on superimposed variants, in: Proc. 4th International Conference on Generative Programming and Component Engineering, in: LNCS, vol. 3676, Springer, 2005, pp. 422–437.
[35] T. Ziadi, L. Hélouët, J.-M. Jézéquel, Towards a UML profile for software product lines, in: Proc. 5th International Workshop on Product-Family Engineering, in: LNCS, vol. 3014, Springer, 2003, pp. 129–139.
[36] D. Fischbein, S. Uchitel, V. Braberman, A foundation for behavioral conformance in software product line architectures, in: Proc. Workshop on Role of Software Architecture for Testing and Analysis, ACM, 2006, pp. 39–48.
[37] K. Larsen, U. Nyman, A. Wasowski, Modal I/O automata for interface and product line theories, in: Proc. 16th European Symposium on Programming Languages and Systems, in: LNCS, vol. 4421, Springer, 2007, pp. 64–79.
[38] J. Liu, J. Dehlinger, R. Lutz, Safety analysis of software product lines using state-based modeling, J. Syst. Softw. 80 (11) (2007) 1879–1892.
[39] K. Lauenroth, S. Thning, K. Pohl, Model checking of domain artifacts in product line engineering, in: Proc. 24th IEEE/ACM International Conference on Automated Software Engineering, IEEE Computer Society, 2009, pp. 269–280.
[40] P. Asirelli, M. ter Beek, A. Fantechi, S. Gnesi, A logical framework to deal with variability, in: Proc. 8th International Conference on Integrated Formal Methods, in: LNCS, vol. 6396, 2010, pp. 43–58.
[41] S. Gnesi, M. Petrocchi, Towards an executable algebra for product lines, in: Proc. 16th International Software Product Line Conference, ACM, 2012, pp. 66–73.
[42] M. ter Beek, A. Lluch-Lafuente, M. Petrocchi, Combining declarative and procedural views in the specification and analysis of product families, in: Proc. 17th International Software Product Line Conference, ACM, 2013, pp. 10–17.
[43] P. Höfner, R. Khédri, B. Möller, An algebra of product families, Softw. Syst. Model. 10 (2) (2011) 161–182.
[44] M. Tribastone, Behavioral relations in a process algebra for variants, in: Proc. 18th International Software Product Line Conference, ACM, 2014, pp. 82–91.
[45] M. ter Beek, E. de Vink, Using mCRL2 for the analysis of software product lines, in: Proc. 2nd Workshop on Formal Methods in Software Engineering, ACM, 2014, pp. 31–37.
[46] M. Leucker, D. Thoma, A formal approach to software product families, in: Proc. 5th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change, in: LNCS, vol. 7609, Springer, 2012, pp. 131–145.
[47] H. Beohar, M.R. Mousavi, Input–output conformance testing based on featured transition systems, in: Proc. Symposium on Applied Computing, Software Verification and Testing Track, ACM, 2014, pp. 1272–1278.
[48] H. Beohar, M.R. Mousavi, Spinal test suites for software product lines, in: Proc. Workshop on Model-Based Testing (MBT 2014), in: EPTCS, vol. 141, 2014, pp. 44–55.
[49] M. Cordy, A. Classen, G. Perrouin, P. Schobbens, P. Heymans, A. Legay, Simulation-based abstractions for software product-line model checking, in: Proc. 34th International Conference on Software Engineering, IEEE, 2012, pp. 672–682.
[50] M. Bernardo, A. Aldini, F. Corradini, A Process Algebraic Approach to Software Architecture Design, Springer, 2010.
[51] H.R. Andersen, C. Stirling, G. Winskel, A compositional proof system for the modal $\mu$-calculus, in: Proc. 9th Annual Symposium on Logic in Computer Science, IEEE Computer Society, 1994, pp. 144–153.
[52] L. Aceto, A. Ingólfsdóttir, K. Larsen, J. Srba, Reactive Systems: Modelling, Specification and Verification, Cambridge University Press, 2007.